Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Felix Max Martin Hahn

Entwicklung und Aufbau eines prototypischen Bildschirmlesegerätes auf Basis eines Einplatinencomputers

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Felix Max Martin Hahn

Entwicklung und Aufbau eines prototypischen Bildschirmlesegerätes auf Basis eines Einplatinencomputers

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Henning Dierks
Zweitgutachter : Prof. Dr. rer. nat. Annabella Rauscher-Scheibe

Abgegeben am 4. April 2016

Felix Max Martin Hahn

Thema der Bachelorthesis

Entwicklung und Aufbau eines prototypischen Bildschirmlesegerätes auf Basis eines Einplatinencomputers

Stichworte

Einplatinencomputer, Raspberry Pi, Texterkennung, Tesseract, OCR, Bildverarbeitung, OpenCV, C++

Kurzzusammenfassung

In dieser Arbeit geht es um die Entwicklung eines Bildschirmlesegerätes auf Basis eines aktuellen Einplatinencomputers. Ein Bildschirmlesegerät ist eine Kombination von Kamerasystem und Monitor. Es ist eine Lesehilfe für Menschen mit eingeschränktem Sehvermögen. Während auf dem Markt befindliche Geräte meist über eine Kamera mit großem optischem Zoom verfügen, wird hier ein anderer Ansatz verfolgt. Im Rahmen dieser Arbeit wird untersucht, in wieweit sich mit einem Foto eines Textes eine Zeichenerkennung auf einem modernen Einplatinencomputer durchführen lässt. Ein erkannter Text kann beliebig skaliert und auch unabhängig vom Layout des ursprünglichen Textes angezeigt werden.

Felix Max Martin Hahn

Title of the paper

Development and construction of a prototypic video magnifier based on a single-board computer

Keywords

Single-board computer, SBC, Raspberry Pi, Optical character recognition, Tesseract, OCR, Image processing, OpenCV, C++

Abstract

This thesis is about the development of a video magnifier based on a current single-board computer. A video magnifier is a combination of a camera system and a monitor. It is a reading aid for people with low vision. While devices on the market usually have a camera with a large optical zoom, a different approach is pursued here. In this thesis it is examined to what extent optical character recognition can be performed on a modern single-board computer with a picture of the text. A recognized text can be scaled and displayed regardless of the layout of the original text.

Danksagung

Danke an meine Frau, die immer für mich da war.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

1. Einführung

In dieser Arbeit wird ein Bildschirmlesegerät auf Basis eines Einplatinencomputers entwickelt. Im Gegensatz zu herkömmlichen Bildschirmlesegeräten, die mit einer großen optischen Vergrößerung arbeiten, wird hier der Text mit einer Kamera aufgenommen und mit einer Zeichenerkennungssoftware digitalisiert.

1.1. Idee

Die Idee ist, den auf dem Bildschirm angezeigten Text vom physikalischen Text zu „entkoppeln“. Sobald der Text in digitaler Form vorliegt, lässt sich dieser an die Anforderungen des Benutzers anpassen. Dies gilt sowohl für die Größe des Textes, als auch für die Farbe oder die Schriftart. Das Layout des Textes kann auch geändert werden, soll aber in seiner Grundform beibehalten werden.
Im Dokument enthaltene Bilder sollen in dessen digitalen Abbild auch als solche angezeigt werden. Die Betrachtung von Bildern steht hier allerdings nicht im Vordergrund, sondern die Erhöhung der Lesbarkeit von Texten.
Die digitale Form des Textes bietet auch den Vorteil, dass die Bedienung des Lesegerätes nicht mehr an das physikalische Dokument gekoppelt ist. Es lassen sich also auch Bedienmöglichkeiten für Menschen realisieren, denen die Handhabung von Dokumenten erschwert ist (z. B. durch Tremor). Weitere Hilfsmittel wie ein integriertes Wörterbuch, eine Übersetzungsfunktion bei fremdsprachigen Texten oder eine einfache Vorlesefunktion sind denkbar.

1.2. Problemstellung

Die Neupreise für aktuelle Bildschirmlesegeräte liegen je nach Modell bei mindestens 1.500 bis über 5.000 Euro.[1] Die Frage ist, ob sich die Kosten auf einen Bruchteil dessen reduzieren lassen, indem man die aufwendige Kameratechnik durch eine einfache Kamera ersetzt,

[1] Quelle: Dütsch [2016] - Finanzierung

welche von einem Computer unterstützt wird. Dabei sollen keine Abstriche bei der Qualität der Textanzeige gemacht werden müssen.
Mit einer hochauflösenden Kamera und entsprechend leistungsfähiger Computerhardware ist es möglich, eine Bildverarbeitung und Texterkennung in Echtzeit durchzuführen. Allerdings hat leistungsstarke Computerhardware entsprechend hohe Preise und eine hohe Leistungsaufnahme. Seit etwa fünf Jahren werden kompakte Einplatinencomputer, meist mit ARM-Prozessoren, angeboten, deren Rechenleistung sich jährlich steigert. Die Preise für solche Geräte liegen meist deutlich unter 100 Euro (vgl. Abschnitt 4.1). Deshalb wird hier der Frage nachgegangen, ob ein moderner Einplatinencomputer in Kombination mit einer einfachen, ausreichend hoch auflösenden Kamera in der Lage ist, eine solche Aufgabe zu bewältigen.

1.3. Themenabgrenzung

Diese Arbeit befasst sich nicht mit der Entwicklung von Kameratreibern oder Texterkennungssoftware, sondern nutzt bekannte und etablierte Softwareprodukte oder Programmierschnittstellen, um daraus ein zusammenhängendes Programm zu erstellen. Diese eigene Implementierung wird mit einer Benutzeroberfläche versehen, welche einfache Eingaben und Ausgaben möglich macht.

1.4. Zielsetzung

Die reinen Hardwarekosten für diesen Prototypen sollen unter 300 Euro liegen. Die dafür nötige Software soll optimal an die Hardwareressourcen angepasst sein. Der Fokus liegt dabei auf einer lauffähigen Texterkennung mit angemessener Verarbeitungszeit. Zusätzlich soll untersucht werden, inwiefern mit einer längeren Rechenzeit das Ergebnis verbessert werden kann.

1.5. Struktur der Arbeit

Diese Arbeit ist in acht Kapitel aufgeteilt. Das erste Kapitel umfasst die Einführung, die mit diesem Abschnitt bereits abgeschlossen ist. In Kapitel 2 werden die Grundlagen dieser Arbeit beschrieben. Dabei geht es schwerpunktmäßig um die optische Zeichenerkennung. Kapitel 3 dokumentiert die Analyse der Anforderungen an die Hard- und Software. In Kapitel 4 erfolgt die Evaluation möglicher Kandidaten für die Hard- und Software dieser Arbeit. Jeweils

am Ende der einzelnen Abschnitte wird auf Grundlage der zuvor gestellten Anforderungen eine Auswahl getroffen wird. Die so ausgewählten Komponenten werden in Kapitel 5 in einen zusammenhängenden Entwurf gebracht. In Kapitel 6 werden die Umsetzung des Entwurfes und dessen Ergebnisse beschrieben. Mit dem dort umgesetzten Prototypen werden in Kapitel 7 Testläufe der Texterkennung durchgeführt. Im Schlussteil, Kapitel 8, werden die Kapitel 3 bis 7 noch einmal zusammengefasst. Anschließend wird ein persönliches Fazit gezogen und ein Ausblick für eine mögliche Weiterentwicklung gegeben.
Danach folgen zwei getrennte Verzeichnisse für Literaturquellen und Internetquellen, welche auch innerhalb der Arbeit durch die Zitierweise deutlich zu unterscheiden sind. Im Anhang A befindet sich der Überblick über den Inhalt der CD, welche auf der letzten Seite der Arbeit eingeklebt ist. Diese enthält die verwendeten Internetquellen, sowie den vollständigen Quelltext der geschriebenen Software. Im weiteren Anhang befinden in den Kapiteln referenzierte Abbildungen.

2. Grundlagen

Um ein besseres Verständnis für die in dieser Arbeit verwendeten Begrifflichkeiten zu schaffen, werden in diesem Kapitel grundlegende Themen aufgegriffen und erklärt.

2.1. Bildschirmlesegerät

Wie in der Einleitung beschrieben, handelt es sich bei Bildschirmlesegeräten um eine Kombination von Kamerasystem und Monitor. Das vor der Kamera befindliche (kleine) Objekt wird auf dem Monitor vergrößert dargestellt (vgl. Abbildung 2.1). Diese werden von Menschen verwendet, denen das Lesen mit einer optischen Sehhilfe (Lesebrille, Lupe) nicht mehr möglich ist. Dies ist der Fall, wenn mehr als eine achtfache Vergrößerung nötig ist.[1] Um das Lesen vom Bildschirm zu verbessern, sind Schrift- und Hintergrundfarbe für einen idealen Kontrast einstellbar. Zum Zeilen- oder Seitenwechsel wird das Schriftstück von Hand unter der Kamera bewegt.

2.2. Einplatinencomputer

Als Einplatinencomputer bezeichnet man ein Computersystem, dessen Komponenten auf einer einzelnen Platine vereint sind. Ein zum Betrieb notwendiges Netzteil wird üblicherweise extern angebracht. Einplatinencomputer sind dabei keine Erfindung des 21. Jahrhunderts, da die ersten Heimcomputer, im Gegensatz zu heutigen Heimcomputern, nicht aus einzelnen Komponenten zusammengesetzt wurden. In der Industrie ersetzen sie festverdrahtete Steuerungen, um eine Softwaresteuerung möglich zu machen. In Form von eingebetteten Systemen befinden sie sich seit den 90er Jahren in Kraftfahrzeugen, Haushaltsgeräten und nahezu allen elektrischen Geräten wie Routern, Receivern oder Fernsehern (SmartTV).
Moderne Einplatinencomputer sind meist mit einem System-on-a-Chip (SoC) bestückt, in dem neben einer leistungsfähigen CPU und GPU auch Standardschnittstellen wie USB oder Ethernet integriert sind. Modelle für den privaten Gebrauch sind heute (2016) für unter 100 Euro zu erwerben (vgl. Abschnitt 4.1).

[1] Quelle: Dütsch [2016]

Abbildung 2.1.: Produktbeispiel Bildschirmlesegerät: Hirzel Optik Topaz HD.
Quelle: Hirzel Optik [2014]

2.3. Optische Zeichenerkennung (OCR)

Bei optischer Zeichenerkennung (engl. „optical character recognition“, OCR) handelt es sich um die automatisierte Erkennung von Einzelzeichen. Das Kürzel OCR wird allerdings vielmals als Synonym für automatisierte Texterkennung verwendet. Da dies ein zentrales Thema dieser Arbeit ist, wird es im Folgenden detailliert erklärt.

2.3.1. Geschichte

Mit der Geschichte der Texterkennung wird mit dem Beginn der elektronischen Datenverarbeitung (EDV) angefangen. Zwar gab es schon zu Beginn des 20. Jahrhunderts Erfindungen, welche das Erkennen von Zahlen oder Buchstaben mit Maschinen ermöglichten, allerdings sind diese für die hier verwendete Texterkennung nicht relevant.
Mit dem Ende der Lochrasterkarten und dem Aufkommen der elektronischen Datenverarbeitung Anfang der 1960er Jahre stieg die Nachfrage nach der Möglichkeit, Druckvorlagen automatisiert einzulesen. Um die Erkennung einfach und sicher zu gestalten, veröffentlichte die American National Standards Institute (ANSI) 1968 die Normschrift OCR-A (ANSI INCITS 17-1981)[2] (vgl. Abbildung 2.2). Diese zeichnet sich durch einfache und deutlich

[2] Quelle: Beinert [2015]

unterscheidbare Zeichen aus und ist damit auf das Einlesen durch Maschinen optimiert. Zusätzlich enthält sie Steuerzeichen, welche der Texterkennungssoftware beim Erkennen der Formatierung helfen. Für den Menschen ist die Schriftart schwieriger zu lesen.

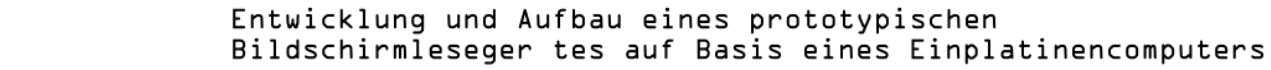


Abbildung 2.2.: Textbeispiel der Schriftart OCR-A (TrueType-Version von Matthew Skala), Umlaute fehlen im Zeichensatz

Im Jahr 1973 entwickelte der Schweizer Schriftgestalter Adrian Frutiger für die US-amerikanische Druckfirma Monotype Imaging eine Schriftart nach Standard der European Computer Manufacturers Association (ECMA, seit 1994 Ecma International) (vgl. Abbildung 2.3). Diese trägt den Namen OCR-B damit der direkte Nachfolger von OCR-A und wurde noch im selben Jahr weltweiter Standard (ISO 1073-2).[3] Sie ist auch für den Menschen einfach zu lesen.

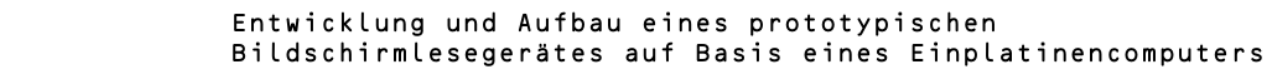


Abbildung 2.3.: Textbeispiel der Schriftart OCR-B (TrueType-Version von Matthew Skala)

Heute wird OCR-B beispielsweise für Ziffern bei Strichcodes (Barcodes) verwendet. Die Postbank verwendet auf ihren Vordrucken beispielsweise für die Scheckeinreichung auch noch OCR-A. Ansonsten haben diese Schriftarten ihre Bedeutung verloren. Moderne OCR-Software soll schriftunabhängig arbeiten.

2.3.2. Anwendung

Anfangs wurde die Texterkennung eingesetzt, um erstellte Ausdrucke andernorts wieder einlesen zu können. Mit fortschreitender Entwicklung im Bereich der EDV und der Texterkennung wurde diese zur Digitalisierung von Dokumenten eingesetzt. Damit hatte der Mensch nur noch die Aufgabe der Korrektur, statt wie vorher das (auch nicht korrekturfreie) Abtippen der Dokumente. Durch schnellere Computersysteme und hochauflösende Scanner oder Kameras erweiterte sich das Anwendungsfeld. Heutige Anwendungen sind z. B.:

- Sortierung von Briefen (nach PLZ, etc.) bei der Post
- Kennzeichenerkennung zur Verkehrsüberwachung oder für Maut
- Einlesen von ausgefüllten Vordrucken
- Digitalisierung bereits gescannter (historischer) Dokumente

[3] Quelle: Beinert [2015]

2.3.3. Verfahren

Die vier Verfahren zur Texterkennung nach Kampffmeyer u. a. [1994, S. 16ff.] werden hier kurz erklärt. Dabei handelt es sich bei der ersten Methode um einen Mustervergleich, während die anderen drei Methoden zu den topologischen Verfahren gehören. Diese sind toleranter gegenüber Druckfehlern oder Verunreinigungen, da sie charakteristische Eigenschaften der Zeichen nutzen. Die Grafiken wurden ebenfalls aus der genannten Quelle übernommen.

Pixelmustervergleich

Die Methode des Pixelmustervergleiches („Pattern matching“) ist die älteste und einfachste Methode zur Zeichenerkennung. Durch das Abtasten eines Textes mit einem Scanner (vgl. Abbildung 2.4) entsteht aus den Buchstaben ein sog. Pixelmuster. Dieses Muster wird dann mit Standardmustern aus einer auf dem Computer installierten Datenbank verglichen. Da es, bedingt durch die Abtastung, niemals zu einer 100%igen Übereinstimmung der beiden Muster kommen kann, können für jedes Zeichen in der Datenbank bestimmte Toleranzbereiche festgelegt werden.
Texterkennungssoftware, welche den Pixelmustervergleich nutzt, ist in der Regel lernfähig. Das heißt die Erkennungsrate kann durch das Trainieren einer bestimmten Schriftart gesteigert werden. Wird bei den zu erkennenden Texten ausschließlich eine gut trainierte Schriftart genutzt, ist der Pixelmustervergleich gut zur Erkennung geeignet.
Bei einem Wechsel der Schriftart innerhalb eines Textes oder wenn diese vor der Erkennung nicht bekannt ist, ist es erforderlich, dass die Datenbank der Standardmuster alle gängigen Schriftarten beinhaltet.

Vektorverfahren

Im Vektorverfahren wird kein 1:1 Vergleich der Pixelmuster durchgeführt, sondern es werden aus dem Bild für das jeweilige Zeichen charakteristische Eigenschaften abgeleitet. Dazu zählen etwa im Zeichen enthaltene Geraden und Kreisbögen. Diese werden als Vektoren dargestellt und mit einer auf dem Computer installierten Datenbank mit Vektorbildern verglichen.
Man unterscheidet zwischen drei Vektor-basierten Verfahren: Mit einfachen Richtungsvektoren arbeitende Verfahren, mit Abweichungen vom Basis-Richtungsvektor (sog. „Fächer“) arbeitende Verfahren und mit den Räumen um die Vektoren (sog. „Wolken“) arbeitende Verfahren (vgl. Abbildung 2.5).
Vorteil dieser Methode ist es, dass sie unabhängig von der Darstellung der Zeichen (Schriftart) funktioniert. Allerdings ist die Zuordnung gerade bei der Wolken-Methode nicht immer

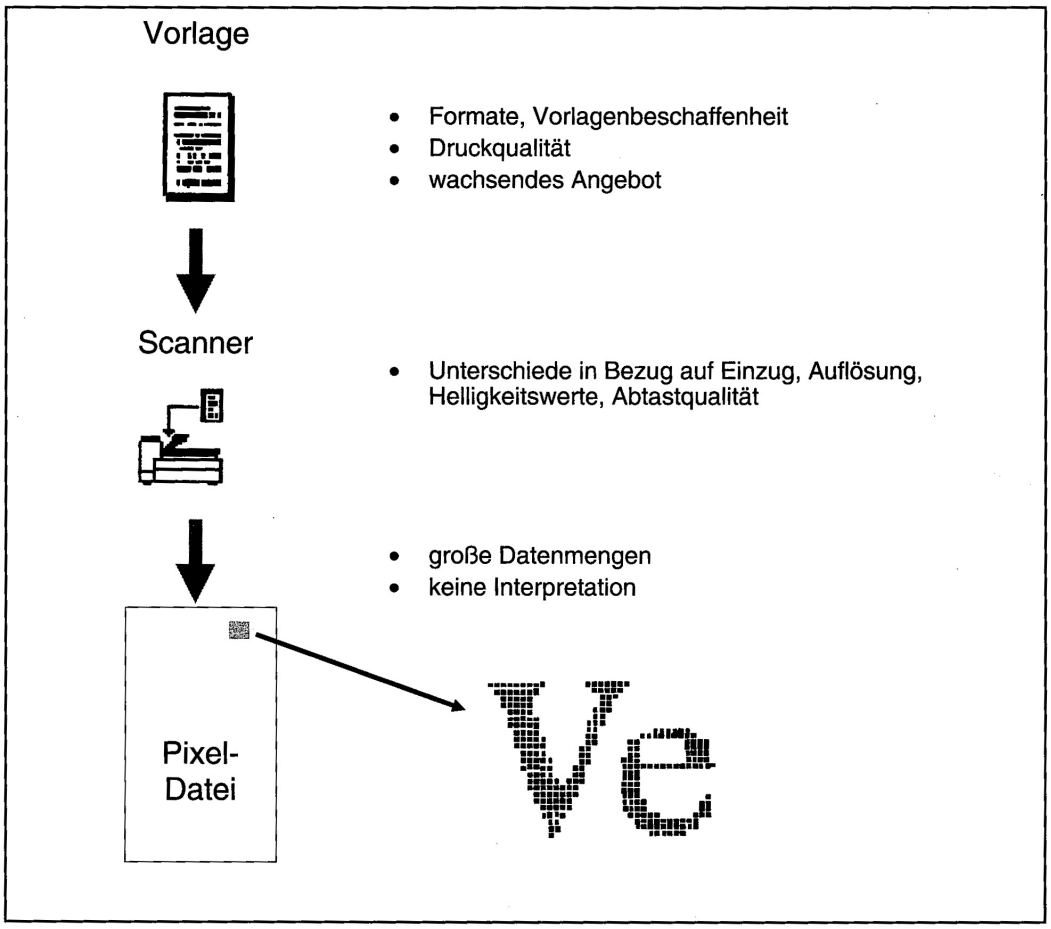


Abbildung 2.4.: Erzeugung einer Rastergrafik aus einem Dokument: Die Buchstaben werden als Pixelmuster digitalisiert. Quelle: [Kampffmeyer u. a., 1994]

eindeutig, was durch eine große Datenbank (z. B. durch sprachabhängige Zeichen oder Sonderzeichen) noch verstärkt wird.

