UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Online Handschrifterkennung chinesischer Schriftzeichen auf androidfähigen mobilen Endgeräten

Bachelorarbeit
zur Erlangung des Grades
Bachelor of Science
im Studiengang Computervisualistik

vorgelegt von

Shuyi Weng

Betreuer: Prof. Dr. Karin Harbusch, Prof. Dr.-Ing. Dietrich Paulus Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Erstgutachter: Prof. Dr. Karin Harbusch, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Zweitgutachter: Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau

Koblenz, im Juli 2014

Kurzfassung

Um mobile Wörterbücher oder Übersetzer zu verwenden, braucht es eine Eingabe. Diese muss zuvor verarbeitet werden, um nutzbar zu sein. Für chinesische Zeichen bietet sich die Handschrift an, da die Schrift hauptsächlich aus Piktogrammen und Ideogrammen besteht.

In dieser Bachelorarbeit wird ein prototypisches Erkennungssystem auf einem mobilen Endgerät implementiert. Die Erkennung soll dabei online und somit während des Schreibens erfolgen. Dies kann dem Benutzer Zeit ersparen, indem verschiedene erkannte Vorschläge zur Laufzeit gegeben werden.

Es werden Grundlagen erläutert und ein Überblick über den aktuellen Stand der Forschung gegeben. Ein Ansatz wird ausgewählt und implementiert, der möglichst schnell ist und wenig Speicherplatz erfordert. Die Implementation wird getestet und es wird gezeigt, dass es möglich ist, eine schnelle Erkennung auf einem kleinen Gerät laufen zu lassen. Es werden Verbesserungen und Erweiterungen vorgeschlagen, sowie ein Ausblick gegeben.

Abstract

Usage of mobile dictionaries or translators requires an input. This input has to be processed and recognized beforehand. Chinese characters are more suited for a handwritten input than a keyboard based one. Reason for that are the characters consisting mostly of pictograms or ideograms.

This thesis deals with an implementation of a prototypical recognition system on a mobile device. The recognition process should be online and therefore running while writing. It can save time for the user, because suggestions are made during runtime.

Basics and an overview over the current state of the art in online handwriting recognition will be given. An approach will be chosen and implemented, such that the recognition process is fast and needs little memory. The implementation will be tested and it will show, that a fast recognition can be possible on small devices. Suggestions for expansions and improvements will be given, including a future work part.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Vereinbarung der Arbeitsgruppe für Studien- und Abschlussarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

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

Koblenz, den 30. Juli 2014

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im Zeitalter der zunehmenden Globalisierung und Internationalisierung ist es nicht mehr unüblich, für den Urlaub, Arbeitsaufenthalt oder zwecks Studium das Land zu wechseln, z.B. nach China. Bei solchen Möglichkeiten ist es sehr praktisch, ein mobiles Wörterbuch bei sich zu tragen. Smartphones sind dafür ideal geeignet und ihre Nutzung hat in den letzten Jahren stark zugenommen.

Chinesische Schriftzeichen bestehen zum Großteil aus Ideogrammen und Piktogrammen. Dadurch ist das Abzeichnen und das handschriftliche Schreiben der Symbole einfach gestaltet. Eine Tastatur würde wenig nutzen, da es 5000 Schriftzeichen für den Grundgebrauch gibt. Für westliche Sprachen würde sich eine Eingabe über Tastatur hinsichtlich Schnelligkeit besser eignen [TSW90], da hier nur 26 Buchstaben existieren, aus denen jedes Wort besteht.

Das Problem bei der Erkennung sind die Schreibregeln für jedes Zeichen, die ein nicht-nativer Schreiber nicht kennt. Hauptprobleme sind dabei die Ordnung der Striche und die Menge an Zeichen [TSW90, DLX07, LJN04].