Konturcodeverfahren

Das Konturcodeverfahren wird in der industriellen Bildverarbeitung vielseitig eingesetzt. Neben Flächen und Abbildungen, wie z. B. Firmenlogos, können auch komplexe Zeichen durch einen Konturcode (vgl. Abbildung 2.6) beschrieben werden. Die Objektkanten werden dabei durch einen Richtungscode (je nach Genauigkeit 0 bis 7 oder auch 0 bis 15) beschrieben. Angefangen in der linken oberen Kante, ergibt sich ein Zahlencode, der das Zeichen beschreibt. Durch die Anzahl der Schritte in eine Richtung ergibt sich die relative Größe der jeweiligen Kante. Dieser Zahlencode wird wieder mit dem sog. Grundmuster der Buchsta-

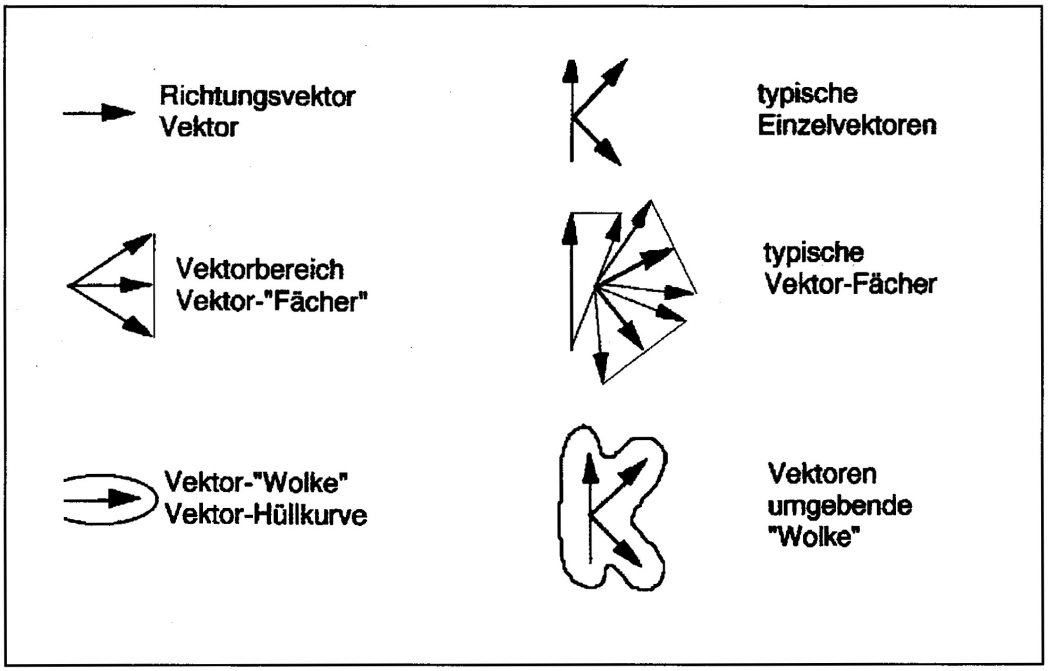


Abbildung 2.5.: Beispiele der drei Vektorverfahren: Links allgemein, rechts für den Buchstaben „K“. Quelle: [Kampffmeyer u. a., 1994]

ben in einer Datenbank abgeglichen.
Der Vorteil ist hier wie beim Vektorverfahren die größere Toleranz und zusätzlich eine Erkennung der Orientierung. Werden die Zeichen nicht erkannt, so kann durch Drehung des Richtungssternes bzw. durch Hinzufügen von „1“ zum Zahlencode (bei dem gezeigten Stern mit acht Richtungen dann mit Überlauf nach der 7) und erneutem Abgleich mit der Datenbank, auch ein gedrehtes Zeichen erkannt werden. Damit ist das Konturcodeverfahren zwar flexibler als das Vektorverfahren, aber auch rechenaufwendiger und somit langsamer.

Klassifikationsverfahren

Bei den Klassifikationsverfahren werden die Zeichen mit verschiedenen Methoden in mehrere Bestandteile zerlegt und diese als Merkmale des Zeichens klassifiziert. Die Erkennung des Zeichens erfolgt mit statistischen Methoden.
Als Beispiel für dieses Verfahren dient hier die sog. Winkelschnittanalyse (WSA). Dabei wird über das zu erkennende Einzelzeichen in verschiedenen Winkeln (0 bis 165 Grad in 15 Grad Schritten) eine Geradenschar (Abstand zwischen den Geraden ca. 0,1 mm) gelegt. Für jeden Winkel werden nun die Schwarzpunkte (Punkte der Geraden auf dem Zeichen) und die Schnittpunkte der Geraden mit dem Zeichen ermittelt (vgl. Abbildung 2.7). Aus diesen Werten werden Funktionen abgeleitet, welche die charakteristischen Merkmale der Zeichen dar-

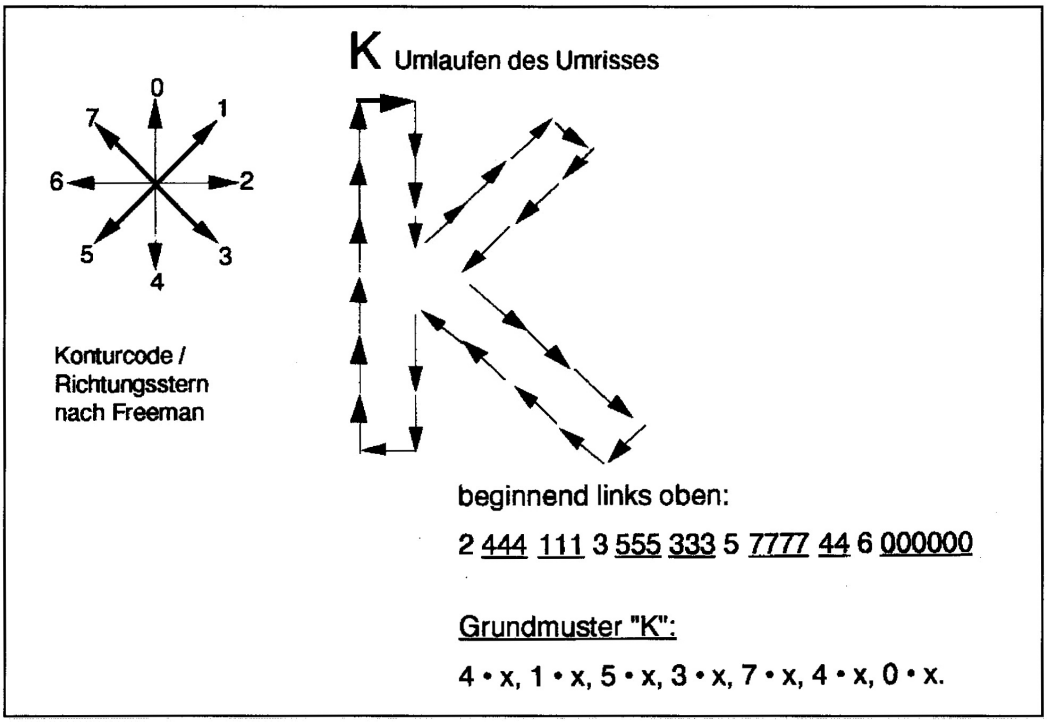


Abbildung 2.6.: Beim Konturcodeverfahren wird der Umriss eines Zeichens umfahren und als Zahlencode dargestellt. Die Anzahl der Richtungsschritte stellt dabei die relative Größe dar, ist aber nicht weiter von Bedeutung, da jede Richtung mit mindestens „x“ (im Beispiel ist x = 2) Richtungsschritten als Hauptrichtung (unterstrichen) festgelegt wird. Das Grundmuster zeigt die Reihenfolge der Hauptrichtungen des Buchstaben. Quelle: [Kampffmeyer u. a., 1994]

stellen. Dieses Verfahren ist zwar relativ aufwendig, bietet aber gute Ergebnisse bei schneller Erkennung.

Zusammenfassung

Die einzelnen Methoden bieten verschiedene Vor- und Nachteile in Bezug auf Genauigkeit, Geschwindigkeit und Anwendbarkeit auf verschiedene Schriftarten. Moderne Texterkennungssoftware nutzt meist eine Kombination der Verfahren und wertet diese statistisch aus. So kann bei bei geringer Erkennungswahrscheinlichkeit ein anderes Verfahren eingesetzt werden und mit dessen Ergebnis die Erkennungswahrscheinlichkeit und insgesamt auch die Texterkennungsrate erhöht werden.

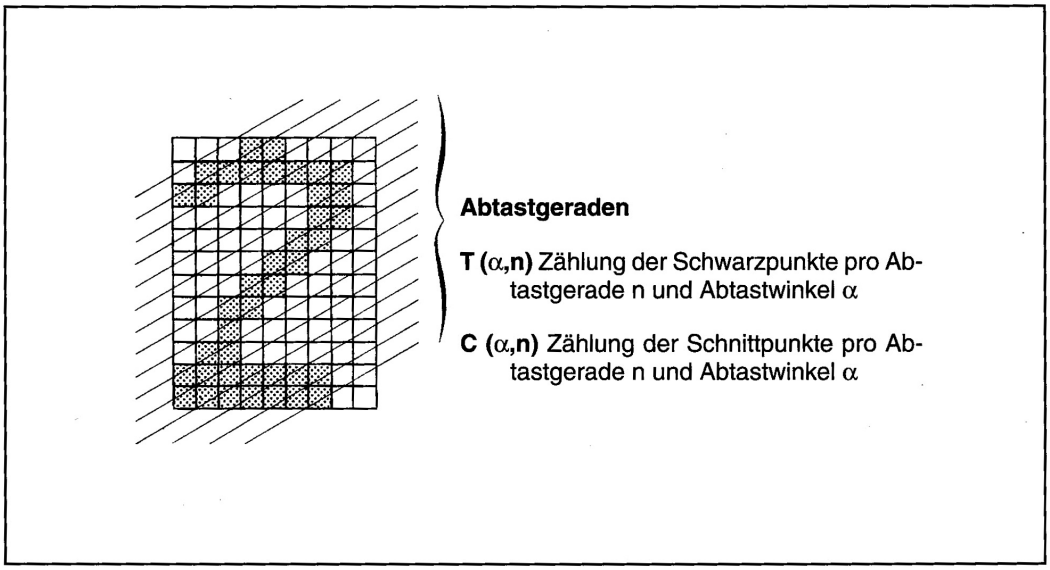


Abbildung 2.7.: Klassifikationsverfahren: Eine Geradenschar bei der Winkelschnittanalyse (WSA) über dem Pixelmuster einer „2“. Quelle: [Kampffmeyer u. a., 1994]

2.3.4. Grenzen der Texterkennung

Für den effektiven Einsatz der Texterkennung muss die Erkennungsrate entsprechend hoch sein. Die Anforderungen an eine nutzbare Texterkennung liegen bei einer Erkennungsrate von über 99%. Hat eine bedruckte Seite DIN A4 durchschnittlich 30 Zeilen mit 80 Zeichen, insgesamt also 2400 Zeichen, so gibt es bei 99%iger Erkennungsrate noch 24 Fehler pro Seite und damit fast einen Fehler pro Zeile. Diese lassen sich durch entsprechende Methoden teilweise beheben. Texterkennung mit anschließender Korrektur wird auch intelligente Zeichenerkennung (engl. „intelligent character recognition“ (ICR)) genannt.
Kampffmeyer u. a. [1994, S. 25f.] beschreibt die Grenzen der OCR/ICR-Methoden unter verschiedenen Gesichtspunkten. Abgesehen von der damals, im Vergleich zu heute, sehr geringen Rechenleistung von Computersystemen und den darauf zurückzuführenden Einschränkungen, werden dort die sog. „Problembuchstaben“ beschrieben. Dabei handelt es sich um Buchstaben oder Buchstabenkombinationen, welche sich nicht eindeutig erkennen lassen. Dazu zählen:

Schriftzeichen mit geringen Unterscheidungsmerkmalen

Hierzu zählen Zeichen, welche sich optisch sehr ähnlich sind. Dazu zählen Beispiele wie „I“ (großes „i“), „l“ (kleines „L“), „|“ (Senkrechter Strich) und „1“ (Eins) oder „0“ (Null) und „O“ (großes „o“). Je nach Schriftart sind auch andere Ähnlichkeiten möglich.

Ligaturen

Bei Ligaturen handelt es sich um untereinander verbundene Zeichenketten. Sie dienen zur Schließung optischer Lücken in einem Text und sollen die Lesbarkeit verbessern. In deutschen Texten häufig verwendete Ligaturen sind z.B. „ff“, „fi“, „fl“ und „ft“. Verbundene Zeichen können aber auch durch Druckfehler oder Verunreinigungen entstehen.
Bei der Texterkennung werden Ligaturen als ein einziges Zeichen erkannt. Je nach Software können die Buchstaben aufgetrennt werden oder gängige Ligaturen trainiert werden.

Unter- und Oberlängen

Als Unterlänge bezeichnet man den Teil eines Kleinbuchstabens, welcher unter die sog. Grundlinie eines Textes ragt (DIN 16507-2), z. B. bei dem Buchstaben „g“. Als Oberlänge bezeichnet den Teil eines Kleinbuchstabens, welcher über die Mittellinie nach oben ragt, z. B. bei dem Buchstaben „f“. Dabei ist ein Kleinbuchstabe mit Oberlänge meist höher als die Versalhöhe (Höhe eines Großbuchstaben von der Grundlinie aus).
Durch Unter- und Oberlängen kann es dazu kommen, dass aufeinanderfolgende Zeilen nicht mehr klar zu trennen sind, weil sich die Unterlänge der oberen und die Oberlänge der unteren Zeile überschneiden.

Kombinierte Zeichen

Kombinierte Zeichen sind Zeichen, welche aus mehreren, sich nicht berührenden Teilen zusammensetzt sind. Dazu zählen neben „i“, „j“ und den Umlauten auch Zeichen mit Akzent. Je nach Schriftart gibt es hier verschiedene Fehlerquellen. So erhöht sich die Ähnlichkeit zu anderen Zeichen, zusätzlich gibt es eine große Ähnlichkeit bei demselben Buchstaben mit verschiedenen Akzenten, wie z. B. „ô“, „ó“, „ò“, „õ“ und „ö“.

Unterschneidungen

Unterschneidungen sind das, was durch Ligaturen vermieden wird. Dabei handelt es sich um aufeinander folgende Buchstaben, zwischen denen keine senkrechte weiße Linie gezogen werden kann. Bei kursiven Texten sind Unterschneidungen kaum zu vermeiden, z. B. zwischen „f“ und „g“ in „*Aufgabe*“. Auch hier muss die Erkennungs-Software die Buchstaben korrekt auftrennen können.

Zusammenfassung

Die beschriebenen Grenzen der Texterkennung sind auch nach 25 Jahren noch aktuell. Durch die heute deutlich höhere Zahl von (teils ausgefallenen) Schriftarten verstärken sich die Probleme noch, da gerade im Anwendungsfall dieser Arbeit ein Trainieren auf bestimmte Schriftarten nicht sinnvoll ist. Viele Fehler können durch Korrektur mit einem Wörterbuch behoben werden. Allerdings kann ein zu wenige Wörter umfassendes Wörterbuch falsche Fehler verursachen und ein zu großes Wörterbuch (gerade bei mehrsprachigen Texten) kaum Fehler beheben.

2.3.5. Moderne Verbesserungsansätze

Durch die fortschreitende technische Entwicklung gibt es immer wieder neue Ideen zur Verbesserung von Texterkennung. Zwei Methoden des 21. Jahrhunderts werden hier kurz erklärt.

Künstliche neuronale Netzwerke

Künstliche neuronale Netzwerke sind ein Teilgebiet der künstlichen Intelligenz. Sie dienen zur Abstraktion von Informationsverarbeitung und sind lernfähig. So gewannen tiefe vorwärtsgerichtete neuronale (Long short-term memory, LSTM-) Netzwerke zwischen 2009 und 2012 drei Wettbewerbe zur Erkennung von Handschrift. Diese LSTM-Netzwerke lernen dabei gleichzeitig die Segmentierung und die Erkennung. Erkennungsraten von modernen Systemen mit mehreren Neutronenlagen übertreffen in bestimmten Szenarien (z.B. Verkehrszeichenerkennung) die des Menschen [Schmidhuber, 2012].

reCAPTCHA

Ein CAPTCHA (engl. Completely Automated Public Turing test to tell Computers and Humans Apart = „vollautomatischer öffentlicher Turing-Test zur Unterscheidung von Computern und Menschen“) wird von Internetseiten genutzt, um vor der Bereitstellung bestimmter Dienste festzustellen, ob die Anfrage von einem Menschen oder einer Maschine (eines sog. Bot) kommt. Dies soll verhindern, dass ein automatisierter Bot diese Dienste nutzt (z. B. Anlegen von Email-Adressen für Spam). Ein CAPTCHA besteht oft aus schlecht lesbaren Zeichen. Nach einer Studie der Carnegie Mellon University verbringen Internetnutzer täglich 150.000 Stunden damit CAPTCHAs zu lösen.[4]

[4] Quelle: Carnegie Mellon University [2008]

Da diese „Gehirnleistung“ nicht genutzt wurde, entwickelten Mitarbeiter der Universität reCAPTCHA. Es nutzt Bilder von Wörtern, die von einer Texterkennungssoftware nicht eindeutig (zu viele mögliche Treffer) erkannt werden. Mit dem von mehreren reCAPTCHA-Lösern eingegebenen Wort wird die Texterkennungssoftware dann trainiert. Diese Auslagerung von Rechenleistung nennt sich Crowdsourcing. Im Jahr 2009 hat Google reCAPTCHA gekauft und genutzt, um Google Books zu verbessern. Ab 2012 nutzte Google es für die Erkennung von Straßennamen und Hausnummern von Google Streetview. Seit Ende 2014 ist das System bei Google fast komplett abgelöst durch NoCAPTCHA reCAPTCHA, welches keine Eingabe des Nutzers mehr verlangt, sondern durch die Interaktion mit dem Browser zwischen Mensch und Maschine unterscheidet.

3. Analyse

In diesem Kapitel werden die Anforderungen an die einzelnen Komponenten von Hard- und Software gestellt. Es werden verschiedene Szenarien erdacht, aus denen die Forderungen abgeleitet werden können.

3.1. Hardware

Als Hardware werden in diesem Projekt ein Einplatinencomputer, eine Kamera, ein Monitor und eine entsprechende Halterung zum Einsatz kommen. Da die Steuerung eines Monitors (Helligkeit, Kontrast) ohne Eingreifen in dessen Steuerung (z. B. über eine Schnittstelle) selten möglich ist, werden an diesen keine gesonderten Anforderungen gestellt.

3.1.1. Anforderungen an den Einplatinencomputer

Der Einplatinencomputer ist der Mittelpunkt der Hardware dieser Arbeit, da dort alles zusammenläuft. Er interpretiert die Eingaben des Benutzers und steuert die Kamera an. Er führt die nötigen Berechnungen zur Texterkennung durch und gibt die Ergebnisse auf dem Monitor aus. Deshalb muss hier besonders auf große Kompatibilität und ausreichende Leistungsfähigkeit geachtet werden.

Betriebssystem

Seit 2015 gibt es neben diversen Linux-Derivaten auch ein Windows IoT (engl. „Internet of Things“ = „Internet der Dinge“) für Einplatinencomputer. Für diese Arbeit wird allerdings auf ein Linux-basiertes Betriebssystem gesetzt, da die Auswahl an verfügbaren Bibliotheken sehr groß ist.

Rechenleistung

Da die Software dieser Arbeit mehrere Dinge gleichzeitig berechnen soll, sollte der SoC des Einplatinencomputers einen Mehrkernprozessor beinhalten. Da Einplatinencomputer keine dedizierte Grafikeinheit mit eigenem Grafikspeicher (VRAM) haben, nutzt die Grafikeinheit des SoC einen Teil ihres Arbeitsspeichers (RAM) als VRAM. Je nach Anforderung sollten 128 bis 256 MB als VRAM genutzt werden. Damit noch ausreichend RAM für die hier entwickelte Software zur Verfügung steht, sollten dem SoC mindestens 512 MB RAM zur Verfügung stehen.

Schnittstellen

Als Schnittstelle ist, zum Anschluss von Peripherie, während der Entwicklung und zur späteren Bedienung, sowie für den Anschluss einer Kamera, USB notwendig. Für den Anschluss an einen Monitor sollte ein digitaler Bildausgang (DVI oder HDMI) vorhanden sein, damit das Bild möglichst scharf dargestellt werden kann. Für die Steuerung einer möglichen Ausleuchtung der Dokumentenablage sollten GPIO-Pins verfügbar sein.

Zusammenfassung

Folgende Eigenschaften muss der Einplatinencomputer aufweisen:

- Native Linux-Unterstützung
- Mehrkernprozessor
- 512 MB RAM
- USB-Anschluss
- Digitaler Bildausgang
- GPIO-Pins erreichbar

3.1.2. Anforderungen an die Kamera

Die Wahl der Kamera ist von entscheidender Bedeutung. Die Kamera liefert das Foto, welches die Software verarbeitet und schließlich zur Texterkennung nutzt. Um die Anforderungen aufzustellen müssen zunächst einige Randbedingungen festgelegt werden.

Auflösung

Die Auflösung einer Kamera bezeichnet die Anzahl der einzelnen Bildpunkte (Pixel), sowie das Seitenverhältnis. Die Auflösung alleine ist kein Merkmal für die Qualität des Bildes, dennoch wird sie als Hauptkriterium betrachtet. Je höher die Auflösung, um so mehr Daten müssen auch verarbeitet werden. Je geringer die Auflösung, desto schwieriger ist es für die Software, den fotografierten Text zu digitalisieren. Für eine ausreichend genaue Erkennung bei mittlerer Schriftgröße ist eine Punktdichte von 300 dpi (engl. „dots per inch“ = „Punkte pro Zoll“) notwendig [Kampffmeyer, 1996, S. 6]. Um daraus eine Mindestauflösung für die Kamera zu errechnen, muss festgelegt werden, welche Größe der, von der Kamera zu erfassende, Bildausschnitt aufweisen soll. Da es kein vorgeschriebenes Format für Bücher gibt, wird sich an der gängigen Größe für Taschenbücher orientiert. Üblicherweise ist eine Seite eines Taschenbuches kleiner als DIN A5 (148 mm$\times$210 mm). Dafür ergibt sich:

$$210\text{ mm} \cdot \frac{1\text{ inch}}{25{,}4\text{ mm}} = 8{,}268\text{ inch} \rightarrow 8{,}268\text{ inch} \cdot 300\text{ dpi} = 2.480\text{ dots}$$

$$148\text{ mm} \cdot \frac{1\text{ inch}}{25{,}4\text{ mm}} = 5{,}828\text{ inch} \rightarrow 5{,}828\text{ inch} \cdot 300\text{ dpi} = 1.748\text{ dots}$$

Somit lässt sich die Anzahl der nötigen Bildpunkte berechnen:

$$2.480 \cdot 1.748\text{ dots} = 4.335.040\text{ dots}$$

Für die Kamera ist folglich eine Auflösung von mindestens 4,3 Megapixel (MP) notwendig. Das DIN A5 Format entspricht in etwa dem Seitenverhältnis 4:3, welches auch für Kameras üblich ist. Eine maximale Auflösung wird nicht festgelegt, allerdings sollte die Kamera im Optimalfall nicht viel höher auflösen und nur wenig außerhalb des geforderten Ausschnittes aufnehmen.

Fokussierung

Die Fokussierung bezeichnet das Einstellen der Kamera auf eine bestimmte Bildentfernung. Um ausreichend Platz zum Umblättern eines Buches zu haben, wird die Höhe der Kamera auf etwa 20 cm über der Dokumentenablage festgelegt. Die Kamera muss auf diese geringe Entfernung fokussieren können. Eine automatische Fokussierung (Autofokus) ist nicht zwingend erforderlich, da sich die Entfernung kaum ändert. Die Linse der Kamera darf nicht einen so großen Winkel haben, dass bei einem Abstand von 20 cm die, unter Abschnitt 3.1.2 beschriebenen, Anforderungen nicht mehr eingehalten werden.

Kompatibilität

Um ein einfaches Anschließen an den Einplatinencomputer zu ermöglichen, sollte die Kamera einen USB-Anschluss bieten. Da ein Linux-basiertes Betriebssystem zum Einsatz kommt, muss der von der Kamera verwendete Chipsatz vom Kernel unterstützt werden.

Zusammenfassung

Folgende Eigenschaften muss die Kamera aufweisen:

- Mindestauflösung 4,3 MP
- Seitenverhältnis 4:3
- Fokussierbar auf 20 cm
- USB-Anschluss
- Linux-kompatibel

3.2. Software

Die Software ist der zentrale Inhalt dieser Arbeit. Sie steuert die Kamera, die Verarbeitung und die Ausgabe. Die hier gestellten Anforderungen sollen ein Zusammenspiel dieser drei Komponenten sicherstellen.
Während der spätere Ablauf der bereits mit vier einfachen Schritten (Foto aufnehmen $\rightarrow$ Bild für Erkennung vorbereiten $\rightarrow$ Text erkennen $\rightarrow$ Text ausgeben) beschrieben werden kann, so ist die Entwicklung einer lauffähigen Software aufwendig. In diesem Abschnitt werden zunächst abstrakte Anforderungen an die Software gestellt. Um diese aufzustellen, müssen zunächst die Stakeholder ermittelt werden. Dabei wird sich an den Methoden des Softwareengineering von Stephan Kleuker orientiert.

3.2.1. Stakeholder

Ein Stakeholder ist jemand, der Einfluss auf die Anforderungen hat, da er vom System betroffen ist (Systembetroffener) [Kleuker, 2008]. Die erste Aufgabe bei der Ermittlung der Anforderungen an das hier zu entwickelnde System ist die Ermittlung der Stakeholder. Da es sich hier um einen Prototypen handelt, reduziert sich die Anzahl der Stakeholder:

- Entwickler
- Tester
- Anwender

3.2.2. Ziele

Die Ziele werden mit der Schablone nach Kleuker [2008, S. 56] aufgestellt. Die Ziele müssen dabei verständlich und überprüfbar sein. Im wesentlichen wird sich hier auf zwei Ziele konzentriert, welche die Kernfunktionen des Prototypen darstellen: Vorschaubild und Texterkennung in angemessener Zeit. Die im zweiten Ziel angegebene Zeit soll dabei von der Aufnahme des Bildes bis zur Anzeige des Textes gelten.

Die in Tabelle 3.1 und Tabelle 3.2 dargestellten Ziele werden im späteren Verlauf der Arbeit wieder aufgegriffen und überprüft.

Ziel	Die Software muss das Bild der Kamera und den darauf erkannten Text anzeigen können
Stakeholder	Entwickler
Auswirkungen auf Stakeholder	Entwickler: Software muss entsprechende Komponenten ansteuern können.
Randbedingungen	Zeitliche Beschränkung (siehe Ziel 2).
Abhängigkeiten	-
Sonstiges	Das Layout des Textes soll beibehalten werden.

Tabelle 3.1.: Ziel 1: Funktion der Software

Ziel	Die Software soll zur Texterkennung einer DIN A5 Seite nicht länger als 30 Sekunden benötigen.
Stakeholder	Entwickler
Auswirkungen auf Stakeholder	Entwickler: Die Software muss die vorhandene Rechenleistung optimal nutzen.
Randbedingungen	Seite eines Taschenbuches mit deutschem Text. Erkennungsrate von 99%.
Abhängigkeiten	-
Sonstiges	Benötigte Rechenleistung ist nicht bekannt.

Tabelle 3.2.: Ziel 2: Benötigte Zeit

3.2.3. Use-Cases

Als Use-Cases bezeichnet man reale Benutzungsszenarien. Diese müssen vor der Entwicklung bekannt sein, um daraus die Anforderungen an die Software abzuleiten. Bei der Erstellung der Use-Cases geht es um die Beantwortung zentraler Fragen nach den Hauptaufgaben des Systems, der daran beteiligten Aktoren und die zur Erfüllung dieser Aufgaben nötigen Schritte. Dazu werden existierende Bildschirmlesegerätes analysiert und mit den in Abschnitt 1.1 beschriebenen Vorstellung verknüpft.

Zur vergleichbaren Beschreibung der folgenden Use-Cases wird eine Schablone nach Kleuker [2008, S. 63f.] genutzt.

Name des Use Case	1	Einschalten des Prototypen
Nummer	1	1
Paket	2	-
Autor	1	Felix Hahn
Version	1	1.0, 07.02.2016
Kurzbeschreibung	1	Der Prototyp wird durch Anlegen der Versorgungsspannung eingeschaltet, startet automatisch die Erkennungssoftware und zeigt ein Vorschaubild an.
beteiligte Aktoren (Stakeholder)	1	Entwickler, Anwender
Fachverantwortlicher	1	Felix Hahn
Referenzen	2	-
Vorbedingungen	2	Der Prototyp ist ausgeschaltet.
Nachbedingungen	2	Der Prototyp wird eingeschaltet und es wird ein Vorschaubild des Ablagefläche angezeigt (Preview-Mode).
typischer Ablauf	2	Das Gerät wird eingeschaltet.
alternative Abläufe	3	Das Gerät wird zurückgesetzt.
Kritikalität	3	hoch, das Vorschaubild dient zur Orientierung des Buches / Schriftstückes.
Verknüpfungen	3	-
funktionale Anforderungen	4	Die Software startet nach dem Hochfahren automatisch. Das Kamerabild muss in Echtzeit angezeigt werden.
nicht-funktionale Anforderungen	4	Die Spannungsversorgung muss verbunden sein. Der Monitor muss eingeschaltet sein.

Tabelle 3.3.: Use-Case 1: Einschalten des Prototypen

Der Use-Case 1: Einschalten des Prototypen (vgl. Tabelle 3.3) beschreibt die erste Interaktion eines Aktors mit dem Bildschirmlesegerät. Ein automatisches Starten bis zur Anzeige eines Vorschaubildes ist dabei für einen Anwender wichtig, um die Bedienung zu erleichtern.

Name des Use Case	1	Betrachten einer Seite eines Taschenbuches
Nummer	1	2
Paket	2	-
Autor	1	Felix Hahn
Version	1	1.0, 01.02.2016
Kurzbeschreibung	1	Es wird ein Buch auf die Ablage unter der Kamera gelegt und per Tastendruck die Texterkennung gestartet. Anschließend wird der erkannte Text angezeigt.
beteiligte Aktoren (Stakeholder)	1	Entwickler, Anwender
Fachverantwortlicher	1	Felix Hahn
Referenzen	2	-
Vorbedingungen	2	Der Prototyp ist eingeschaltet, die Software wurde gestartet und befindet sich im Preview-Mode.
Nachbedingungen	2	Der Text wird auf dem Monitor vergrößert dargestellt.
typischer Ablauf	2	Der Anwender legt ein aufgeschlagenes Buch unter das Bildschirmlesegerät und startet die Erkennung.
alternative Abläufe	3	Die Erkennung startet automatisch.
Kritikalität	3	sehr hoch, dies ist die Hauptfunktion.
Verknüpfungen	3	Use-Case 1 wird zuvor durchgeführt.
funktionale Anforderungen	4	Die Software muss den Text erkennen können. Die Software muss den erkannten Text anzeigen können.
nicht-funktionale Anforderungen	4	Die Spannungsversorgung muss verbunden sein. Der Monitor muss eingeschaltet sein. Die Beleuchtung muss ausreichend gut sein. Der Text muss in einer festgelegten Sprache geschrieben sein.

Tabelle 3.4.: Use-Case 2: Betrachten einer Seite eines Taschenbuches

Der Use-Case 2: Betrachten einer Seite eines Taschenbuches (vgl. Tabelle 3.4) beschreibt die übliche Benutzung eines Bildschirmlesegerätes und stellt damit die zentrale Funktion dieser Arbeit dar.

Name des Use Case	1	Testmodus
Nummer	1	3
Paket	2	-
Autor	1	Felix Hahn
Version	1	1.0, 07.02.2016
Kurzbeschreibung	1	Der Testtext wird in einer zu prüfenden Schriftart und -größe auf die Ablage gelegt und der Testmodus gestartet. Die Testergebnisse werden ausgegeben.
beteiligte Aktoren (Stakeholder)	1	Entwickler, Tester
Fachverantwortlicher	1	Felix Hahn
Referenzen	2	-
Vorbedingungen	2	Die Software befindet sich im Preview-Mode.
Nachbedingungen	2	Die Fehleranzahl wurde ausgegeben. Die Software befindet sich im Preview-Mode.
typischer Ablauf	2	Der Tester legt einen vorher festgelegten Text in das Sichtfeld der Kamera und startet einen Testlauf.
alternative Abläufe	3	-
Kritikalität	3	mittel, die Funktion wird für ein Bildschirmlesegerät nicht benötigt, sondern dient zu Testzwecken.
Verknüpfungen	3	Use-Case 1 wird zuvor durchgeführt.
funktionale Anforderungen	4	Kamerabild muss in Echtzeit angezeigt werden. Die Software muss den Text erkennen können. Der erkannte Text muss mit einem der Software bekannten Text verglichen werden. Die Ergebnisse müssen ausgegeben werden.
nicht-funktionale Anforderungen	4	Der Testtext muss alle gebräuchlichen Zeichen enthalten. Die Testergebnisse müssen vergleichbar sein. Die Spannungsversorgung muss verbunden sein. Der Monitor muss eingeschaltet sein. Die Beleuchtung muss ausreichend gut sein.

Tabelle 3.5.: Use-Case 3: Ausführen des Testmodus

Der Use-Case 3: Ausführen des Testmodus (vgl. Tabelle 3.5) beschreibt den sog. Testmodus. Dieser wird für eine spätere Überprüfung der aufgestellten Ziele benötigt, da ein manuelles Auszählen von Fehlern der Texterkennung zu aufwendig und fehleranfällig ist.

3.2.4. Funktionale Anforderungen

Zunächst werden alle in den Use-Cases formulierten funktionalen Anforderungen zusammengefasst:

1. Die Software startet nach dem Hochfahren automatisch.
2. Die Software muss das Kamerabild in Echtzeit anzeigen können.
3. Die Software muss den Text erkennen können.
4. Die Software muss den erkannten Text anzeigen können.
5. Die Software muss den erkannten Text vergleichen können.
6. Die Software muss Testergebnisse ausgeben können.

Anschließend werden diese nach Kleuker [2008, S. 71-77] nach drei Typen kategorisiert und entsprechend formuliert. Dabei steht die erste Zahl für den Use-Case in dem die Anforderung erstmals genannt wird.

Typ 1: Selbständige Systemaktivität

Eine selbstständige Systemaktivität ist ein Prozess, den das System selbstständig durchführt.

FA 1.1: Beim Start muss das System die Software im Preview-Mode starten.

FA 1.2: Im Preview-Mode soll das System das Kamerabild in Echtzeit darstellen

FA 2.1: Bei der Texterkennung muss das System den Text des Schriftstückes erkennen.

FA 2.2: Nach der Erkennung muss das System den erkannten Text anzeigen.

FA 3.1: Nach dem Starten des Testmodus soll das System in den erkannten Text vergleichen.

FA 3.2: Nach dem Beenden des Vergleiches soll das System die Testergebnisse ausgeben.

Typ 2: Benutzerinteraktion

Bei einer Benutzerinteraktion stellt das System dem Anwender die Prozessfunktionalität zur Verfügung.

FA 2.3: Im Preview-Mode muss das System dem Anwender die Möglichkeit bieten, die Texterkennung manuell zu starten.

FA 3.3: Im Preview-Mode muss das System dem Tester die Möglichkeit bieten, in den Testmodus zu wechseln.

Typ 3: Schnittstellenanforderungen

Bei den Schnittstellenanforderungen werden Prozesse beschrieben, die durch ein externes Ereignis gestartet werden. Dieser Typ kommt nicht vor.

3.2.5. Nicht-funktionale Anforderungen

Aus den Use-Cases werden auch die nicht-funktionalen Anforderungen abgeleitet [Kleuker, 2008, S. 77-81].

nFA 1: Die Spannungsversorgung muss verbunden sein.

nFA 2: Der Monitor muss eingeschaltet sein.

nFA 3: Die Beleuchtung muss ausreichend gut sein.

nFA 4: Der Text muss in einer festgelegten Sprache geschrieben sein.

nFA 5: Der Monitor muss mit dem Einplatinencomputer verbunden sein.

Zusätzlich werden noch folgende nicht-funktionale Anforderungen festgelegt:

nFA 6: Die Software muss unter einem Linux-basierten Betriebssystem lauffähig sein.

nFA 7: Die Software wird in C++ programmiert.

3.2.6. Randbedingungen

Für die Erstellung der Software gelten die folgenden Randbedingungen:

RB 1: Die Programmierung in C++14 ist ohne Einschränkungen möglich.

RB 2: Die Priorität liegt auf der Erkennung deutschsprachiger Texte.

3.2.7. Zusammenfassung

Aus den gestellten Anforderungen wird in den folgenden Kapiteln eine lauffähige Software entstehen. Die gesamte Softwareentwicklung muss dabei als iterativer Prozess (Inkrementelles Vorgehensmodell) betrachtet werden, wobei diese Arbeit den ersten Iterationsschritt mit einem ersten Prototypen darstellt. Mit dem daraus gewonnenen Erkenntnissen lassen sich neue Anforderungen für eine weitere Entwicklung stellen, was in Kapitel 8 dieser Arbeit auch im Ansatz passiert. Der Fokus liegt hier auf für ein Bildschirmlesegerät zentrale Funktionen, welche sich im gegebenen Zeitrahmen realisieren lassen.

4. Evaluation

In diesem Kapitel werden zu den gestellten Anforderungen mögliche Lösungen gesucht, gegenübergestellt und bewertet, um anschließend eine begründete Auswahl zu treffen.

4.1. Vergleich aktueller Einplatinencomputer

Durch detaillierte Recherche werden drei aktuelle Einplatinencomputer eruiert, welche im Folgenden kurz vorgestellt und nach einem anschließenden Vergleich bewerten werden.

4.1.1. Raspberry Pi 2 Model B

Der Raspberry Pi ist ein Einplatinencomputer von der Größe einer Kreditkarte. Entwickelt wurde er von der britischen Stiftung Raspberry Pi Fundation, um insbesondere jungen Leuten eine kostengünstige Plattform zur Aneignung von Programmierkenntnissen zu bieten.[1] Das hier vorgestellte Modell ist die im Februar 2015 erschienene zweite Generation des Raspberry Pi mit neuem SoC, das Raspberry Pi 2 Model B. Dieses weist folgende Spezifikationen auf:

Größe	85,0 mm×56,0 mm×17,0 mm (L×B×H)
SOC	Broadcom BCM2836
CPU	ARM Cortex-A7, 4×900MHz
GPU	Broadcom VideoCore IV
RAM	1GB LPDDR2-SDRAM
LAN	Microchip LAN9514, 10/100 MBit
IO	HDMI 1.4, 4×USB2.0, 40 GPIO-Pins

Tabelle 4.1.: Spezifikationen Raspberry Pi 2 Model B[2]

[1] Quelle: Belam [2012]
[2] Quelle: Embedded Linux Wiki [2016]

Mit einem Preis von rund 38 Euro zählt die zweite Generation des Raspberry Pi zu den günstigeren Einplatinencomputer. Für diesen gibt es eine Vielzahl speziell angepasster Distributionen, als Standard-Betriebssystem dient Raspbian, ein Debian-Derivat. Besonders hervorzuheben ist die, durch den großen Erfolg schnell gewachsene, Community hinter dem Raspberry Pi, welche für Anfänger oder bei Problemen Unterstützung bietet.

4.1.2. Banana Pi M2

Der Banana Pi ist das chinesische Pendant Raspberry Pi. Er erschien im März 2014 und bietet circa die doppelte Leistung des Raspberry Pi (erste Generation). Die GPIO-Pins sind dabei Pin-kompatibel zu bestimmten Versionen des Raspberry Pi. Kurz nach dem Erscheinen des Raspberry Pi 2 Model B, erschien auch auch eine neu aufgelegte Version des Banana Pi. Dieser Banana Pi M2 hat folgende technische Spezifikationen:

Größe	92,0 mm×60,0 mm×18,0 mm (L×B×H)
SOC	Allwinner A31s
CPU	ARM Cortex-A7, 4×1 GHz
GPU	PowerVR SGX544MP2
RAM	1 GB DDR3 SDRAM
LAN	Realtek RTL8211E/D, 10/100/1000 MBit
IO	HDMI 1.4, 4×USB2.0 + USB-OTG, 40 GPIO-Pins

Tabelle 4.2.: Spezifikationen Banana Pi M2[3]

Der Banana Pi M2 ist mit einem Preis von rund 54 Euro deutlich teurer als das Raspberry Pi 2 Model B, bietet aber nur wenig Mehrleistung. Durch seinen anderen SoC bietet er zwar Funktionen wie WLAN oder Bluetooth, benötigt aber auch andere Treiber und kann daher nicht auf die gleiche Auswahl an (spezialisierten) Betriebssystemen wie der Raspberry Pi zurückgreifen. Die Platine des Banana Pi M2 ist 16% größer als die des Raspberry Pi.

4.1.3. Odroid C1+

Der Odroid C1+ ist ein südkoreanischer Konkurrent des Raspberry Pi. Er erschien im August 2015 und ist somit im Vergleich der neuste Einplatinencomputer. Allerdings handelt es sich bei dem Modell nur um eine geringfügige Aktualisierung seines Vorgängers C1, welcher im Dezember 2014 erschien und den gleichen SoC nutzt. Besonderes Merkmal des C1+ ist dabei sein großer Aluminium-Kühlkörper. Der C1+ hat die folgenden Spezifikationen:

[3]Quelle: Banana Pi Team [2015]

Größe	85,0 mm×56,0 mm×17,0 mm (L×B×H)
SOC	Amlogic S805
CPU	ARM Cortex-A5, 4×1,5 GHz
GPU	Mali-450 Mp2
RAM	1 GB DDR3 SDRAM
LAN	Realtek RTL8211F, 10/100/1000 MBit
IO	HDMI 1.4, 4×USB2.0 + USB-OTG, 40 GPIO-Pins

Tabelle 4.3.: Spezifikationen Odroid C1+[4]

Mit seinem deutlich höher als beim Raspberry Pi oder Banana Pi getakteten Prozessor bietet der Odroid C1+ die meiste Rechenleistung. Auch die GPU-Leistung übertrifft die der Konkurrenten. Zusätzlich bietet der C1+ noch die Möglichkeit für den Anschluss von eMMC-Speichermodulen, welche wesentlich schneller sind als microSD-Karten. Er bietet 40 GPIO-Pins und die Abmessungen entsprechen denen des Raspberry Pi. Als Betriebssystem werden gängige Distributionen wie Ubuntu, Fedora oder Debian unterstützt. Der Preis liegt mit rund 48 Euro in der Mitte der Konkurrenten.

4.1.4. Fazit

Die gestellten Anforderungen werden von allen drei Modellen erfüllt bzw. übertroffen:

Modell	RPi 2 Model B	Banana Pi M2	Odroid C1+
Native Linux-Unterstützung	Ja	Ja	Ja
Mehrkernprozessor	Ja	Ja	Ja
512 MB RAM	Ja	Ja	Ja
USB-Anschluss	Ja	Ja	Ja
Digitaler Bildausgang	Ja	Ja	Ja
GPIO-Pins erreichbar	Ja	Ja	Ja
Preis in EUR[5]	38	54	48

Tabelle 4.4.: Einplatinencomputer im Vergleich

Die Auswahl an Einplatinencomputers ist groß und es erscheinen stetig neue, leistungsstärkere Modelle. Dies hat den Vorteil, dass, falls im weiteren Verlauf dieser Arbeit eine Mehr-

[4]Quelle: Hardkernel [2015]
[5]Quelle: http://geizhals.de/ (Stand 10.02.2016)

leistung eine signifikante Verbesserung der geforderten Funktion bietet, der Einplatinencomputer durch ein verfügbares, leistungsstärkeres Modell ausgetauscht werden kann.

Für diese Arbeit wird das günstigste Modell ausgewählt, das Raspberry Pi 2 Model B (im weiteren Verlauf kurz „Raspberry Pi“ genannt). Es erfüllt die Anforderungen, bietet die größte Auswahl an Betriebssystemen und es kann bei Problemen auf die mit Abstand größte Community zurückgegriffen werden.