Um eine Übersetzung zu ermöglichen, muss zunächst die Erkennung stattfinden. Dieser Prozess soll in der Arbeit genauer untersucht werden.
Die Schrifterkennung an sich ist bereits von gesellschaftlicher Bedeutung. Im Bereich der Mensch-Maschine-Kommunikation werden solche Systeme beispielsweise bei der Briefsortierung oder Dokumenten- und Formularprüfung etc. bereits genutzt [DLX07].
Allgemein kann der Beginn der online Handschrifterkennung in den späten 50er Jahren angesetzt werden und erlebte einen neuen Schwung an Interesse ab den 80ern [TSW90]. Die konstante Verbesserung von Software und Hardware für Handys, sowie Tablets führt zu vielfältigen Forschungen in diesem Gebiet. Das Betriebssystem Android für Smartphones verhilft dabei zu einheitlichen und einfachen Programmen.

Der Begriff online bedeutet dabei, dass der Erkennungsprozess in Echtzeit während des Schreibens abläuft. Im Gegensatz dazu erfolgt eine offline Erkennung

erst, nachdem das Zeichen vollständig geschrieben wurde. Der Vorteil der online Erkennung liegt zum einen in den gegebenen Rohdaten. Es können Koordinaten, Richtungen, Druck des Eingabestiftes oder Strichreihenfolge aufgenommen werden. Zum anderen spart es dem Benutzer Zeit, wenn während des Schreibens bereits das gewünschte Zeichen als Vorschlag auftaucht.

In der Arbeit soll der aktuelle Stand aufgearbeitet und ein Überblick über den Forschungsstand der online Handschrifterkennung, besonders von chinesischen Zeichen, gegeben werden. Bisherige Systeme weisen eine hohe Erkennungsrate auf, sind jedoch nicht oder nur teilweise auf kleinen Endgeräten implementiert worden.

Um die Arbeit zu realisieren, wurden zunächst Grundlagen aufgearbeitet. Diese werden in den nächsten Kapiteln 2, 3, 4 und 5 erläutert. Dazu zählen die chinesische Schrift, das Betriebssystem Android, die Hidden-Markov-Modelle und die online Handschrifterkennung. Für den praktischen Teil wird ein aktuelles Verfahren gewählt, welches sich auf mobilen Endgeräten implementieren lässt, ohne viel Kapazität und Leistung zu beanspruchen. Dabei wird anstatt eines Referenzmodells für jedes Zeichen die Zerlegung der Schrift in Elementarteile verwendet. Dieses Verfahren wird in Kapitel 6 erläutert.
Der praktische Teil ist die Implementation eines Prototypen zur Erkennung. Das Training und die Umsetzung werden in 7 und 8 dargestellt. Anschließend wird in 9 die Erkennungsrate getestet und eine Zeitmessung durchgeführt. Dabei erfolgt eine Bewertung der Ergebnisse und es wird gezeigt, dass ein schnelles Erkennungssystem durchaus auf kleinen Geräten möglich ist. Danach erfolgt ein Abschnitt über eventuelle Erweiterungen und auch Verbesserungsvorschläge, um die Erkennungsrate zu steigern.
Der Schluss bildet eine Zusammenfassung der Arbeit und einen Ausblick für zukünftige Möglichkeiten.

Kapitel 2

Chinesische Schriftzeichen

2.1 Allgemein

Auf der Basis von chinesischen Schriftzeichen kommuniziert rund ein Viertel der Weltbevölkerung. Davon ist der Großteil in Asien zu finden, wie z.B. China, Japan, Taiwan oder Korea [LJN04].

Der Ursprung der Zeichen lässt sich bis 1000 Jahre v.Chr. zurückführen. Eine offizielle Schrift wurde in der Qin Dynasty 220 v.Chr. eingeführt. Seither verblieb die Schrift mit wenigen Änderungen. Erst 1956-1964 wurde eine Vereinfachung der chinesischen Zeichen eingesetzt, da viele der traditionellen Schriftzeichen aufgrund ihrer hohen Strichanzahl kompliziert waren [DLX07].
Durch die Vereinfachung existieren nun hauptsächlich 3 Gruppen von Zeichen: traditionelles Chinesisch, vereinfachtes Chinesisch und japanische Kanji.

Kanji sind ein Teil der japanischen Sprache und wurden aus dem Chinesischen adaptiert. Die Kanji sind teilweise, vereinfachtes teilweise traditionelles Chinesisch mit leichten Variationen [JLN03]. Im Allgemeinen haben Kanji als auch chinesische Zeichen die gleiche Bedeutung. In Studien um japanische Zeichenerkennung werden explizit Kanji benannt, ansonsten spricht man von chinesischen Zeichen. [TSW90].

2.2 Chinesischer Zeichensatz

Es gibt Zeichensätze, welche bestimmte Symbole beinhalten. Im offiziellen nationalen Standardsatz GB2312-80 sind 3755 Zeichen auf Level 1 bzw. 3008 Zeichen auf Level 2 benannt. In China kann mithilfe von den 6763 Zeichen aus der Serie GB2312-80 circa 99% des allgemeinen Sprachgebrauchs abgedeckt werden [DLX07]. Das Basisvokabular liegt bei 3000 bis 5000 Zeichen. Insgesamt gibt es jedoch an die 50000 Symbole [TSW90].

Als Konsequenz muss eine Software für Schrifterkennung mindestens 5000 Zeichen für den alltäglichen Gebrauch entziffern können.

2.3 Vergleich - Aufbau der westlichen und chinesischen Zeichen

Die chinesische Schrift unterscheidet sich von den westlichen Schriften. Ein Beispiel für eine westliche Sprache wäre Englisch. Es gibt hier 26 elementare Buchstaben in Groß- und Kleinschreibung. Großbuchstaben werden in der Regel mit zwei Linienzügen geschrieben, kleine Buchstaben mit einem. Die Wörter bestehen im Durchschnitt aus einer Sequenz von fünf Buchstaben. Wichtig sind bei der Wortbildung die Position und Größe der Buchstaben. So liegen zum Beispiel Großbuchstaben immer auf der Basislinie und haben volle Größe in einer Zeile. Darüber liegt die Korpuslinie, welche die Grenze für die Kleinbuchstaben bildet [TSW90].

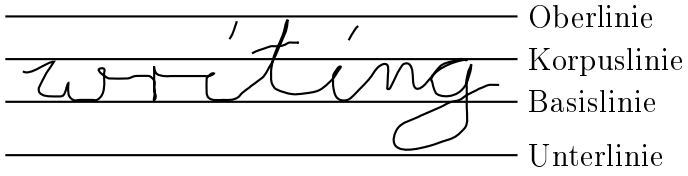

Abbildung 2.1: Ein Beispiel für ein westliches Wort mit Linienbeschriftung

Des Weiteren gibt es sogenannte Oberlinien bzw. Unterlinien. Buchstaben wie *t* oder *g* reichen jeweils an diese oberste bzw. unterste Linie. Das Wort erhält verschiedene Höhen und Tiefen. Die Länge eines Wortes kann zudem variabel sein, da eine Aneinanderreihung von entsprechenden Nomen lange Sequenzen ergibt [JLN03].

Im chinesischen ist ein ganzes Schriftzeichen ein ganzes Wort. Es existiert ein phonetisches Alphabet, welches Pinyin genannt wird und nur zur Aussprache dient. Für jedes Schriftzeichen gibt es eine bestimmte Art und Weise, dieses auszusprechen, zu sehen in Abbildung 2.2.

chuáng

Abbildung 2.2: Das Schriftzeichen für das Wort *Bett* in einer Standardbox

Die Schriftzeichen bestehen aus einzeln gezogenen Linien, welche im Durchschnitt bei 8-10 Strichen liegen. Das komplizierteste Symbol besitzt 36 Striche und das einfachste 1 Strich [TSW90, DLX07]. Dabei orientieren sich die Zeichen in einer quadratischen Box [JLN03]. Im Gegensatz dazu werden westliche Wörter auf einer geraden Linie geschrieben.

Die Anzahl und Position variiert in verschiedenen Zeichen. Die Striche dafür, können gerade durchgezogene Linien oder sogenannte Poly-Striche sein [LJN04]. Ein Beispiel zu den Linien ist in Abbildung 2.3 zu finden.