4.2. Auswahl der Kamera

Für die Auswahl der Kamera werden drei unterschiedliche Kandidaten ausgesucht, welche im Folgenden ausführlich erläutert werden.

4.2.1. Logitech C525 HD Webcam

Bei der Logitech C525 HD Webcam handelt es sich um eine handelsübliche Webcam für Videoanrufe. Sie besitzt einen Autofokus und kann Videos in HD-Qualität (720p) aufnehmen. Angeschlossen wird sie über USB. Offiziellen Support für Linux-basierte Betriebssysteme gibt es nicht, allerdings soll die Kamera mit dem UVC (USB Video Class) Treiber erkannt werden und als V4L2 (Video4Linux 2) Gerät ansteuerbar sein.[6] Für Standbilder wird eine Auflösung von 8 Megapixeln angegeben.

4.2.2. IPEVO Ziggi-HD Plus

Bei der IPEVO Ziggi-HD Plus handelt es sich um eine sogenannte Dokumentenkamera. Auch sie besitzt einen Autofokus und kann Videos in HD-Qualität (1080p) aufnehmen. Sie verfügt über einen USB-Anschluss und hat keinen offiziellen Support für Linux-basierte Betriebssysteme. Auch hier hilft der UVC Treiber, welcher allerdings den Autofokus nicht steuern kann.[7] Die Kamera meldet sich über USB nicht nur als Kamera, sondern auch als HID (Human Interface Device) an, weshalb sich der Autofokus über die HID API steuern lässt. Die Auflösung für Standbilder beträgt 8 Megapixel (3264×2448).

[6] Quelle: Linux Hardware Guide [2012]
[7] Quelle: H. [2015]

4.2.3. Raspberry Pi Camera Module

Es gibt ein speziell für den Raspberry Pi entworfenes Kameramodul. Dieses nutzt das Camera Serial Interface (CSI) und ist somit im Gegensatz zu den Kameras mit USB-Anschluss direkt mit dem Prozessor verbunden. Bei dem Modul handelt es sich um einen OV5647 CCD-Sensor mit einer Auflösung von 5 Megapixeln (2592×1944) für Standbilder und der Möglichkeit, Videos in HD-Qualität (1080p) aufzunehmen. Die Verbindung erfolgt über ein 15-Pin Flachbandkabel. Der Fokus ist fest und mit 1 m - ∞ angegeben. Die Kamera wird vom Betriebssystem Raspbian über einen eigenen Treiber erkannt, kann aber auch als V4L2 Gerät benutzt werden.

4.2.4. Fazit

In der folgenden Tabelle 4.5 werden die recherchierten Kameras mit den in Abschnitt 3.1.2 aufgestellten Anforderungen verglichen.

Modell	C525 HD Webcam	Ziggi-HD Plus	RPi Camera Module
Auflösung >4,3 MP	Ja(8 MP[1])	Ja (8 MP)	Ja (5 MP)
Seitenverhältnis 4:3	Nein (16:9)	Ja	Ja
Fokussierbar auf 20 cm	Ja (7 cm)	Ja (10 cm)	Nein (1 m)
USB-Anschluss	Ja	Ja	Nein (CSI)
Linux-kompatibel	Ja[2]	Ja[2]	Ja
Preis in EUR[8]	43	111	27

Tabelle 4.5.: Kameras im Vergleich ([1]: Software-Interpoliert, [2]: Nicht offiziell)

Die Logitech C525 HD Webcam hat zwei Nachteile: Zum einen wird die Auflösung von 8 MP nur interpoliert[9], was für die Anwendung in dieser Arbeit nicht sinnvoll ist, da durch die Interpolation nur redundante Informationen entstehen. Diese verursachen mehr Rechenzeit bei der Bildbearbeitung, können die Texterkennung aber nicht verbessern. Der zweite Nachteil ist die fehlende offizielle Unterstützung. Daher ist nicht bekannt, welche Funktionen per Software steuerbar sind.

Die IPEVO Ziggi-HD Plus erfüllt die Anforderungen, bietet aber die vergleichsweise schlechteste Treiberunterstützung. Zusätzlich kostet sie im Vergleich viermal so viel wie die günstigste Kamera.

[8] Quelle: http://geizhals.de/ (Stand 12.02.2016)
[9] Quelle: Logitech [2016] - Technische Daten

Das Raspberry Pi Camera Module hat zwar keinen USB-Anschluss, der vorhandene CSI-Anschluss wird allerdings vom ausgewählten Einplatinencomputer unterstützt. Damit bleibt als einziger Nachteil das Fehlen der Einstellung der Fokussierung.

Nach weiterer Recherche stellt sich heraus, dass das Raspberry Pi Camera Module doch einen variablen Fokus besitzt.[10] Der Fokusring ist standardmäßig blockiert, die Blockierung lässt sich aber lösen (die genaue Vorgehensweise dazu ist in Unterabschnitt 5.1.2 beschrieben). Dann ist der Fokus bis auf wenige Zentimeter manuell einstellbar. Da so alle Anforderungen erfüllt sind, fällt die Wahl der Kamera auf das Raspberry Pi Camera Module.

4.3. Bildverarbeitung

Für eine automatische Erkennung von Schriftstücken und zur Vorbereitung für die Texterkennung ist eine Bildverarbeitung des Kamerabildes notwendig. Diese muss in der Lage sein, das Bild zu drehen und durch die Anwendung von Filtern das Bild für die Texterkennung zu optimieren.

Grundsätzlich gibt es eine Vielzahl von Bildbearbeitungsprogrammen, welche die Funktionen bereitstellen. Durch die Auswahl des Raspberry Pi und Raspbian als Betriebssystem, sowie die Notwendigkeit einer C++ API fallen viele Möglichkeiten weg. So bietet auch das wohl bekannteste Bildbearbeitungsprogramm für Linux-basierte Betriebssysteme, das GNU Image Manipulation Program (GIMP), keine passende Programmierschnittstelle. Zwei andere, gängige Möglichkeiten zur Bildbearbeitung werden hier kurz vorgestellt.

4.3.1. ImageMagick

ImageMagick ist ein Softwarepaket, welches weitreichende Funktionen zur Bildbearbeitung über die Kommandozeile bereitstellt. Es handelt sich um freie Software, die unter der Apache-Lizenz 2.0 veröffentlicht wird. Die aktuelle Version ist 6.9.3-1[11], welche über 200 Bildformate unterstützt. Für die Entwicklung bietet ImageMagick mit Magick++ eine API für C++. Mit Magick++ lassen sich benötigte Funktionen in die Software dieser Arbeit voraussichtlich gut integrieren.

[10] Quelle: Upton [2013]
[11] Stand 18.01.2016

4.3.2. OpenCV

OpenCV ist eine Bibliothek, die neben der Bildverarbeitung auch Algorithmen für das maschinelle Sehen (engl. „machine vision“ (MV), modern „computer vision“ (CV)) beinhaltet. Dazu zählen z. B. Gesten- und Objekterkennung. Auch OpenCV ist freie Software und wird mit der BSD-Lizenz veröffentlicht. Die aktuelle Version ist 3.1[12]. Es ist in C und C++ geschrieben und bietet mit der sog. OpenCV 2.x API eine C++ API.

4.3.3. Fazit

Zwar bieten sowohl ImageMagick als auch OpenCV jeweils eine gut dokumentierte API, allerdings bietet OpenCV zwei entscheidende Vorteile. Zum einen unterstützt es bereits Algorithmen zur Objekterkennung, welche in dieser Arbeit für die Erkennung und auch Unterscheidung von Schriftstücken genutzt werden könnten. Zum anderen gibt es für die in Unterabschnitt 4.2.4 ausgewählte Kamera eine zu OpenCV kompatible API, sodass die aufgenommenen Bilder ohne Speicherung auf der (langsamen) SD-Karte weiterverarbeitet werden können.

4.4. Texterkennung

Zur Texterkennung gibt es viele Programme, von denen drei näher betrachtet und nachfolgend kurz vorgestellt werden.

4.4.1. ABBYY FineReader

FineReader stammt von der russischen Software-Firma ABBYY. Die erste Version erschien 1993, die aktuelle Version 12 erschien 2014. Der ABBYY FineReader erhält immer wieder gute Bewertungen aus der Fachpresse und gilt deshalb in vielen Vergleichen als Referenz.[13] Die Software ist proprietär und kostenpflichtig, die Preise liegen zwischen 99 und 189 Euro.[14] Es gibt mit der „ABBYY FineReader Engine for Embedded OS“ sogar ein SDK mit Unterstützung von ARM-Prozessoren und Linux-Betriebssystemen.

[12] Stand 18.01.2016
[13] Quelle: Mendelson [2014]
[14] Quelle: ABBYY [2016a]

Der FineReader teilt zur Erkennung die Wörter in einzelne Buchstaben auf und erkennt diese. Unterstützt werden in der aktuellen Version 190 Sprachen, für 48 davon nutzt FineReader ein Wörterbuch zur Korrektur (ICR). ABBYY wirbt mit einer Erkennungsrate von 99,9%.[15]

4.4.2. Tesseract

Tesseract wurde zwischen 1985 und 1994 von Raymond Smith bei Hewlett-Packard (HP) entwickelt. 1998 wurde der Code nach C++ konvertiert und seitdem auch in dieser Sprache weiterentwickelt. Seit 2005 wird der Quellcode unter Apache-Lizenz 2.0 über SourceForge freigegeben. Im selben Jahr übernahm Google die Weiterentwicklung von Tesseract und nutzt es seit 2006 als Grundlage für Google Books. Die aktuelle Weiterentwicklung findet auf GitHub[16] statt. Die aktuelle Version von Tesseract ist 3.04 (Release 11.07.2015).
Bei Tesseract handelt es sich um eine reine Zeichenerkennung. Seit Version 3.01 können alle Unicode-Zeichen (UTF-8) erkannt werden. Es werden aktuell 39 Sprachen unterstützt und es ist möglich, weitere Sprachen anzulernen. Deutsch wird standardmäßig unterstützt [Smith, 2007].

4.4.3. OCRopus

OCRopus wurde 2004 von Google konzipiert, um für das damalige Google Print herkömmliche, gedruckte Bücher durchsuchbar zu machen. Seitdem wird es, mit Unterstützung seitens Google, am Deutschen Forschungszentrum für Künstliche Intelligenz (DKFI) unter der Leitung von Prof. Dr. Thomas Breuel entwickelt. Die erste Version wurde 2007 unter Apache-Lizenz 2.0 veröffentlicht. Die aktuelle Version ist 1.0 (02.11.2014) und die Veröffentlichung erfolgt auf GitHub[17]. Die verwendeten Programmiersprachen sind C++ und Python.
Ocropus ist modular aufgebaut und nutzte bis Version 0.4 (2009) Tesseract als Plugin zur Zeichenerkennung. Seitdem nutzt es dazu eine eigene Engine. Der Fokus liegt auf der Verbesserung der Erkennung durch die Verwendung von künstlichen neuronalen Netzwerken und deren Trainierbarkeit.

4.4.4. Fazit

Der ABBYY FineReader ist seit über 20 Jahren auf dem Markt ist. Seit der Freigabe von Tesseract 2005 nutzen viele freie Software-Projektes dieses als Backend, da die eigenen

[15] Quelle: ABBYY [2016b]
[16] Quelle: Smith u. a. [2016]
[17] Quelle: Breuel [2016]

Erkennungs-Engines von Tesseract übertroffen wurden.
Dadurch, dass der FineReader nicht quelloffen ist, kann er in dieser Arbeit nicht verwendet werden. Das verhältnismäßige neue OCRopus wird der Recherche nach kaum verwendet und ist leider nicht hinreichend dokumentiert. Zu Tesseract gibt es von dessen leitenden Entwickler Raymond Smith eine Reihe von Veröffentlichungen, bei denen auf verschiedene Anwendungszwecke eingegangen wird. Zusätzlich gibt es innerhalb der Community des Raspberry Pi bereits Erfahrungsberichte mit der C++ API von Tesseract. Daher fällt die Wahl der Texterkennungssoftware auf Tesseract.

4.5. Textausgabe

Die Ausgabe des Textes stellt zwar die Kernfunktion von einem Bildschirmlesegerät dar, rückt bei der Erstellung dieser Arbeit allerdings in den Hintergrund. Dies hat vor allem damit zu tun, dass hier die Leistung eines Einplatinencomputer in Bezug auf OCR untersucht wird. Deshalb wird hier, im Gegensatz zu den anderen Komponenten der Hard- und Software, auf eine Gegenüberstellung verschiedener Bibliotheken verzichtet und die bereits in OpenCV integrierte Highgui ausgewählt. Diese stellt Basisfunktionen zur Anzeige von Bildern und zur Darstellung von Texten bereit. Außerdem ist Highgui kompatibel zu größeren UI Frameworks wie z. B. WinForms oder Qt. Daher wird neben Highgui Qt als optionale Erweiterungsmöglichkeit der Basisfunktionen ausgewählt.

4.5.1. Qt

Bei Qt handelt es sich um eine in C++ geschriebene Bibliothek für die Erstellung von grafischen Benutzeroberflächen (GUIs). Die Entwicklung begann bereits 1991 durch zwei norwegische Programmierer. Daraus entwickelte sich später die Firma Trolltech, welche 2008 von Nokia aufgekauft wurde. Im Jahr 2011 wurde Qt unter dem Namen Qt-Project als freie Software veröffentlicht und gleichzeitig die kommerzielle Lizenzierung an Digia verkauft. Seit 2014 wird Qt von The Qt Company, einer Tochterfirma von Digia, weiterentwickelt. Bereits 2008 wurde die Open-Source-Version unter der LGPL veröffentlicht [Blanchette und Summerfield, 2009]. Die aktuelle Version ist 5.5.1 (15.10.2015), die ebenfalls unter LGPL (Version 3) veröffentlicht wird.
Qt-GUIs sind weit verbreitet. Zu den bekannteren Programmen, welche eine Qt-GUI nutzen, gehört z. B. der VLC Media Player (unter Windows und Linux).

4.6. Zusammenfassung

Nach einiger Recherche und anschließender Bewertung konnten für alle relevanten Funktionen geeignete Lösungen gefunden werden. Diese bestehen auf der Seite der Hardware aus einem Raspberry Pi 2 Model B mit passendem Raspberry Pi Camera Module. Auf der Seite der Software wurden die APIs OpenCV und Tesseract als Hauptbestandteile gewählt. Dazu kommt die in Unterabschnitt 4.2.3 erwähnte API des Kameramoduls. Aus den einzelnen Komponenten wird im folgenden Kapitel jeweils für die Hard- und die Software ein Gesamtentwurf entwickelt.

5. Entwurf

In diesem Kapitel werden die Entwürfe für die Hard- und Software vorgestellt.

5.1. Hardwareentwurf

Der Hardwareaufbau orientiert sich an dem Aufbau anderer Bildschirmlesegeräte. Als Basis wird hier mit einem Monitorständer mit Höhenverstellung geplant. Dies soll die Montage der Kamera unterhalb des Monitors an dessen Ständer ermöglichen. Der Einplatinencomputer soll ebenfalls am Ständer befestigt werden. Unterhalb der Kamera ist eine Ablagefläche für Schriftstücke vorgesehen. Die Interaktion mit dem Benutzer soll bei diesem Prototypen über eine herkömmliche USB-Tastatur erfolgen.
Diese Vorplanung wird unter der Verwendung des Monitorständers eines HP L1955 zu einem Gesamtentwurf weiterentwickelt. Die einzelnen Schritte und das Ergebnis dieser Entwicklung werden nachfolgend beschrieben.

5.1.1. Einplatinencomputer

Um den Raspberry Pi vor Verschmutzung und unbeabsichtigten Kurzschlüssen zu schützen, wird für diesen ein Kunststoffgehäuse der Marke Aukru (vgl. Abbildung 5.1) vorgesehen. Das Gehäuse bietet für alle Anschlüsse Aussparungen und steht auf gummierten Füßen. Es besteht aus zwei Teilen und lässt sich auch öffnen, wenn der Raspberry Pi darin verschraubt ist. Dieses Gehäuse soll am Monitorständer befestigt werden. Diese Befestigung für diesen Prototypen kann mit Kabelbindern oder doppelseitigem Klebeband erfolgen.

5.1.2. Kameramodul

Das Raspberry Pi Camera Module muss mittig über dem abgelegten Schriftstück platziert werden. Um dies zu erreichen, wird eine Art Arm konzipiert, an welchem die Kamera auf einer Art Schlitten bewegt werden kann. Dieser Arm soll dabei am, in Unterabschnitt 5.1.1 beschriebenen, Gehäuse befestigt werden, das am Monitorständer befestigt wird.

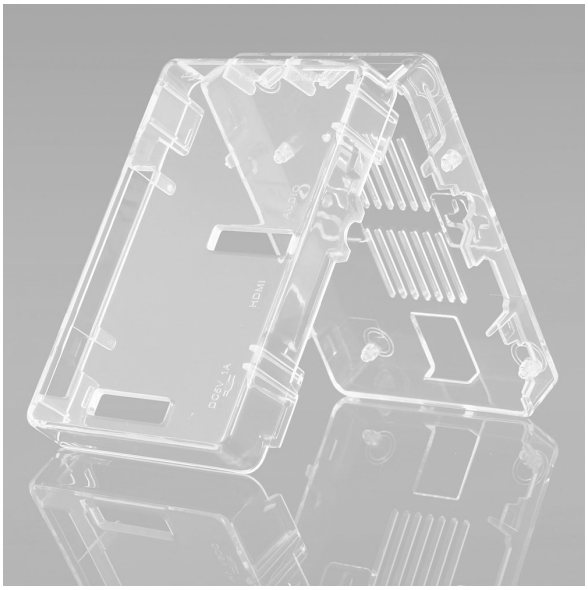


Abbildung 5.1.: Zweiteiliges Kunststoffgehäuse für Raspberry Pi 2 Model B von Aukru. Quelle: http://www.amazon.de/gp/product/B00UCSO9G6

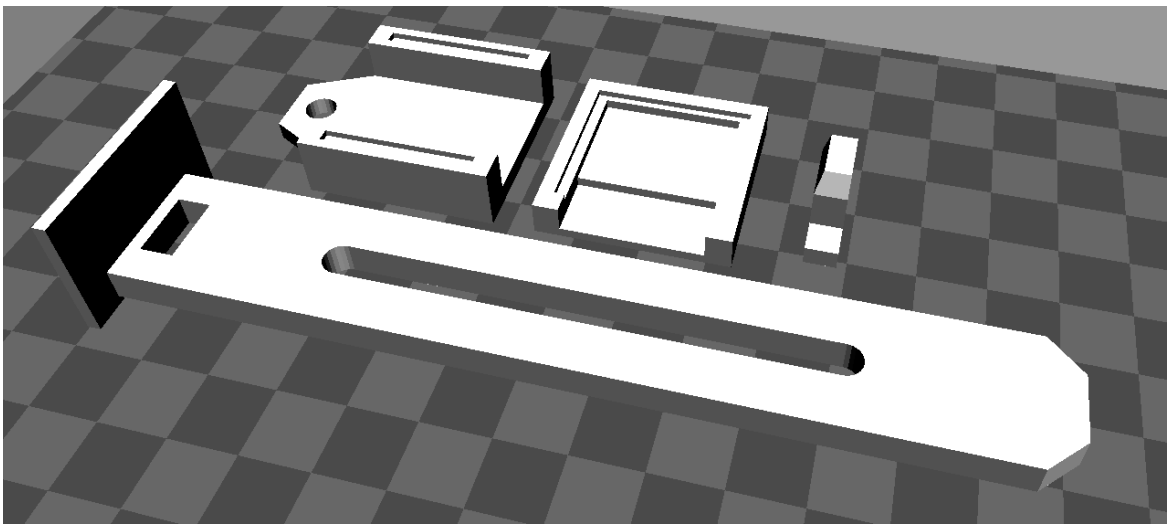


Abbildung 5.2.: Mit SketchUp erstellte Komponenten der Kamerahalterung, Screenshot von Cura (3D-Druck Software)

Der Arm wird anschließend auf Basis einer technischen Zeichnung in SketchUp konstruiert. Der 3D-Entwurf ist dabei so dimensioniert, dass der Arm durch die Öffnung des Gehäuses passt, welche für das Flachbandkabel zum Anschluss eines LCD gedacht ist. Da dieser Anschluss nicht genutzt wird, kann diese Öffnung genutzt werden (untere Öffnung auf Abbildung 5.1). Der Arm soll dazu von innen durch die Aussparung gesteckt werden und mit einer Art Keil in Position gehalten werden (Keil oben rechts auf Abbildung 5.2).
Der erwähnte Schlitten für die Kamera wird dabei in zwei Teilen konstruiert, welche hinterher verklebt werden müssen. Die Oberseite hat ein Loch, durch welches der Schlitten auf dem Arm arretiert werden kann (z. B. mit einer Schraube). Die Unterseite bietet für die Kamera passende Aussparungen, sodass diese dort eingesetzt werden kann. Der zusammengeklebte Schlitten soll auf den Arm passen und sich dort verschieben lassen, damit der für die Kamera sichtbare Bereich geeignet gewählt werden kann.

Damit die Kamera das passende Format (DIN A5, hochkant) aufnehmen kann, muss die Kamera um 90° gedreht verbaut werden. Dies kann durch den zweiteiligen Schlitten realisiert werden, indem das Unterteil entsprechend gedreht verklebt wird. Das aufgenommene Bild muss später in der Software wieder zurück gedreht werden.

Modifikation zur Fokussierung

Wie in Unterabschnitt 4.2.4 beschrieben, ist es möglich, den festen Fokus des Raspberry Pi Camera Modules zu lösen. Dies wurde von Liz Upton von der Raspberry Pi Fundation wie folgt beschrieben: Man bricht den Klebstoff, welcher die Linse in Position hält, heraus. Dazu wird diese mit einer Zange oder mit einem weichen Radiergummi gedreht.[1] Hier bei besteht allerdings das Risiko, die Optik der Kamera zu beschädigen. Mit dieser Methode ist es zwar möglich den Fokus zu ändern, allerdings ist dies aufgrund der Größe der Kamera nicht mit der Hand möglich.

Hier hilft der „Raspberry pi Camera - Focus ring“ (Thing 211641) von Thingiverse-Nutzer „anykey“ (vgl. Abbildung 5.3). Dabei handelt es sich um einen Entwurf eines Fokusrings, welcher mit einem 3D-Drucker herzustellen ist. Er wird mit Klebstoff auf die Linse geklebt und hat beim Drehen ausreichend Hebelwirkung, um die Blockade zu entfernen. Danach lässt sich über den Ring der Fokus per Hand einstellen.

5.1.3. Gesamtentwurf

Für den Gesamtentwurf wird ein Modell des Monitorständers des HP L1955, des Raspberry Pi Camera Module und einer Bodenplatte konstruiert. Das Modell des Raspberry Pi und

[1] Quelle: Upton [2013]

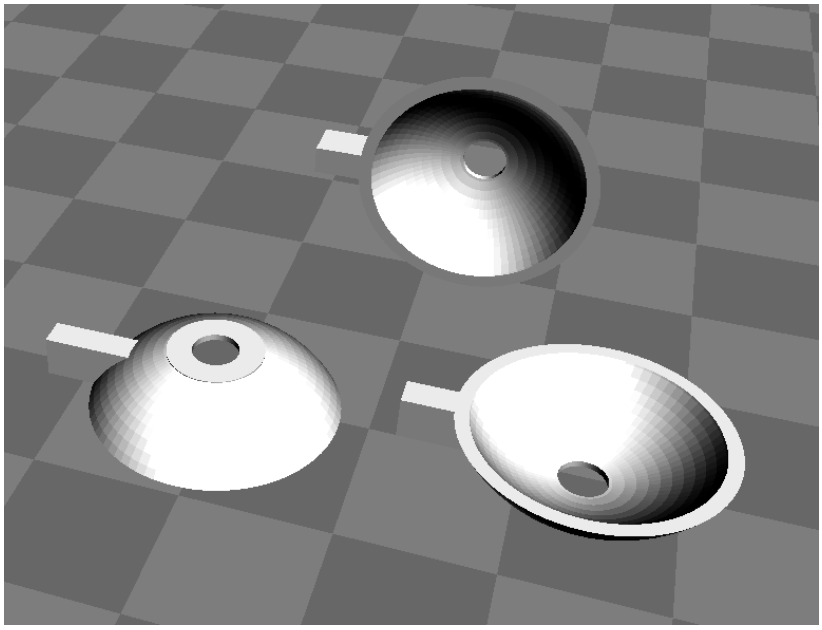
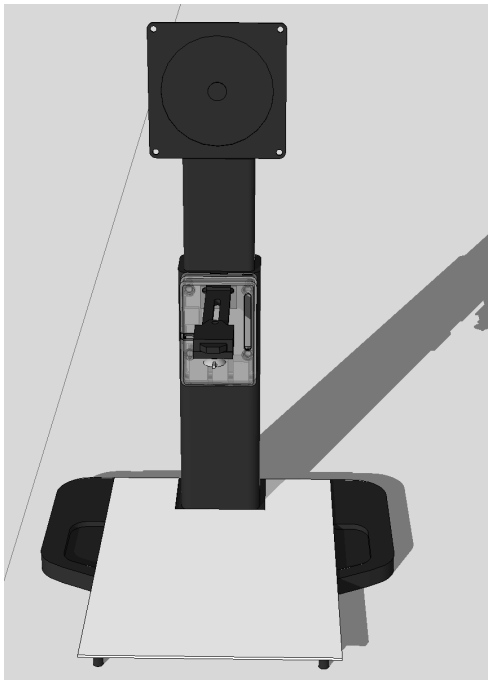


Abbildung 5.3.: Die druckbare 3D-Datei (.stl) des „Raspberry pi Camera - Focus ring“ aus mehreren Winkeln, Screenshot von Cura (3D-Druck Software)

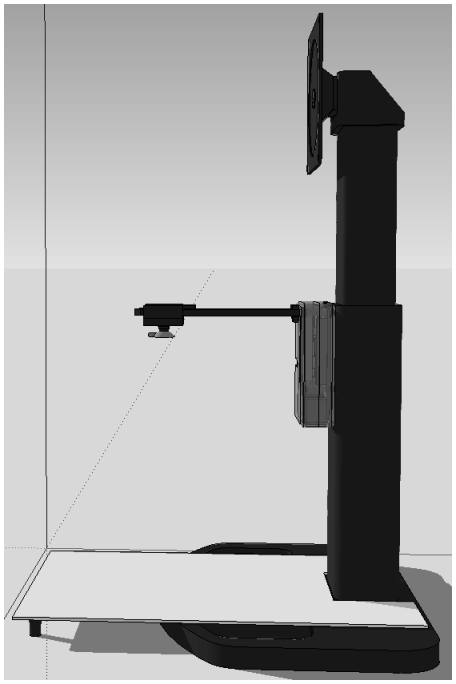
des Gehäuses wird von Thingiverse-Benutzer „fl3d“ (www.futurelab3d.com) übernommen. Es handelt sich dabei um dessen Entwurf „Raspberry Pi Type B+“ (Thing 402356).

Alle Modelle werden in SketchUp eingefärbt, um ein realistisches Modell des Gesamtentwurfes zu erhalten. Anschließend werden die einzelnen Objekte passend angeordnet. Der fertige Entwurf ist auf Abbildung 6.2 zu sehen. Der Raspberry Pi ist im Gehäuse senkrecht in der Mitte des Monitorständers montiert. Der Arm ragt aus der oberen Öffnung des Gehäuses. Die Kamera befindet sich dadurch mittig über der Bodenplatte. Diese liegt auf der einen Seite auf dem Monitorständer auf und wird auf der anderen Seite von zwei Füßen gestützt.

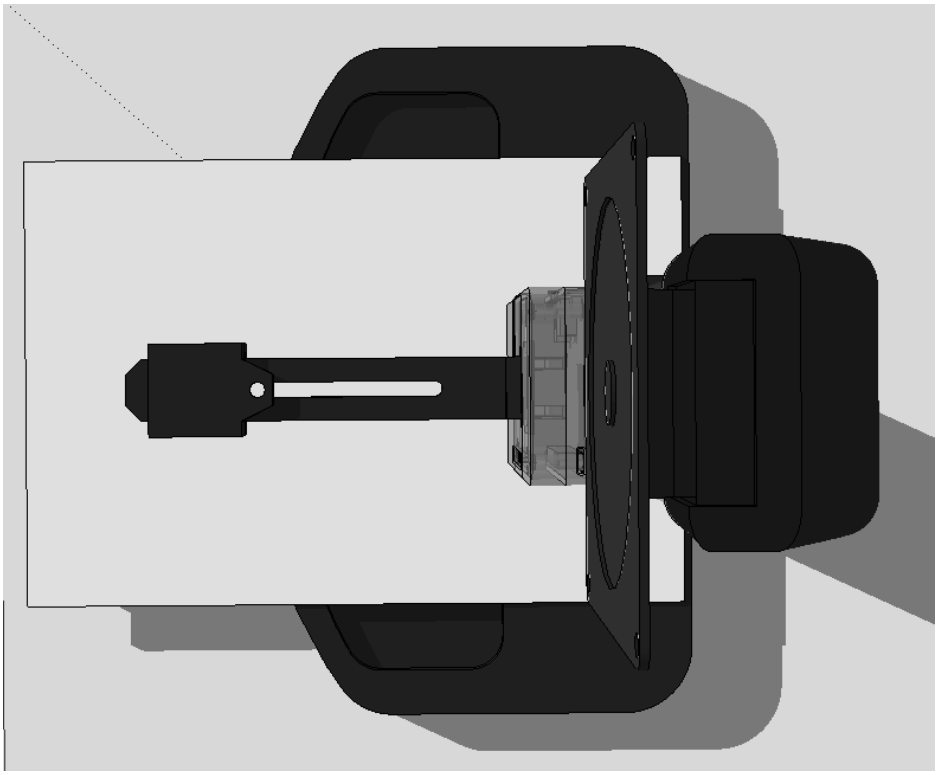
Als Material für die Bodenplatte ist, aufgrund der benötigten Stabilität, entweder ein dünnes Metallblech oder eine wenige Millimeter dicke Kunststoffplatte vorgesehen. Die Kamerahalterung, sowie der Fokusring sind für den Druck mit einem 3D-Drucker vorgesehen (Material PLA oder ABS).



(a) Frontansicht



(b) Seitenansicht



(c) Draufsicht

Abbildung 5.4.: Verschiedene Ansichten des Gesamtentwurfes, Screenshots von SketchUp

5.2. Softwareentwurf

Hier entsteht ein Entwurf der Software des Bildschirmlesegerätes. Zunächst werden die Abläufe der einzelnen Schritte beschrieben. Diese werden anschließend in einen zusammenhängenden Kontext gebracht, welcher den Gesamtablauf der Software repräsentiert. Die Umsetzung erfolgt in Abschnitt 6.3. Die Abbildungen der Programmablaufpläne befinden sich im Anhang B.

5.2.1. Abläufe der Einzelschritte

Bildaufnahme

Die RaspiCam API kennt zwei verschiedene Kamera-Modi. Der erste ist für die Aufnahme von Videobildern gedacht, der andere für die Aufnahme von Standbildern. Der Videomodus ist dabei auf eine Auflösung von 1280×960 Pixel begrenzt. Dieser soll für die Aufnahme des Vorschaubildes verwendet werden. Da für die Texterkennung allerdings die vollen 5 MP der Kamera benötigt werden, muss eine Umschaltung auf den Standbildmodus erfolgen. Dafür kann je ein Objekt der Kamera für die beiden Modi erstellt werden. Die Kamera muss vor einem Wechsel der Modi freigegeben und im jeweiligen neuen Modus erneut initialisiert werden.
Nach dem Programmstart soll als erstes ein Vorschaubild angezeigt werden (Preview-Mode). Dazu soll die Kamera im Videomodus initialisiert werden und in einer Schleife Bilder aufgenommen, gedreht und angezeigt werden. Die Schleife soll dabei durch einen Tastendruck abgebrochen werden können. Je nach gedrückter Taste soll entweder in den Testmodus gewechselt werden, ein Standbild aufgenommen werden, der Text erkannt und angezeigt werden oder das Programm beendet werden.
Der Entwurf der Bildaufnahme ist in Abbildung B.2 dargestellt.

Preprocessing

Als Preprocessing (deut. „Vorverarbeitung“) bezeichnet man die Vorbereitung von Daten für die eigentliche Verarbeitung. In diesem Fall müssen die Bilddaten aus dem aufgenommenen Foto für die Texterkennung vorbereitet werden. Es müssen durch ungleichmäßige Beleuchtung entstandene Kontrastunterschiede ausgeglichen werden. Hierzu kann das sog. „Adaptive Thresholding“ genutzt werden.

Adaptive Thresholding Thresholding (deut. „Schwellwertbildung“) bezeichnet die Umwandlung eines Graustufenbildes in ein binäres (schwarz/weiß) Bild. Entscheidungswert für schwarz oder weiß sind dabei die jeweiligen (Helligkeits-)Informationen jedes einzelnen Pixels. Die festgelegte Entscheidungsschwelle (engl. „threshold“) legt fest, ab welchem Wert ein Pixel schwarz wird.
Thresholding wird benutzt, um relevante Informationen aus einem Bild zu extrahieren. Bei mittig eingestelltem Thresholding (Wert 127 bei 8 Bit) werden bei einem gescannten Text die Buchstaben schwarz und der Hintergrund weiß dargestellt. Sollte der Scan über- oder unterbelichtet sein, so muss man die Schwelle entsprechend anpassen. Kleine Fehler wie z. B. Knicke im Papier oder Verunreinigungen können so herausgefiltert werden.
Wird nun, wie in dieser Arbeit, ein Foto aufgenommen, ist die Belichtung, im Gegensatz zu einem Scan, niemals gleichmäßig. Wendet man hier Thresholding mit festem Wert an, wird, je nach Schwelle, entweder ein Großteil des Bildes schwarz (schattiger Teil des Fotos) oder weiß (ausgeleuchteter Teil des Fotos) sein (vgl. Abbildung 5.5 oben rechts).
Um dieses Problem zu lösen, vergleicht man den Wert jedes Pixels mit dem Mittelwert (engl. „Mean“) seiner benachbarten Pixel. Weicht ein Wert stark von seiner Umgebung ab, wird dieser entsprechend festgelegt. Dieses Verfahren nennt man „Adaptive Thresholding“ (deut. „sich anpassende Schwellwertbildung“). Die Größe der Nachbarschaft und die Stärke der Abweichung lassen sich dabei einstellen. Das Ergebnis ist eine von der Beleuchtung unabhängige Schwellwertbildung (vgl. Abbildung 5.5 unten).

OpenCV bietet mit `cv::adaptiveThreshold` eine entsprechende Funktion, welche eine Vorbereitung des Fotos für die Texterkennung ermöglicht [Brahmbhatt, 2013, S. 124ff.]. Dafür muss das aufgenommene Foto zunächst in ein Graustufenbild umgewandelt werden und anschließend die genannte Funktion angewendet werden.

Der sich daraus ergebende Ablaufplan des Preprocessings ist in Abbildung B.3 dargestellt.

Texterkennung

Die Tesseract API ermöglicht das modulare Einbinden von Tesseract. Dabei kann innerhalb einer Funktion ein Zeiger auf ein Objekt vom Typ `tesseract::TessBaseAPI` erzeugt werden, mit welchem aus dem aufgenommenen und bearbeiteten Bild der Text extrahiert werden kann. Die Funktion kann den Text dann als `std::string` zurückgeben. Dies ließe sich auch parallelisieren, indem die Funktion von einem Thread aufgerufen wird. Dazu ist es nötig, innerhalb des Preprocessings bereits Textfelder zu erkennen oder den Text in Teile (z. B. Zeilen) aufzuteilen.
Der sich daraus ergebende Ablaufplan ist in Abbildung B.4 dargestellt.

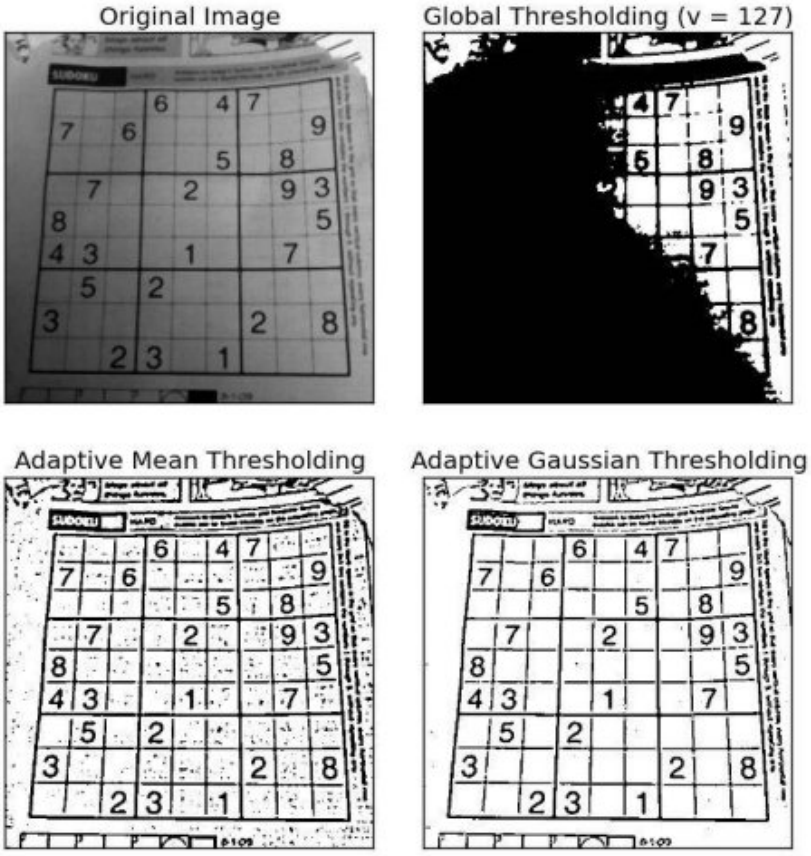


Abbildung 5.5.: Ergebnisse für verschiedene Thresholding-Methoden am Beispiel eines Fotos eines Sudokus: Oben rechts einfaches Thresholding mit festem Wert, unten links adaptive Thresholding mit Mittelwertbildung, unten rechts adaptive Thresholding mit gewichteter Summe nach Gauß.
(Quelle: http://docs.opencv.org/3.1.0/ada_threshold.jpg)

Ausgabe

Zur Ausgabe des erkannten Textes soll Highgui von OpenCV genutzt werden (vgl. Abschnitt 4.5). Dessen Funktionen zur Textausgabe sind einfach gehalten. Zur Anzeige eines Textes muss dieser zuvor in ein Bild geschrieben werden, welches dann ausgegeben werden kann. Die Funktion `cv::putText` kann dabei eine Zeile Text schreiben. Zur Ausgabe eines erkannten Textes muss also die Formatierung (Zeilenumbrüche) und Größe des Textes vom Programm ermittelt werden. Dazu soll der Text zunächst passend für das Ausgabefenster skaliert werden.
Während der Textausgabe soll das Programm erneut in einer Schleife auf Benutzereingaben warten. Je nach Tastendruck soll dabei entweder in den Preview-Mode zurückgekehrt werden, der angezeigte Text geändert (z. B. Schriftart) werden oder das Programm verlassen werden.
Der sich daraus ergebende Ablaufplan ist in Abbildung B.5 dargestellt.

Testmodus

Im Testmodus sollen in einer Schleife mehrere Bilder eines Vergleichstextes gemacht werden, deren Text erkannt und diese Texte mit einem gespeicherten Text verglichen werden. Als Maß der Ähnlichkeit der beiden Texte soll die sog. Levenshtein-Distanz genutzt werden. Diese soll nach jedem Durchlauf ausgegeben werden.

Levenshtein-Distanz Die Levenshtein-Distanz ist ein von dem russischen Mathematiker Wladimir Lewenstein entwickeltes Verfahren zur Bestimmung der Ähnlichkeit zweier Texte. Das Verfahren zählt die nötigen Schritte, um den einen Text in den anderen zu ändern. Die erlaubten Operationen sind dabei: Löschen, Einfügen und Ersetzen von Zeichen.

Der sich daraus ergebende Ablaufplan ist in Abbildung B.1 dargestellt.

5.2.2. Gesamtablauf

Der Gesamtablauf besteht aus der Kombination der beschriebenen Einzelschritte. Der in Abbildung B.6 dargestellte Ablaufplan zeigt, dass alle Schritte in einer Schleife ausgeführt werden, damit mehrere Schriftstücke nacheinander erkannt werden können.

6. Realisierung

In diesem Kapitel geht es darum, die Entwürfe des vorherigen Kapitels möglichst genau umzusetzen. Hard- und Software werden dabei getrennt betrachtet.

6.1. Hardwareaufbau

Dieser erste Abschnitt befasst sich mit der Umsetzung des Hardwareentwurfes. Zunächst wird der Aufbau des Raspberry Pi beschrieben. Da das Kameramodul bereits in Unterabschnitt 4.2.3 ausreichend erläutert wird, wird sich auf die Konstruktion konzentriert. Am Ende wird der Gesamtaufbau beschrieben und abgebildet.

6.1.1. Raspberry Pi 2 Model B

Der Raspberry Pi wird mit Kühlkörpern bestückt und in das Kunststoffgehäuse (vgl. Unterabschnitt 5.1.1) eingebaut. Die Abbildung 6.1 zeigt ihn im geöffneten Gehäuse. Mittig an der unteren Seite befindet sich der HDMI-Anschluss für den Monitor. Links davon ist der Spannungseingang in Form einer Micro-USB-Buchse. Als Spannungsversorgung dient hier ein Netzteil mit einer Leistung von 10 W. Rechts von dem HDMI-Anschluss befindet sich der Anschluss für das Flachbandkabel zur Kamera (CSI), daneben eine 3,5 mm-Klinkenbuchse zur Audioausgabe. An der rechten Seite befinden sich oben die vier (2×2) USB-2.0-Anschlüsse und darunter der RJ-45 Port für den Anschluss an ein Netzwerk. An der oberen Seite befinden sich die 40 GPIO-Pins. An der linken Seite befindet sich ein weiterer Anschluss für ein Flachbandkabel (DSI). Dieser ermöglicht die Verbindung zu einem speziell für den Raspberry Pi entwickelten Multitouch-fähigem LCD. Unter den Kühlkörpern befinden sich der SoC (großer Kühler) und der Netzwerk-Controller (kleiner Kühler). Der Einschub für eine microSD-Karte befindet sich auf der Unterseite.

Als Hauptspeicher kommt eine microSD-Karte (16 GB, Class 10) zum Einsatz. Auf dieser wird Raspbian Jessie installiert. Um genug Platz für die benötigten Bibliotheken zur haben, muss der Speicher manuell auf die vollen 16 GB erweitert werden (Standardgröße der von

Raspbian angelegten Partition sind 2,6 GB). Zur Bedienung und Entwicklung wird eine Tastatur angeschlossen und der Raspberry Pi über ein Netzwerk mit einem Entwicklungsrechner verbunden.