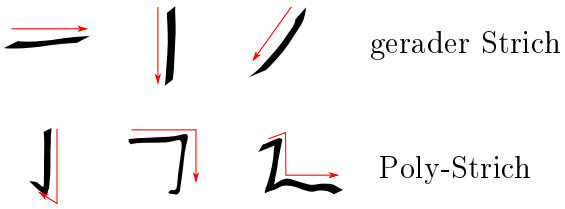

Abbildung 2.3: Beispiele für gerade Striche und Poly-Striche

Dabei ist der senkrechte gerade Strich ähnlich dem Poly-Strich mit einem Haken. Der zweite kann als eine Kombination aus dem waagerechten und dem senkrechten Strich ohne Abzusetzen gesehen werden. Das letzte Beispiel besitzt eine senkrechte und waagerechte Richtung, wobei die einzelnen Formen keiner geraden Linie entsprechen.

Hinzu kommt, dass ein einzelnes Zeichen aus anderen Zeichen und aus unabhängigen Substrukturen bestehen kann. Eine solche Substruktur wird Radikal genannt. Diese sind Grundzeichen, aus denen andere Zeichen komponiert werden können. Einige der Radikale fungieren als eigenständiges Schriftzeichen und andere sind nur eine Kombination aus Strichen bzw. Poly-Strichen.
Durch die bausteinartige Zusammensetzung entsteht eine hierarchische Struktur. Dieser Aufbau kann für die Erkennung praktisch sein, da die Anzahl der Referenzmodelle verringert werden kann.

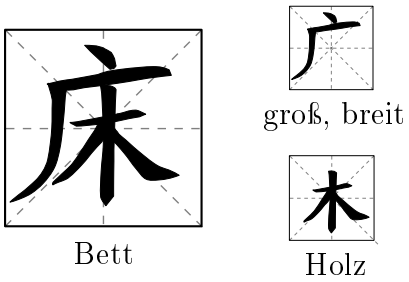

Abbildung 2.4: Das Zeichen für *Bett* besteht aus dem Zeichen *Holz* und dem Radikal für *breit*

Neben den Komponenten, aus denen ein Zeichen besteht, gibt es noch eine bestimmte Reihenfolge, in der diese aufgeschrieben werden. Solch eine Schreibregel gibt es für jedes Zeichen. Diese werden in der Schule beigebracht, sowohl für Kanji als auch für chinesische Schriftzeichen [JLN03].

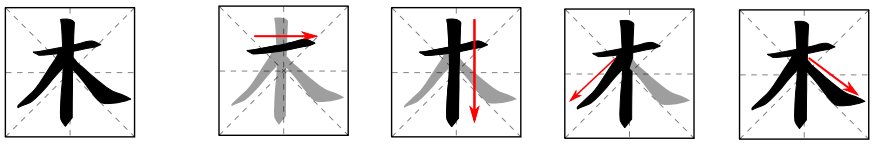

Abbildung 2.5: Beispiel für eine Schreibregel für das Wort Holz

Die Gemeinsamkeit von westlichen Sprachen und chinesischen Schriftzeichen besteht in der elementaren Zusammensetzung von Substrukturen, hier Buchstaben bzw. Radikale und einzelne Linien.

Obwohl die Anzahl der Striche im Chinesischen sehr variieren kann, so befindet sich das Wort jedoch immer in einer einheitlichen quadratischen Box. Im westlichen dagegen werden die Wörter von links nach rechts geschrieben. Die Länge eines Wortes ist dabei sehr variabel. So kann es unter anderem zu Unterschieden in der Verarbeitung und Erkennung kommen.

2.4 Schreibarten

Es gibt verschiedene Schreibarten sowohl im Westlichen als auch Chinesischen. Im Englischen wird zum Beispiel in Block- und Schreibschrift geteilt. Die chinesischen Zeichen können in regulär, flüssig und kursiv geschrieben werden. Dabei zählen flüssige und kursive Schreibweise als Schreibschrift.

Abbildung 2.6: Englisches Wort für *Bett* in Block- und Schreibschrift

Die reguläre Schrift oder Blockschrift im Chinesischen wird mit geraden Strichen gezogen [DLX07]. Die Strichanzahl und -position der Zeichen entsprechen dem Standard.

Flüssige Schreibschrift ist vergleichbar mit westlicher Schreibschrift. Es werden Buchstaben bzw. Striche mit Kurven verbunden [DLX07, LJN04].

In kursiv werden Striche schnell und in geringer Anzahl geschrieben. Hier werden Linien zusammengezogen. Hinzu kommt, dass Radikale noch vereinfacht geschrieben werden [TSW90], wodurch die Zeichen stark vom Standard abweichen. Die

Unterschiede reichen so weit, dass auch Menschen die Zeichen schwer entziffern können [LJN04].

Abbildung 2.7: Beispiele für reguläre, flüssige und kursive chinesische Schrift aus der Datenbank CASIA [CLLa]

Kapitel 3

Statistische Methode zur Zeichenerkennung

3.1 Hidden-Markov-Modelle

Dieser Teil des Grundlagen Kapitels widmet sich dem Hidden-Markov-Modell (*HMM*). Sie werden hier an einem kleinen Beispiel erklärt und formal beschrieben. Zudem bilden sie die Grundlage zur verwendeten Erkennungstechnik in dieser Arbeit. Die Aussagen, die hier getroffen werden und die Einführung der formalen Schreibweise lehnen an [Rab89, Sta04] an.

Hidden-Markov-Modelle(*HMMs*) sind statistisches Modelle, denen ein Markov-Prozess zugrunde liegt. Dabei kann der Zustand des modellierten Systems zwischen verschiedenen Zuständen mit bestimmten Wahrscheinlichkeiten wechseln. Die Besonderheit hierbei ist, dass der Prozess erster Ordnung ist. Das bedeutet, dass der nächste Zustand nur eine Abhängigkeit zum letzten hat.
Bei einem *HMM* sind diese Zustände versteckt. Sie emittieren jedoch messbare Daten, die je nach Zustand mit verschiedenen Wahrscheinlichkeiten auftauchen. Mithilfe der Beobachtungen bzw. Emissionen können Rückschlüsse auf die versteckten Zustände getroffen werden.
HMMs finden vor allem in der Signalverarbeitung, speziell auch in der Spracherkennung Nutzen.

3.2 Notation mit Beispiel

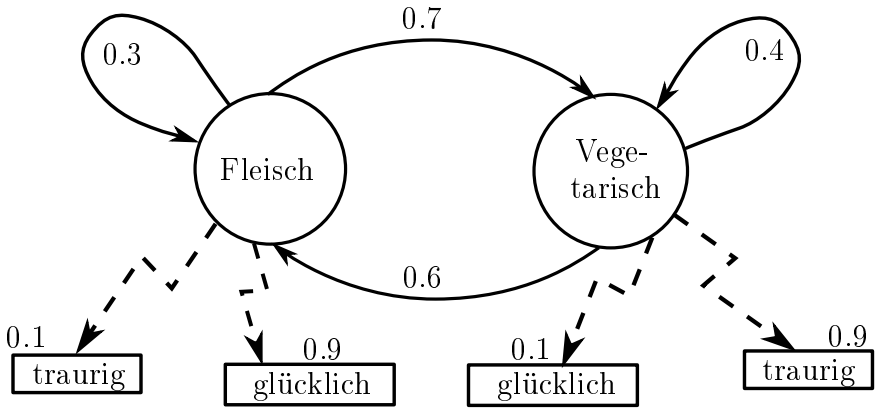

Abbildung 3.1: Visualisierung von einem Hidden Markov Modell

Angenommen an einer Uni in der Mensa gibt es zu Mittag immer nur ein Fleischgericht F oder eine vegetarische Mahlzeit V. Dies kann als Markov Prozess modelliert werden. Es gibt $N = 2$ Zustände die das Mittagsgericht einnehmen kann. Jeder eindeutige individuelle Zustand in N wird mit $S_1 \dots S_N$ gekennzeichnet. Ein Zustand zu einer bestimmten Zeit t wird mit q_t bezeichnet.