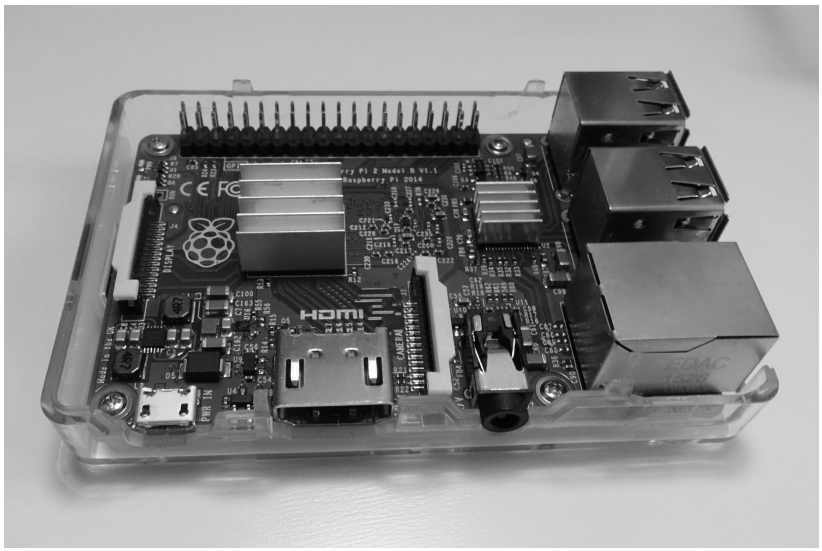


Abbildung 6.1.: Raspberry Pi 2 Model B im geöffneten Gehäuse

6.1.2. Raspberry Pi Camera Module

Das Kameramodul wird entsprechend der Beschreibung aus Abschnitt 5.1.2 mit dem Fokusring[1] bestückt und lässt sich somit manuell fokussieren (vgl. Abbildung 6.2a). Die Kamera wird mit Hilfe der Kamerahalterung[2] mittig über dem für das Schriftstück vorgesehenen Bereich in einer Höhe von 20 cm (ab Linse) über der Ablage befestigt. Der Fokus wird auf etwa 19 cm eingestellt, um bei dickeren Büchern noch ein scharfes Bild zu haben. Die Anbindung an den Raspberry Pi passiert über ein 15 cm langes Flachbandkabel.

6.1.3. Gesamtaufbau

Der Gesamtaufbau stimmt weitgehend mit dem Entwurf aus Unterabschnitt 5.1.3 überein. Das Gehäuse mit dem Raspberry Pi wird mit Kabelbindern am Monitorständer befestigt. Dank der Gummifüße des Gehäuses wird dieses gut in Position gehalten. Die 3D-Drucke werden für eine optimale Passgenauigkeit mit Schlüsselfeilen nachbearbeitet. Der Arm passt

[1] Der Fokusring wurde bei hamburg-druckt-3.de in weißem PLA gedruckt (Drucker: Makerbot Replicator 2)
[2] Die Kamerahalterung wurde bei hamburg-druckt-3.de in schwarzem ABS gedruckt (Drucker: Zortax-M200)

(a) Kameramodul mit Fokusring (b) Kameramodul in passgenauer Halterung

Abbildung 6.2.: Raspberry Pi Camera Module mit Fokusring

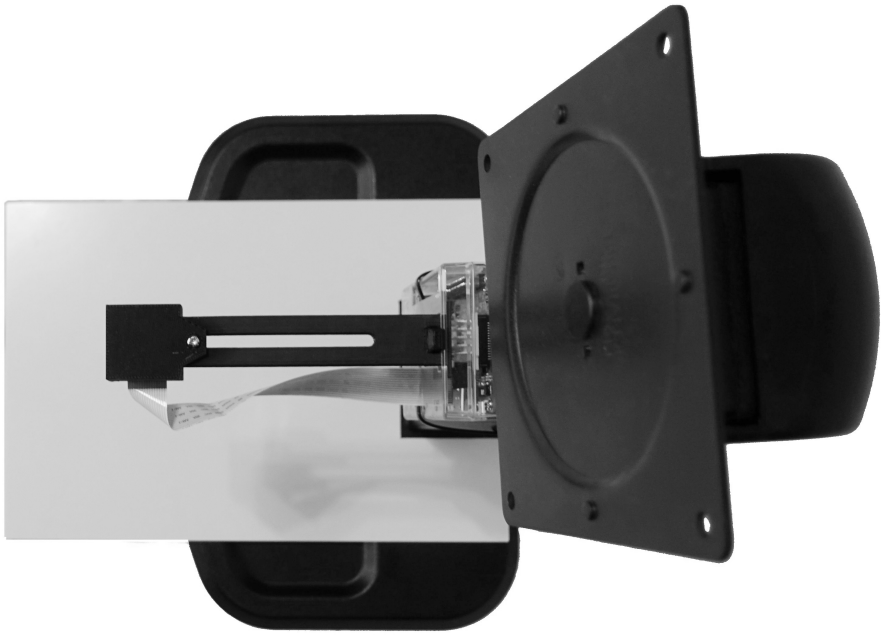
präzise durch die Aussparung des Gehäuses und hält auch ohne Verkleben sicher (mit eingesetztem Keil). Die zwei Teile des Schlittens werden mit Sekundenkleber verklebt und der fertige Schlitten wird mit einer Schraube am Arm der Kamerahalterung fixiert. Für die Bodenplatte kommt eine DIN A4 große, 2 mm starke Kunststoffplatte aus Polystyrol zum Einsatz. Diese wird mit einem Ausschnitt entsprechend des Entwurfes versehen und mit dem Monitorständer verklebt.

USB Anschlüsse und der Netzwerkanschluss ragen nach und unten lassen sich problemlos erreichen, was auch für die seitlichen Anschlüsse für die Spannungsversorgung und des Bildschirms gilt. Der Aufbau ist stabil genug, um sich beim Ein- oder Ausstecken eines Steckers nicht zu verschieben.

Der Aufbau wird aus ähnlichen Winkeln fotografiert, aus denen auch die Screenshots des Gesamtentwurfes (vgl. Abbildung 6.2) gemacht werden. Bei der Darstellung in Abbildung 6.3 ist auf Abbildung 6.3b eine Krümmung der Kamerahalterung zu erkennen. Dies liegt daran, dass das entsprechende Teil (der Arm) in der ersten Version mit zu wenig Füllung (engl. „Infill“) gedruckt wurde. Dieses Problem wird durch Tausch mit einem stärken Arm gelöst. Dieser hält die Kamera gerade über der Ablage.

(a) Frontansicht

(b) Seitenansicht



(c) Draufsicht

Abbildung 6.3.: Verschiedene Ansichten des Gesamtaufbaus

6.2. Softwareeinsatz

Wie in Unterabschnitt 6.1.1 beschrieben, wird der Raspberry Pi für die Entwicklung über ein Netzwerk mit einem Computer verbunden. Auf diesem wird NetBeans IDE 8.1 zur Entwicklung der Software genutzt. Um Probleme durch Cross-Compiling zu vermeiden, lässt sich mit NetBeans via SSH auf den Raspberry Pi zugreifen. Startet man den Kompiliervorgang, so werden die aktuellen Projektdaten auf den Raspberry Pi kopiert und dort mit allen benötigten Compiler-Flags kompiliert.
Auf eine direkte Entwicklung auf dem Raspberry Pi wird verzichtet, da eine IDE, wie z. B. Eclipse, dort viel Speicherplatz benötigt und wegen der begrenzten Rechenleistung nur langsam ausgeführt werden kann.

Verwendete Bibliotheken

Die folgende Tabelle 6.1 zeigt eine Übersicht der für die Software installierten und verwendeten Bibliotheken:

Name der Bibliothek	Version	Verwendung
OpenCV	3.0.0	Bildverarbeitung und Ausgabe
RaspiCam	0.1.3	Ansteuerung des Kameramoduls
Tesseract	3.03	Texterkennung
Leptonica	1.71	Layouterkennung für Tesseract
Qt	5.3.2	Erweiterung Textausgabe

Tabelle 6.1.: Zur Erstellung der Software verwendete Bibliotheken

In den folgenden Abschnitten werden Besonderheiten bei der Verwendung der Bibliotheken auf dem Raspberry Pi geschildert. Dabei geht es um die Bezugsquellen und eine kurze Bewertung.

6.2.1. OpenCV

Die genutzte Version 3.0.0 kann nicht aus dem Repository von Raspbian bezogen werden. Sie kann von GitHub oder Sourceforge heruntergeladen werden und muss anschließend gebaut werden. Dabei wird sich an der Anleitung von PyImageSearch[3] orientiert. Die gesamte Installation benötigt dabei mehrere Stunden und funktioniert aufgrund der Größe nur,

[3] Quelle: Rosebrock [2015]

wenn zuvor in der Raspi-config per „expand_rootfs“ der gesamte Speicherplatz der SD-Karte nutzbar gemacht wird.

6.2.2. Raspicam

Die RaspiCam API muss ebenfalls über Sourceforge bezogen werden. Die verwendete Version 0.1.3 wird zur Zeit nicht weiterentwickelt. Die API ermöglicht zwar den Zugriff auf alle Funktionen der Kamera, enthält aber auch Fehler. Ein Fehler ist, dass der Kameramodus für Standbilder die Kamera nicht wieder freigeben kann. Wird die Kamera nach dem Standbildmodus freigegeben und im Videomodus initialisiert, führt dies zu einer Fehlermeldung. Diese Funktion ist für den Gesamtablauf allerdings zwingend erforderlich.
Der Fehler ist bereits am 13.05.2014 bei Sourceforge vom Benutzer „kbarni“ als Ticket gemeldet worden, welcher auch einen Workaround veröffentlicht hat.[4] Dazu müssen die Quelldateien editiert werden und die RaspiCam API mit den Änderungen neu gebaut werden. Anschließend wird die Kamera korrekt freigegeben, das Bild im Standbildmodus aber ab dem zweiten Aufruf gespiegelt. Dies wird von der Software durch erneutes Spiegeln des Bildes behoben.

6.2.3. Tesseract

Tesseract wird in der Version 3.02 genutzt, welche direkt aus dem Repository bezogen werden kann. Tesseract nutzt die Leptonica Bibliothek zur Layouterkennung und funktioniert nur, wenn diese auch installiert ist. Die Verwendung mit OpenCV funktioniert gut.

6.2.4. Qt

Auf die Nutzung von Qt wird in dieser Arbeit verzichtet. Zwar lassen sich die nötigen Pakete aus dem Repository von Raspbian beziehen, allerdings wurden beim Kompilieren von OpenCV nicht die korrekten Einstellungen für die Nutzung von Qt gewählt. Dies führt zu einem Absturz der Software beim Aufruf entsprechender Funktionen. Die Implementierung der Ausgabe wurde daraufhin zeitlich an das Ende dieser Arbeit verschoben und auf die Erweiterung mit Qt aus zeitlichen Gründen verzichtet.

[4] Quelle: Sourceforge [2014]

6.3. Implementierung

In diesem Abschnitt wird anhand von Code-Beispielen gezeigt, wie wesentliche Funktionen implementiert werden. Die gezeigten Beispiele sind Ausschnitte des Gesamtcodes, welcher sich im Anhang A (CD) befindet.

6.3.1. Bildaufnahme

Die Bildaufnahme wird wie im Entwurf beschrieben umgesetzt. In Quellcode 6.1 ist zu sehen, wie erst die beiden Kamera-Objekte erstellt werden. Diesen werden die unterschiedlichen Auflösungen zugeteilt: `CameraVM` (VideoModus) wird auf 640×480 (0,3 MP), `Camera` auf 2592×1944 (5,0 MP) eingestellt.
Anschließend wird in Zeile 12 `CameraVM` initialisiert. In der Schleife in den Zeilen 15-25 wird die Kamera ausgelöst (Z. 17), die Kamera ausgelesen (Z. 18), das Bild gedreht (Z. 19) und angezeigt (Z.20). Das Bild wird dabei in der `cv::Mat liveImage` gespeichert. Bei jedem Durchlauf werden Tastendrücke aufgezeichnet (Z. 16) und ausgewertet (Z. 21-24). Die Bildwiederholrate in diesem Vorschaumodus beträgt circa 15 Bilder pro Sekunde und ermöglicht eine genaue Positionierung des Schriftstückes.
In Zeile 26 wird die Kamera von `CameraVM` wieder freigegeben und anschließend `Camera` initialisiert (Z. 28). Die Bildaufnahme funktioniert hier analog zum Vorschaumodus, das Bild wird in der `cv::Mat cameraImage` gespeichert. In den Zeilen 32-34 findet das Spiegeln statt, welches in Unterabschnitt 6.2.2 erwähnt wird. Die Variable `flip` wird mit `0` initialisiert und sorgt dafür, dass das Bild erst ab dem zweiten Druchlauf gespiegelt wird.

```
1  raspicam::RaspiCam_Cv CameraVM; //hard coded video mode, max 1280x960
2  raspicam::RaspiCam_Still_Cv Camera; // still mode, 5MP
3  //matrixes for images
4  cv::Mat cameraImage;
5  cv::Mat liveImage;
6  //set camera parameter
7  CameraVM.set(CV_CAP_PROP_FRAME_WIDTH, 640);
8  CameraVM.set(CV_CAP_PROP_FRAME_HEIGHT, 480);
9  Camera.set(CV_CAP_PROP_FRAME_WIDTH, 2592);
10 Camera.set(CV_CAP_PROP_FRAME_HEIGHT, 1944);
11 //open video camera
12 if (!CameraVM.open()) {std::cerr<<"camera error"<< std::endl;return -1;}
13 char pressedKey = ' ';
14 //Start capture
15 while(1) {
16     pressedKey = cv::waitKey(20)%256; //wait 20ms
17     CameraVM.grab();          //trigger camera
```

```
18      CameraVM.retrieve(liveImage); //get image
19      rot90(liveImage, 1);        //rotate
20      cv::imshow(liveWindowName, liveImage);  //display image
21      if(pressedKey == 'q') //quit
22          return 0;
23      if(pressedKey == 'a') //continue
24          break;
25  }
26  CameraVM.release();
27  //open photo camera
28  if (!Camera.open()) {std::cerr << "camera error" << std::endl;return -1;}
29  Camera.grab();
30  Camera.retrieve(cameraImage);
31  //Workaround for bug in raspicam api
32  if(flip)
33      cv::flip(cameraImage, cameraImage,0);
34  flip = 1;
35  rot90(cameraImage, 1);
```

Quellcode 6.1: Codebeispiel Vorschaubild und Standbildaufnahme

6.3.2. Preprocessing

Das Preprocessing besteht, wie im Entwurf, aus zwei Funktionen (Quellcode 6.2). Diese werden beide von OpenCV zur Verfügung gestellt. Im ersten Schritt wird aus dem `cameraImage` mit der Funktion `cv::cvtColor` („convert color“) ein Graustufenbild errechnet und dieses in der `cv::Mat editorImage` gespeichert. Im zweiten Schritt wird mit der Funktion `cv::adaptiveThreshold` das im Entwurf beschriebene Thresholding durchgeführt. Bei dem Parameter `255` handelt es sich um den maximalen Ausgabewert (255 entspricht Weiß). Genutzt wird hier das „adaptive gaussian thresholding“ durch Verwendung des Flag `CV_ADAPTIVE_THRESH_GAUSSIAN_C`. Mit dem Flag `CV_THRESH_BINARY` wird der Typ ausgewählt, möglich wäre hier auch `CV_THRESH_BINARY_INV` für ein Ausgabebild mit invertierten Schwarz/Weiß-Werten. Bei dem vorletzten Parameter handelt es sich um die Größe der Nachbarschaft, deren Helligkeitswerte durch Kreuzkorrelation mit der Gaußschen Glockenkurve gewichtet werden. Der letzte Parameter ist eine Konstante, welche von der Summe abgezogen wird. Deren Werte `55` und `25` wurden empirisch ermittelt.

```
1  cv::cvtColor(cameraImage,editorImage,CV_RGB2GRAY);
2  cv::adaptiveThreshold(editorImage, editorImage, 255,
       CV_ADAPTIVE_THRESH_GAUSSIAN_C, CV_THRESH_BINARY, 55, 25);
```

Quellcode 6.2: Codebeispiel Preprocessing

6.3.3. Texterkennung

Die Funktion zur Texterkennung konnte gemäß des Entwurfes umgesetzt werden (Quellcode 6.3). Der Funktion `getText` wird eine `cv::Mat` übergeben. Es wird ein Zeiger auf ein Objekt vom Typ `tesseract::TessBaseAPI` erzeugt. Mit diesem wird Tesseract (`tess`) initialisiert (Z. 5). Anschließend wird die `cv::Mat` als von `tess` zu erkennendes Bild festlegen (Z. 11). Dann wird die Texterkennung in Zeile 12 gestartet und in Zeile 13 der erkannte Text ausgelesen. Danach wird Tesseract beendet und der Speicher wieder freigegeben (Z. 25). Abschließend wird der erkannte Text als `std::string` zurückgegeben.
Der Vorgang dauert für eine Taschenbuchseite rund 15 Sekunden.

```
1  std::string getText(cv::Mat img)
2  {
3      tesseract::TessBaseAPI *tess = new tesseract::TessBaseAPI();
4      // Initialize tesseract-ocr with German
5      if (tess->Init(NULL, "deu")) {
6          fprintf(stderr, "Could not initialize tesseract.\n");
7          exit(1);
8      }
9      char *outText;
10     //convert img to text
11     tess->SetImage((uchar*)img.data, img.size().width, img.size().height,
           img.channels(), img.step1());
12     tess->Recognize(0);
13     outText = tess->GetUTF8Text();
14   tess->End();
15     return std::string(outText);
16 }
```

Quellcode 6.3: Codebeispiel Funktion zur Texterkennung

6.3.4. Ausgabe

Von der umgesetzten Textausgabe wird hier das Schreiben in ein leere Bild gezeigt (Quellcode 6.4). Zuvor werden Textgröße `textsize` (in Pixeln) und Schriftstärke `thickness` berechnet und die Schriftart `textFont` und Schriftfarbe `textColor` festgelegt. Der Faktor `fontScale` dient zur Umrechnung der dimensionslosen Schriftgrößen von OpenCV in Pixel. Im `std::vector rows` sind die einzelnen Zeilen des erkannten Textes gespeichert.
Zunächst wird in der linken oberen Ecke ein zur Schriftgröße passender Ursprung errechnet (Z. 1). In der Schleife in den Zeilen 3-6 werden die einzelnen Zeilen mit den o. g. Parametern in das Ergebnisbild `resultImage` geschrieben (Z. 4). Die Funktion `cv::putText()`

unterstützt dabei nur die Darstellung von ASCII-Zeichen. Anschließend wird der Ursprung nach unten verschoben (Z. 5). Nach dem Schreiben wird das Ergebnisbild angezeigt (Z. 7). Danach beginnt eine Warteschleife auf Tasteneingaben, ähnlich wie in Quellcode 6.1.

```
1 cv::Point org(10, textsize.height * 1.5); //factor x form fontScale to
      pixel
2 std::cout << "org calculated" << std::endl;
3 for(auto & row : rows) {
4     cv::putText( resultImage,row, org, textFont, fontScale, textColor,
          thickness, lineType);
5     org.y += textsize.height * 1.5;
6 }
7 cv::imshow(liveWindowName, resultImage);
8 cv::waitKey(30);
```

Quellcode 6.4: Codebeispiel Textausgabe

6.3.5. Testmodus

Der Testmodus enthält Elemente aus der Bildaufnahme, dem Preprocessing und der Texterkennung (Quellcode 6.5). Da im Testmodus alle fünf Bilder aufgenommen werden ohne den Kameramodus zu wechseln, ist hier kein Spiegeln erforderlich (Z. 11). In der Funktion `checkText()` (Z. 4) wird neben der Levenshtein-Distanz noch eine weitere Fehleranzahl berechnet und ausgegeben. Diese wurde aufgrund schlechter Vergleichbarkeit in Kapitel 7 nicht genutzt.

```
1  for(int i = 0; i < 5; i++) {
2      std::cout << "Erkennung Nr. " << i+1 << std::endl;
3      std::cout << ocrText << std::endl;
4      checkText(ocrText);
5      ocrText.clear();
6      if(i == 4){ break;}
7      Camera.grab();
8      Camera.retrieve(cameraImage);
9      rot90(cameraImage, 1);  //workaround not needed
10     cv::cvtColor(cameraImage,editorImage,CV_RGB2GRAY);
11     cv::adaptiveThreshold(editorImage, editorImage, 255,
           CV_ADAPTIVE_THRESH_GAUSSIAN_C, CV_THRESH_BINARY, 55, 25);
12     ocrText = getText(editorImage);
13 }
14 return 0;
```

Quellcode 6.5: Codebeispiel Testmodus

7. Testläufe

Um einen Überblick über die Möglichkeiten der Texterkennung mit dem Prototypen zu bekommen, wird eine Testreihe aufgenommen und ausgewertet.

7.1. Testverfahren

Für den Test wird derselbe Text in verschiedenen Schriftarten und -größen ausgedruckt und mit dem Prototypen erkannt. Als Text wird der englische und deutsche Teil des Beispieltextes aus der Tesseract-Dokumentation[1] genutzt (vgl. Abbildung 7.1). Der Text enthält 200 Zeichen, worin auch viele Sonderzeichen enthalten sind. Getestet werden die Schriftgrößen 8 pt, 10 pt, 12 pt, 16 pt und 20 pt (DTP-Punkte, 1 pt = 0,352778 mm), da diese für normale Taschenbücher und Taschenbücher mit Großschrift gebräuchlich sind. Als Schriftarten werden die in Unterabschnitt 2.3.1 genannten Schriftarten OCR-A und OCR-B, die Standardschriftarten Arial und Times New Roman, sowie die einer Handschrift nachempfundene Schriftart Lucida Handwriting genutzt.

Im automatisierten Testmodus werden der String des erkannten Textes und der des originalen Textes durch Berechnung der Levenshtein-Distanz verglichen. Bei Fehlern, welche durch Texterkennung entstanden sind, ist die automatisierte Bewertung der Fehler schwierig. Neben Verwechslungen wie in Unterabschnitt 2.3.4 beschriebenen, gibt es auch fehlerhafte Erkennungen, bei denen die Anzahl der Fehler unterschiedlich definiert werden kann. Beispielsweise wird bei der Schriftart „New Times Roman“ das „M“ teilweise als „IVI“ (kleines “L“, großes „V“, kleines „L“) erkannt. Bei der Berechnung der Levenshtein-Distanz sind dabei drei Operationen notwendig, obwohl es sich dabei nur um ein falsch erkanntes Zeichen handelt.

[1] Quelle: Smith u. a. [2016] Wiki - Command Line Usage

The (quick) [brown] {fox} jumps! Over the $43,456.78 <lazy> #90 dog & duck/goose, as 12.5% of E-Mail from aspammer@website.com is spam. Der „schnelle“ braune Fuchs springt über den faulen Hund.

The (quick) [brown] {fox} jumps! Over the $43,456.78 <lazy> #90 dog & duck/goose, as 12.5% of E-Mail from aspammer@website.com is spam. Der „schnelle“ braune Fuchs springt über den faulen Hund.

The (quick) [brown] {fox} jumps! Over the $43,456.78 <lazy> #90 dog & duck/goose, as 12.5% of E-Mail from aspammer@website.com is spam. Der „schnelle“ braune Fuchs springt über den faulen Hund.

The (quick) [brown] {fox} jumps! Over the $43,456.78 <lazy> #90 dog & duck/goose, as 12.5% of E-Mail from aspammer@website.com is spam. Der „schnelle“ braune Fuchs springt über den faulen Hund.

The (quick) [brown] {fox} jumps! Over the $43,456.78 <lazy> #90 dog & duck/goose, as 12.5% of E-Mail from aspammer@website.com is spam. Der „schnelle“ braune Fuchs springt über den faulen Hund.

Abbildung 7.1.: Testtext in den verwendeten Schriftarten: OCR-A, OCR-B, Arial, Times New Roman, Lucida Handwriting (von oben)

7.2. Ergebnisse

Das Testverfahren wird für die verschiedenen Textausdrucke durchgeführt. Es werden jeweils fünf Fotos aufgenommen und von jedem Foto eine Texterkennung durchgeführt und die Levenshtein-Distanz des erkannten Textes zum Originaltext ausgegeben. Innerhalb dieser fünf Läufe gab es, trotz unveränderter Beleuchtung und Positionierung, teilweise große Unterschiede. In der folgenden Tabelle wird jeweils das beste Ergebnis (Lauf mit der niedrigsten Levenshtein-Distanz) festgehalten. Scheinbar fehlerhafte Tests (signifikant hohe Abweichung zum vorherigen oder folgenden Test) werden dabei maximal einmal wiederholt. Die Ursache der Abweichung innerhalb der fünf Läufe wird nicht weiter untersucht.

Zu allen Schriftarten wird die durchschnittliche Levenshtein-Distanz in allen verwendeten Schriftgrößen errechnet. Die Auswertung der Ergebnisse erfolgt im nächsten Abschnitt.

Die folgende Tabelle 7.1 zeigt die Ergebnisse der Testläufe:

	Schriftgröße [pt]											
	8		10		12		16		20		Durchschnitt	
	LD		LD		LD		LD		LD		LD	
Schriftart	abs	%	abs	%	abs	%	abs	%	abs	%	abs	%
OCR-A	16	8,0	19	9,5	33	16,5	29	14,5	26	13,0	24,6	12,3
OCR-B	6	3,0	4	2,0	5	2,5	15	7,5	10	5,0	8	4,0
Times New Roman	6	3,0	4	2,0	2	1,0	1	0,5	3	1,5	3,2	1,6
Arial	7	3,5	1	0,5	1	0,5	1	0,5	1	0,5	2,2	1,1
Lucida Handwriting	100	50,0	92	46,0	109	54,5	98	49,0	98	49,0	99,4	49,7

Tabelle 7.1.: Ergebnisse der Testläufe (LD: Levenshtein-Distanz, absolut und in Relation zur Zeichenanzahl des Testtextes)

7.3. Auswertung

Um einen besseren Überblick über die Ergebnisse zu bekommen, werden diese in der Abbildung 7.2 grafisch dargestellt:

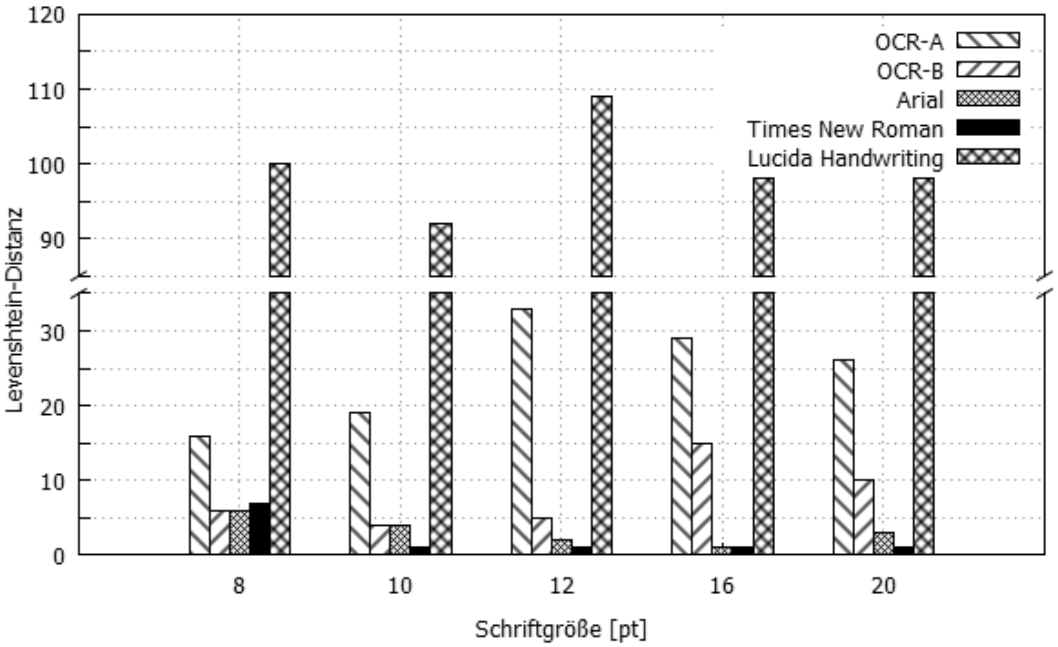


Abbildung 7.2.: Levenshtein-Distanz der Testläufe bei verschiedenen Schriftarten und -größen

Die signifikant schlechte Erkennung von Lucida Handwriting liegt daran, dass Tesseract nicht auf Handschriften ausgelegt ist. Obwohl es sich nicht um eine echte Handschrift handelt, wird der Text unabhängig von der Schriftgröße so schlecht erkannt, dass sich kaum die Sprache zuordnen lässt. Die Zahlen werden ähnlich gut wie bei den anderen Schriftarten erkannt.
Auch die Erkennung von OCR-A ist verhältnismäßig schlecht. Dies liegt hier vor allem an den Zahlen und Sonderzeichen. Da deren Darstellung von jener herkömmlicher Schriftarten stark abweicht und Tesseract nicht auf diese trainiert wurde, werden sie nicht erkannt. Die Erkennungsrate der Buchstaben liegt dabei etwa gleichauf mit der von OCR-B.
Die Erkennung der häufig verwendeten Schriftarten Arial und Times New Roman in den Schriftgrößen ab 10 pt ist gut. Insbesondere bei Times New Roman liegt die Erkennungsrate bei 99,5% und ist damit einwandfrei lesbar. Bei dem einen Fehler handelt es sich um das Dollarzeichen ($), welches in keinem der Testläufe korrekt erkannt wurde. Die meisten Läufe erkannten es als „S“, „5“ oder „8“. Mit dieser Erkennungsrate wäre ein Bildschirmlesegerät mit Texterkennung möglich, allerdings wird diese Genauigkeit nicht immer erreicht.

8. Schluss

Der Schluss dieser Arbeit fasst noch einmal alle Kapitel in Kürze zusammen und bildet ein Fazit in Bezug auf das entwickelte Projekt. Der danach folgende Ausblick geht auf Funktionen und Verbesserungen ein, die zukünftig realisiert werden können.

8.1. Zusammenfassung

Nach der Einleitung werden im zweiten Kapitel die Grundlagen erklärt, auf welche im weiteren Verlauf mehrmals zurückgegriffen wird, um Vorgänge zu erläutern. Im dritten Kapitel werden verschiedene Anforderungen an die Hardware und die Software gestellt. Im vierten Kapitel werden aufgrund dieser Anforderungen die Basiskomponenten der Arbeit ausgewählt. Aus diesen werden im fünften Kapitel Entwürfe für die Hard- und Software erstellt. Im sechsten Kapitel wurden diese präzise umgesetzt. Das siebte Kapitel testet die Hauptfunktion, das Erkennen von Text.

Durch die gute Umsetzung von Hard- und Software können die gestellten Anforderungen erfüllt und die Ziele erreicht werden. Die Gesamtzeit vom Start der Erkennung einer Seite DIN A5 bis zur Anzeige des Textes dauert, je nach Schriftgröße und damit Wortanzahl, 15-20 Sekunden.

Komponenten	Kosten in Euro
Raspberry Pi 2 Model B	38
Raspberry Pi Camera Module	27
Gehäuse + Netzteil	14
microSD Karte	10
Bodenplatte	3
Summe	92

Tabelle 8.1.: Aufstellung der Kosten des Hardwareaufbaus

Das in der Einleitung gestellte Preisziel wird ebenfalls erreicht. Die Kosten für den Hardwareaufbau betragen 92 Euro (vgl. Tabelle 8.1), worin der Monitorständer und die 3D-Drücke

nicht enthalten sind, da diese kostenfrei zur Verfügung gestellt wurden. Für einen fertigen Prototypen fehlt auch noch ein Monitor. Bei dem Preisziel von weniger als 300 Euro bleiben 200 Euro für einen Bildschirm. Für die Entwicklung und den Testlauf wurde ein vorhandener Monitor genutzt.

8.2. Fazit

Die Arbeit hat gezeigt, dass eine Texterkennung auf einem modernen Einplatinencomputer grundsätzlich möglich ist und auch die Erkennungszeit akzeptabel ist. Für den Einsatz als Bildschirmlesegerät ist dieser Ansatz im Nachhinein nur bedingt geeignet. Liegt die Erkennungsrate unter 99%, wird die Lesbarkeit stark beeinträchtigt. Problematisch bei der Verbesserung der Erkennungsrate ist vor allem, dass es keinen Erwartungswert gibt. Bei den in Unterabschnitt 2.3.2 genannten Anwendungen ist zumeist bekannt, wie der Text aufgebaut ist. Beispielsweise können bei einer Kennzeichenerfassung Sonderzeichen ausgeschlossen werden und es ist bekannt, in welcher Reihenfolge Buchstaben und Zahlen vorkommen. Dadurch, dass dies hier nicht der Fall ist, kann weder die Anzahl der möglichen Zeichen vor der Erkennung eingrenzt werden, noch eine gute Nachbearbeitung durchgeführt werden. Zwar ließen sich einige Regeln für richtig erkannte Wörter aufstellen, allerdings gibt es meistens eine Ausnahme.

Im Bereich des Preprocessing wurden bereits folgende Verbesserungsansätze verfolgt, welche aufgrund der schlechten Ergebnisse nicht weiter dokumentiert wurden:

Es wurde die Eingrenzung von Textfeldern mit einem abwandelten Algorithmus zur Kennzeichenerkennung versucht. Dieser Ansatz soll die relevanten Bildausschnitte mit Text filtern und so die Erkennungszeit reduzieren. Die Erkennung der Bereiche ist leider stark von der Schriftgröße abhängig und führt dazu, dass nur noch Texte in bestimmten Schriftgrößen als Text erkannt werden.

Im Zusammenhang mit den Textfeldern wurde die Parallelisierung der Erkennung durch Tesseract getestet, allerdings werden je nach Anzahl der Textfelder teilweise so viele Threads gestartet, dass die Zeit für die Suche und Erkennung der Felder größer ist, als die für eine Erkennung des Gesamtbildes nötige Zeit.

Eine Erkennung der Drehung des Textes und Korrektur der Ausrichtung sollte die Erkennung verbessern. Die Erkennung funktioniert nur für kleine Winkel und führt aufgrund des durch die Drehung entstandenen Aliasing zu einer Verschlechterung der Erkennung. Zusätzlich kann Tesseract eine Drehung von wenigen Grad selbst korrigieren.

Die Probleme der Erkennung hängen meist mit den in Unterabschnitt 2.3.4 erläuterten Erkennungsproblemen zusammen. Diese Probleme existieren seit den Anfängen der Texterkennung. Zwar ist die Rechenleistung moderner Systeme wesentlich gestiegen (und im zeitlichen Vergleich ist auch die eines Einplatinencomputer hoch), allerdings werden heute deutlich mehr Schriftarten genutzt.
Zu Beginn dieser Arbeit wurde fälschlicherweise angenommen, dass die Texterkennung an sich im Vergleich zu den anderen Operationen, wie Videoanzeige oder das Anwenden von Filtern, wesentlich mehr Rechenzeit benötigen wird. Das Gegenteil ist der Fall. Trotz der nicht erfüllten persönlichen Erwartungen, wird die Entwicklung und der Aufbau des Prototypen als erfolgreich betrachtet.

8.3. Ausblick

Durch den begrenzten Zeitrahmen der Arbeit, konnten Ansätze und Ideen mit geringer Priorität nicht weiter verfolgt werden. In diesem Ausblick werden Möglichkeiten zur Weiterentwicklung aufgezeigt.

Im Bereich der Hardware lässt sich die Rechenleistung durch einen Wechsel auf den Raspberry Pi 3 Model B steigern. Dieses wurde am 29.02.2016 vorgestellt und bietet einen neuen SoC mit 64-bit Architektur, 33% höherer Taktfrequenz, sowie integriertem WLAN und Bluetooth, bei gleichem Preis (Broadcom BCM2837 mit $4\times1{,}2$ GHz, ARM Cortex-A53)[1]. Bei einem Wechsel auf dieses Modell kann der restliche Aufbau beibehalten werden, da das Layout mit dem des hier verwendeten Modells übereinstimmt. Durch den höheren Takt würde sich die Dauer der Texterkennung voraussichtlich reduzieren.

Im Bereich der Software entwickelte sich im Verlauf dieser Arbeit eine alternative Idee zur Umsetzung von Erkennung und Ausgabe. Diese Idee setzt nach dem manuellen Auslösen der Texterkennung an. Sie sieht vor, dass die Bildaufnahme und das Preprocessing in einem Thread ausgeführt werden. Parallel dazu soll ein weiterer Thread aus dem niedrig aufgelösten Bild der Vorschau (letztes Bild vor dem Auslösen) durch Bildverarbeitung die Orientierung erkennen, die Zeilen separieren und die Position der einzelnen Wörter erfassen. Anschließend soll die Position der Wörter im Standbild errechnet und gespeichert werden. Durch die geringere Auflösung (Faktor 15 zwischen Videobild und Standbild) wird geschätzt, dass beide Threads etwa die gleiche Zeit benötigen werden. Das Hauptprogramm wartet derweil auf die Beendigung beider Threads und zeigt das Standbild an. Dann startet es die Texterkennung der einzelnen Wörter in einzelnen Threads. Von diesen sollen maximal drei parallel laufen, damit dem Hauptprogramm genügend Rechenleistung zugeteilt werden kann. Dieses zeigt indes das gefilterte (adaptive Thresholding) Standbild an und ersetzt nach und nach die

[1] Quelle: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Wörter des Fotos durch die von den Threads erkannten Wörter (entsprechende Schriftgröße wird auch berechnet).
Dieser Ansatz hat zwei Vorteile. Erster Vorteil: Die ersten Wörter und Zeilen werden wesentlich früher durch scharfe Schrift dargestellt. Zweiter Vorteil: Durch die parallele Texterkennung durch drei Threads kann die gesamte Erkennungszeit theoretisch um 66% reduziert werden. Geschätzt wird eine reale Reduzierung um 40-50%. Im Idealfall könnten somit, während die ersten Zeilen gelesen werden, bereits die nächsten Zeilen erkannt werden.

Als Ansatz zur Verbesserung der Erkennungsrate könnte man ausgewählte Schriftarten trainieren. Um eine Auswahl treffen zu können, muss allerdings zuvor das Anwendungsfeld eingeschränkt werden.

Zur Verbesserung der Textausgabe kann der Entwurf mit der Nutzung von Qt umgesetzt werden. Dies ermöglicht die Ausgabe jeglicher Zeichen aus dem UTF-8 Zeichensatz.

Literaturverzeichnis

[Blanchette und Summerfield 2009] BLANCHETTE, J. ; SUMMERFIELD, M.: *C++ GUI Programmierung mit Qt 4: Die offizielle Einführung*. Pearson Deutschland, 2009 (Programmer's Choice Addison Wesley). – URL https://books.google.de/books?id=T373zcyFfvoC. – ISBN 9783827327291

[Brahmbhatt 2013] BRAHMBHATT, S.: *Practical OpenCV*. Apress, 2013 (Practical OpenCV). – URL https://books.google.de/books?id=M_FZAgAAQBAJ. – ISBN 9781430260790

[Dembowski 2013] DEMBOWSKI, K.: *Raspberry Pi - Das Handbuch: Konfiguration, Hardware, Applikationserstellung*. Springer Fachmedien Wiesbaden, 2013. – URL https://books.google.de/books?id=fwXnAwAAQBAJ. – ISBN 9783658031671

[Duschl 2014] DUSCHL, D.: *Softwareentwicklung mit C++: Einführung mit Visual Studio 2012*. Springer Fachmedien Wiesbaden, 2014 (Studienbücher Informatik). – URL https://books.google.de/books?id=pCgkBAAAQBAJ. – ISBN 9783658015862

[Follmann 2014] FOLLMANN, R.: *Das Raspberry Pi Kompendium*. Springer Berlin Heidelberg, 2014 (Xpert.press). – URL https://books.google.de/books?id=dUr7AwAAQBAJ. – ISBN 9783642549113

[Foster 2014] FOSTER, E.: *Software Engineering: A Methodical Approach*. Apress, 2014 (SpringerLink : Bücher). – URL https://books.google.de/books?id=RWMnCgAAQBAJ. – ISBN 9781484208472

[Gay 2015] GAY, W.: *Exploring the Raspberry Pi 2 with C++*. Apress, 2015. – URL https://books.google.de/books?id=xdELCwAAQBAJ. – ISBN 9781484217399

[Kampffmeyer u. a. 1994] KAMPFFMEYER ; ANDERE ; GEMEINSAME PROJEKTGRUPPE AUTOMATISCHE ERFASSUNG VON SCHRIFTGUT DER AWV-FACHAUSSCHÜSSE OPTISCHE INFORMATIONSSYSTEME UND BÜROORGANISATION, INFORMATIONSVERWALTUNG: *Automatische Erfassung von Schriftgut: Strukturierung von Schriftgutvorlagen für die automatische elektronische Erfassung ; technische Grundlagen, Gestaltung von Schriftgut, elektronische Erfassung und Erschliessung*. AWV-Eigenverl., 1994 (AWV-Schriftenreihe:

Ausschuß für Wirtschaftliche Verwaltung). – URL https://books.google.de/books?id=1ezkGwAACAAJ

[Kampffmeyer 1996] KAMPFFMEYER, Dr. U.: *Erfassung und Erschließung von Dokumenten ; Dokumentenmanagement*. 1996. – URL http://www.project-consult.net/Files/Download_Erfassung_Erschlie%C3%9Fung_Kff_2_96.pdf. – Zugriffsdatum: 17.02.2016

[Kleuker 2008] KLEUKER, S.: *Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten*. Vieweg+Teubner Verlag, 2008. – URL https://books.google.de/books?id=4pz9-cfEAe0C. – ISBN 9783834803917

[Kleuker 2009] KLEUKER, S.: *Formale Modelle der Softwareentwicklung: Model-Checking, Verifikation, Analyse und Simulation*. Vieweg+Teubner Verlag, 2009 (Datenbanken und Softwareentwicklung). – URL https://books.google.de/books?id=XNtzT8ADSn8C. – ISBN 9783834806697

[Membrey und Hows 2013] MEMBREY, P. ; HOWS, D.: *Learn Raspberry Pi with Linux*. Apress, 2013 (SpringerLink : Bücher). – URL https://books.google.de/books?id=N2QTLjo0NSUC. – ISBN 9781430248224

[Mori u. a. 1992] MORI, Shunji ; SUEN, Ching Y. ; YAMAMOTO, Kazuhiko: Historical review of OCR research and development. In: *Proceedings of the IEEE* 80 (1992), Jul, Nr. 7, S. 1029–1058. – ISSN 0018-9219

[Nischwitz u. a. 2011] NISCHWITZ, A. ; FISCHER, M. ; HABERÄCKER, P. ; SOCHER, G.: *Computergrafik und Bildverarbeitung*. Vieweg+Teubner Verlag, 2011 (Computergrafik und Bildverarbeitung / Alfred Nischwitz). – URL https://books.google.de/books?id=LNCjfPfx7QkC. – ISBN 9783834883001

[Rice u. a. 2012] RICE, S.V. ; NAGY, G. ; NARTKER, T.A.: *Optical Character Recognition: An Illustrated Guide to the Frontier*. Springer US, 2012 (The Springer International Series in Engineering and Computer Science). – URL https://books.google.de/books?id=O_kGCAAAQBAJ. – ISBN 9781461550211

[Schmidhuber 2012] SCHMIDHUBER, Jurgen: Multi-column Deep Neural Networks for Image Classification. In: *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Washington, DC, USA : IEEE Computer Society, 2012 (CVPR '12), S. 3642–3649. – URL http://dl.acm.org/citation.cfm?id=2354409.2354694. – ISBN 978-1-4673-1226-4

[Smith 2007] SMITH, Raymond W.: An Overview of the Tesseract OCR Engine. In: *Proceedings of the Ninth International Conference on Document Analysis and Recognition -*

Volume 02. Washington, DC, USA : IEEE Computer Society, 2007 (ICDAR '07), S. 629–633. – URL http://dl.acm.org/citation.cfm?id=1304596.1304846. – ISBN 0-7695-2822-8

[Smith 2009] SMITH, Raymond W.: Hybrid Page Layout Analysis via Tab-Stop Detection. In: *Proceedings of the 2009 10th International Conference on Document Analysis and Recognition*. Washington, DC, USA : IEEE Computer Society, 2009 (ICDAR '09), S. 241–245. – URL http://dx.doi.org/10.1109/ICDAR.2009.257. – ISBN 978-0-7695-3725-2

[Soille 2013] SOILLE, P.: *Morphologische Bildverarbeitung: Grundlagen, Methoden, Anwendung*. Springer Berlin Heidelberg, 2013. – URL https://books.google.de/books?id=uEapBgAAQBAJ. – ISBN 9783642721908

[Unnikrishnan und Smith 2009] UNNIKRISHNAN, Ranjith ; SMITH, Ray: Combined Script and Page Orientation Estimation Using the Tesseract OCR Engine. In: *Proceedings of the International Workshop on Multilingual OCR*. New York, NY, USA : ACM, 2009 (MOCR '09), S. 6:1–6:7. – URL http://doi.acm.org/10.1145/1577802.1577809. – ISBN 978-1-60558-698-4

[Waldraff 2013] WALDRAFF, T.: *Digitale Bildauflösung: Grundlagen, Auflösungsbestimmung, Anwendungsbeispiele*. Springer Berlin Heidelberg, 2013 (X.media.press). – URL https://books.google.de/books?id=qQ8dBgAAQBAJ. – ISBN 9783642185809

Internetquellen

[ABBYY 2016a] ABBYY: *ABBYY FineReader*. 2016. – URL http://www.abbyy.com/finereader/. – Zugriffsdatum: 02.04.2016

[ABBYY 2016b] ABBYY: *What is OCR and OCR Technology*. 2016. – URL http://www.abbyy.com/finereader/about-ocr/what-is-ocr/. – Zugriffsdatum: 02.04.2016

[Banana Pi Team 2015] BANANA PI TEAM: *What's the BPi-M2?* 2015. – URL http://www.bananapi.com/index.php/component/content/article?layout=edit&id=73. – Zugriffsdatum: 02.04.2016

[Beinert 2015] BEINERT, Wolfgang: *OCR SCHRIFTEN*. 2015. – URL http://www.typolexikon.de/ocr-schriften/. – Zugriffsdatum: 01.04.2016

[Belam 2012] BELAM, Martin: *The Raspberry Pi: reviving the lost art of children's computer programming*. 2012. – URL http://www.theguardian.com/commentisfree/2012/feb/29/rasperry-pi-childrens-programming. – Zugriffsdatum: 27.01.2016

[Breuel 2016] BREUEL, Prof. Dr. T.: *Python-based tools for document analysis and OCR*. 2016. – URL https://github.com/tmbdev/ocropy. – Zugriffsdatum: 02.04.2016

[Carnegie Mellon University 2008] CARNEGIE MELLON UNIVERSITY: *What is reCAPTCHA*. 2008. – URL http://recaptcha.net/learnmore.html. – Zugriffsdatum: 12.02.2009

[Dütsch 2016] DÜTSCH, Birgit: *Bildschirmlesegeräte*. 2016. – URL http://www.incobs.de/produktgruppen/items/bildschirmlesegeraete.html. – Zugriffsdatum: 30.01.2016

[Embedded Linux Wiki 2016] EMBEDDED LINUX WIKI: *RPi Hardware*. 2016. – URL http://elinux.org/RPi_Hardware. – Zugriffsdatum: 27.01.2016

[H. 2015] H., Mike: *Ipevo Ziggi HD document camera - Libwebcam patch, focussing from software*. Juni 2015. – URL https://sourceforge.net/p/linux-uvc/mailman/message/34192575/. – Zugriffsdatum: 02.04.2016

[Hardkernel 2015] HARDKERNEL: *ODROID-C1+ Technical Detail*. 2015. – URL http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143703355573&tab_idx=2. – Zugriffsdatum: 01.04.2016

[Hirzel Optik 2014] HIRZEL OPTIK: *Das elektronische leistungsfähige Bildschirmlesegerät Topaz*. 2014. – URL http://www.hirzel-optik.ch/wp-content/uploads/2014/05/Topaz.pdf. – Zugriffsdatum: 30.01.2016

[Linux Hardware Guide 2012] LINUX HARDWARE GUIDE: *Logitech C525 HD Webcam*. Juli 2012. – URL http://linux-hardware-guide.de/2012-07-28-logitech-c525-hd-webcam. – Zugriffsdatum: 02.04.2016

[Logitech 2016] LOGITECH: *HD WEBCAM C525*. 2016. – URL http://www.logitech.com/de-de/product/hd-webcam-c525. – Zugriffsdatum: 02.04.2016

[Mendelson 2014] MENDELSON, Edward: *Abbyy FineReader 12 Professional*. 2014. – URL http://www.pcmag.com/article2/0,2817,2468186,00.asp. – Zugriffsdatum: 02.04.2016

[Rosebrock 2015] ROSEBROCK, Adrian: *How to install OpenCV 3 on Raspbian Jessie*. 2015. – URL http://www.pyimagesearch.com/2015/10/26/how-to-install-opencv-3-on-raspbian-jessie/. – Zugriffsdatum: 28.03.2016

[Smith u. a. 2016] SMITH, Ray ; ABDULKADER, Ahmad ; ANTONOVA, Rika ; BEATO, Nicholas ; CHARRON, Samuel ; CHEATLE, Phil ; CROUCH, Simon ; EGER, David ; HUDDLESTON, Sheelagh ; JOHNSON, Dan ; KIELBUS, Thomas ; LEE, Dar-Shyang ; LIU, Zongyi (. ; MOSS, Robert ; NEWTON, Chris ; REIMER, Michael ; RENN, Marius ; ROMANO, Raquel ; RUSSON, Christy ; SAXENA, Shobhit ; SEAMAN, Mark ; SHAFAIT, Faisal ; TAKENAKA, Hiroshi ; UNNIKRISHNAN, Ranjith ; WANKE, Joern ; XIU, Ping P. ; ZIEM, Andrew ; ZUNIGA, Oscar: *Tesseract Open Source OCR Engine (main repository)*. 2016. – URL https://github.com/tesseract-ocr/tesseract. – Zugriffsdatum: 02.04.2016

[Sourceforge 2014] SOURCEFORGE, Kbarni: *release function not implemented in raspicam_still and raspicam_still_cv*. 2014. – URL https://sourceforge.net/p/raspicam/tickets/7/. – Zugriffsdatum: 02.04.2016

[Upton 2013] UPTON, Liz: *Macro filming and photography with the camera board*. Juni 2013. – URL https://www.raspberrypi.org/blog/macro-photography-with-the-camera-board/. – Zugriffsdatum: 02.04.2016

A. CD

Auf der letzten Seite dieser Arbeit ist eine Daten-CD eingeklebt, welche für diese Arbeit relevante Informationen enthält. Aufbau und Inhalt der CD sind hier kurz aufgeführt.

- Dieses Dokument als PDF (BA.pdf)
- Genutzte Internetquellen als PDF (Ordner „1 Internetquellen“)
- Verwendete und weitere Fotos und Grafiken (Ordner „2 Fotos und Grafiken“)
- Modelle für den 3D-Druck (Ordner „3 Modelle 3D Druck“)
- Software-Projekt mit Quellcodes (Ordner „4 NetBeansProjects“)

Die beiden Exemplare dieser Arbeit mit dem beschriebenen Datenträger befinden sich zur Einsicht beim Erst- und Zweitprüfer.

B. Programmablaufpläne zu Unterabschnitt 5.2.1

Auf den folgenden Seiten befinden sich die Programmablaufpläne der im Unterabschnitt 5.2.1 beschriebenen Teilentwürfe des Softwareentwurfes.

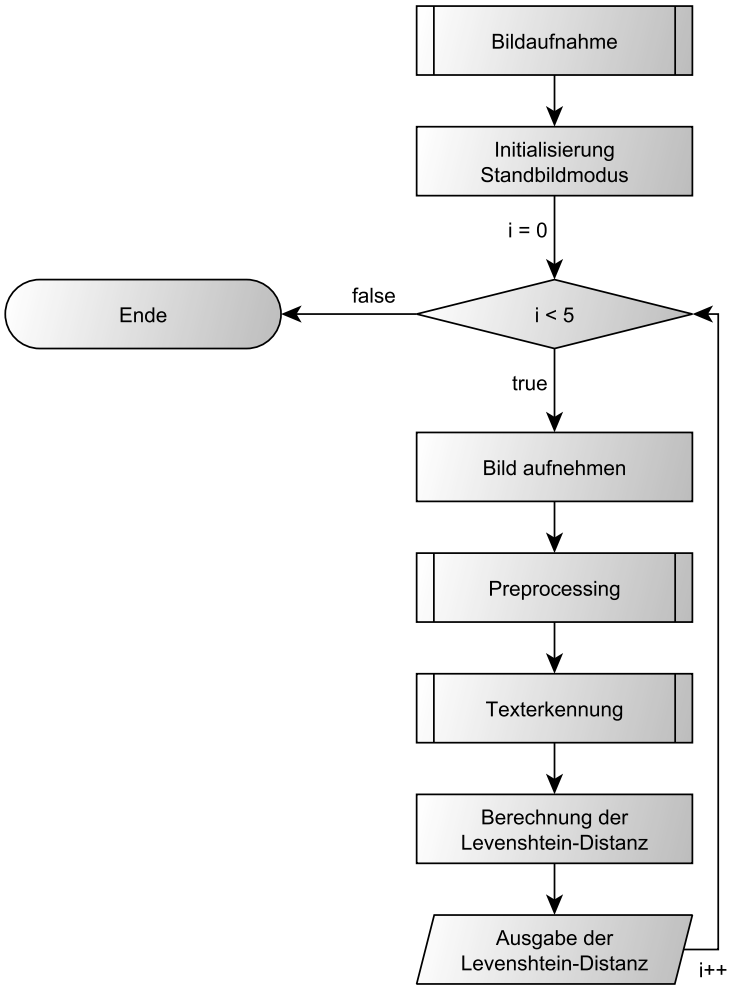


Abbildung B.1.: Programmablaufplan des Testmodus

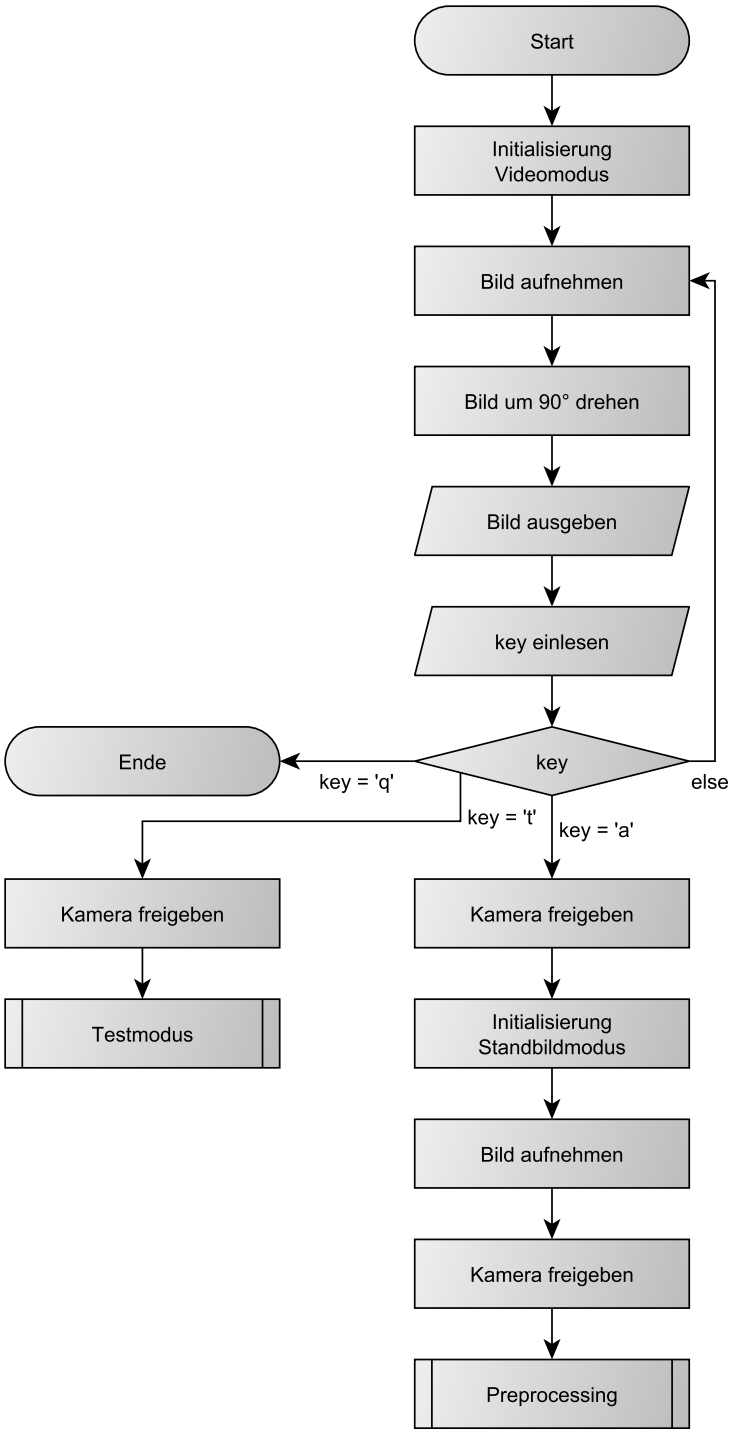


Abbildung B.2.: Programmablaufplan der Bildaufnahme, „key“ steht für eine auf der Tastatur betätigte Taste

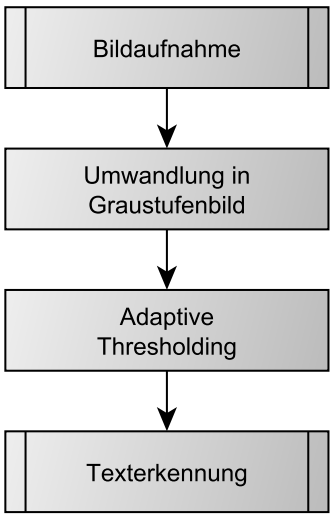


Abbildung B.3.: Programmablaufplan des Preprocessing

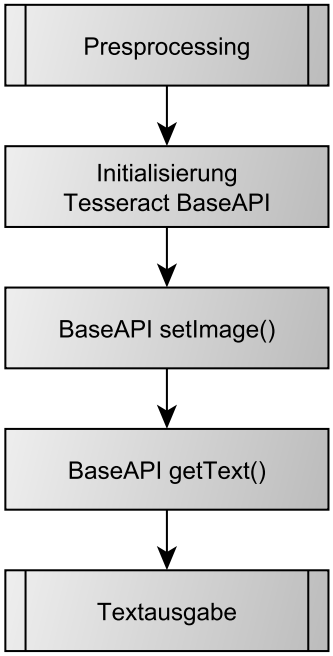


Abbildung B.4.: Programmablaufplan der Texterkennung

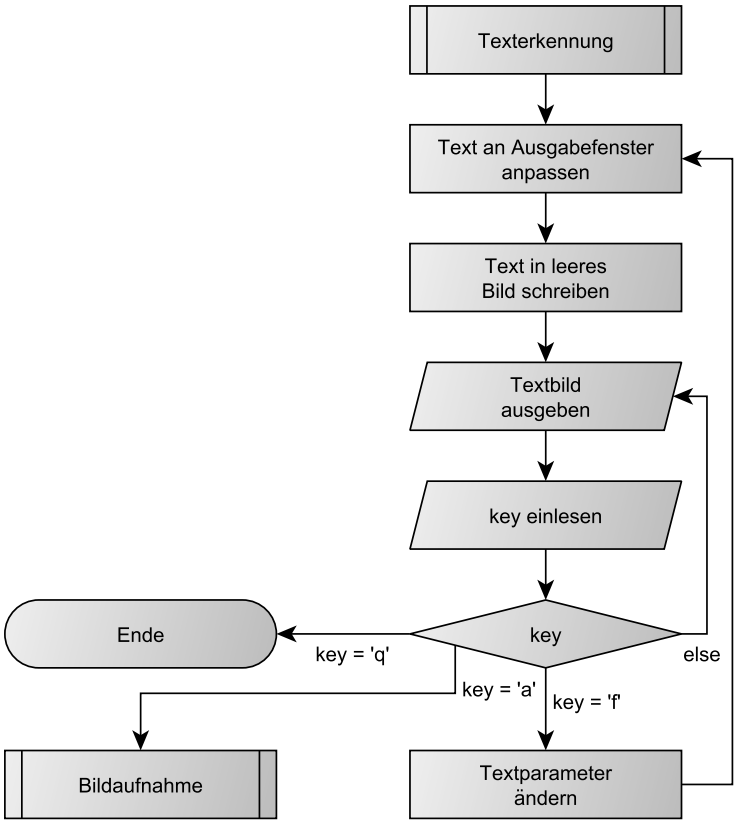


Abbildung B.5.: Programmablaufplan der Textausgabe, „key“ steht für eine auf der Tastatur betätigte Taste

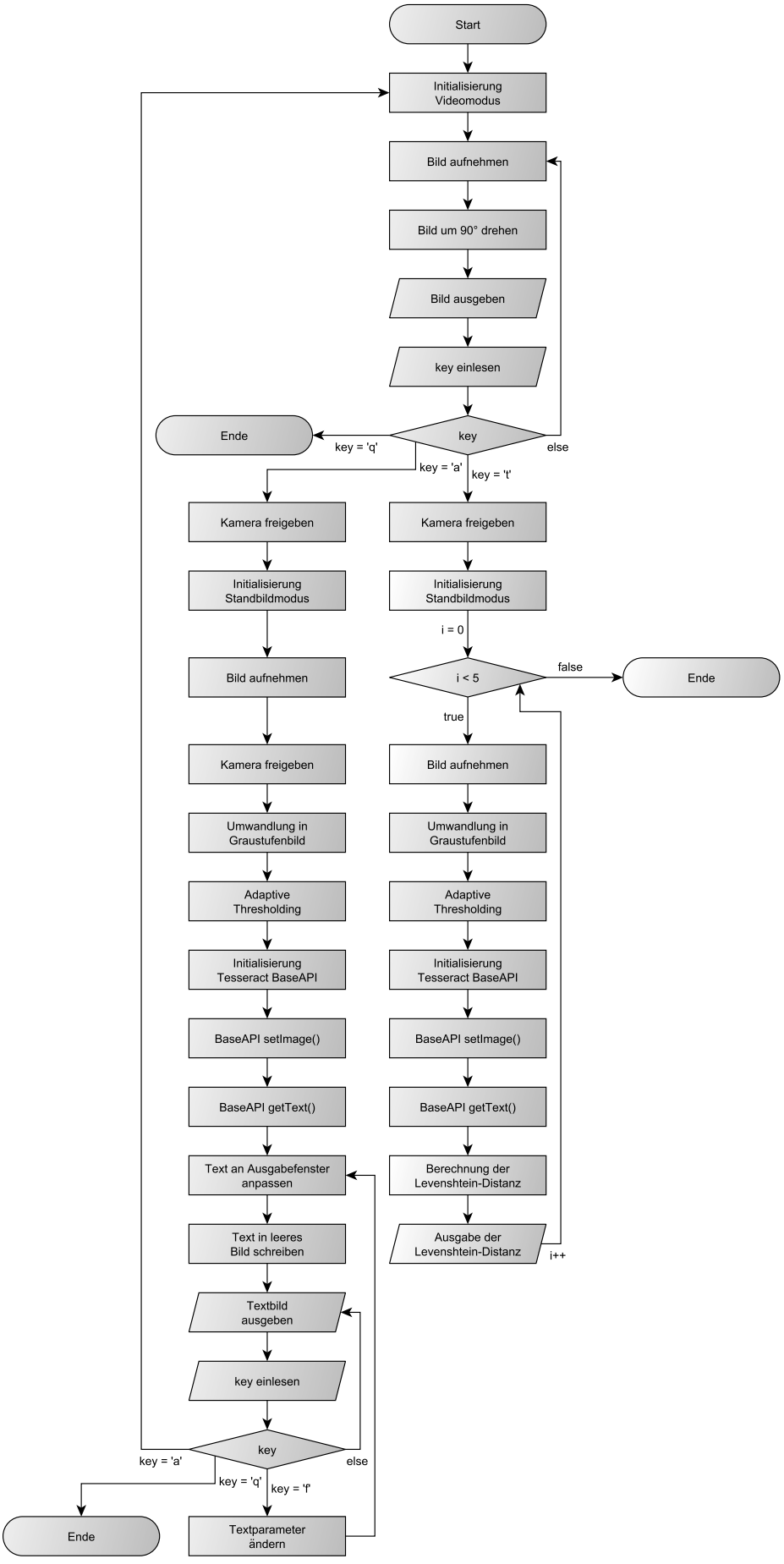


Abbildung B.6.: Vollständiger Programmablaufplan, „key“ steht für eine auf der Tastatur betätigte Taste

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 4. April 2016	
Ort, Datum	Unterschrift