Von einem zum nächsten Tag, kann die Art des Gerichts sich verändern. Dabei gibt es bestimmte Zustandsübergangswahrscheinlichkeiten, die in einer Matrix $\boldsymbol{A}$ dargestellt werden können. Beispielsweise ist die Wahrscheinlichkeit, dass auf Fleisch noch einmal Fleisch folgt 0.3 groß.

$$\boldsymbol{A} = \begin{pmatrix} 0.3 & 0.7 \\ 0.6 & 0.4 \end{pmatrix}. \tag{3.1}$$

Die Matrix $\boldsymbol{A} = \{a_{ij}\}$ hat die Größe $N \times N$ wobei

$$a_{ij} = \mathrm{P}(q_{t+1} = S_j | q_t = S_i) \text{ mit } 1 \leq i, j \leq N. \tag{3.2}$$

a_{ij} ist der Übergang vom Zustand S_i zur Zeit t nach S_j zur Zeit $t + 1$. Hinzu kommen die statistischen Bedingungen:

$$a_{ij} \geq 0 \tag{3.3}$$

$$\sum_{j=1}^{N} a_{ij} = 1. \tag{3.4}$$

Das bedeutet, dass jede Wahrscheinlichkeit größer gleich 0 sein muss und dass eine Zeile in der Übergangsmatrix $\boldsymbol{A}$ sich zu 1 summiert.

Die Zustände des Mittagsgerichts sind jedoch versteckt. Das Beispiel wird um einen Studenten erweitert, der gerne selber kocht, aber sich nicht zwischen Fleisch und Gemüse entscheiden kann. Das soll anhand der Mensa entschieden werden. Er weiß jedoch nicht, was es zu essen gab. Anhand seines Mitbewohners kann er jedoch feststellen, was angeboten wurde. Dieser ist begeisterter Fleischesser und kommt entsprechend glücklich bzw. traurig nach Hause, je nachdem was es zu essen gab.

M ist die Anzahl der möglichen Beobachtungen. Es ist die Beobachtung, wie sich der Mitbewohner fühlt: glücklich (G) oder traurig (T). Die Menge V bezeichnet dabei die möglichen Beobachtungssymbole, im Beispiel somit (G) und (T). Jedes mögliche Symbol einer Beobachtung wird mit $v_1 \dots v_M$ gekennzeichnet.

Falls der Mitbewohner traurig ist, so ist die Wahrscheinlichkeit, dass es vegetarisch gab 0.9. Die Wahrscheinlichkeit, dass er traurig ist, obwohl es Fleisch gab ist 0.1. Die Beobachtungswahrscheinlichkeiten werden in einer Beobachtungsmatrix $\boldsymbol{B}$ zusammengefasst.

$$\boldsymbol{B} = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}. \tag{3.5}$$

Die Matrix $\boldsymbol{B} = \{b_j(k)\}$ hat die Größe $N \times M$ wobei

$$b_j(k) = \mathrm{P}(v_k \text{ zur Zeit } t | q_t = S_j) \text{ mit } 1 \leq j \leq N \tag{3.6}$$

$$1 \leq k \leq M. \tag{3.7}$$

Auch hier gilt, dass die Summe in jeder Zeile in Matrix $\boldsymbol{B}$ sich zu 1 summiert. Neben diskreten Wahrscheinlichkeiten ist es auch möglich eine Wahrscheinlichkeitsverteilung anzunehmen z.B. eine Gaußverteilung.

Bei *HMMs* gibt es einen initialen Zustand. Dieser könnte der Beginn eines Semesters darstellen. Die Wahrscheinlichkeit für Fleisch wäre gleich der für vegetarisch 0.5. Diese Wahrscheinlichkeiten werden in einer Matrix mit $\boldsymbol{\pi}$ gekennzeichnet

$$\boldsymbol{\pi} = \begin{pmatrix} 0.5 & 0.5 \end{pmatrix}. \tag{3.8}$$

Wobei $\boldsymbol{\pi} = \{\pi_i\}$ mit

$$\pi_i = \mathrm{P}(q_1 = S_i) \text{ mit } 1 \leq i \leq N. \tag{3.9}$$

Ein *HMM* benötigt die Modellparameter N und M , die möglichen Zustände S_N und Beobachtungssymbole V_M, sowie die Wahrscheinlichkeiten in den Matrizen $\boldsymbol{A}$, $\boldsymbol{B}$ und $\boldsymbol{\pi}$. Es gibt daher die Kurzschreibweise für ein *HMM* λ:

$$\lambda = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi}). \tag{3.10}$$

Ist ein gültiges *HMM* gegeben, so ist es möglich eine oder mehrere Beobachtungen zu tätigen. Eine Beobachtungssequenz O hat eine Anzahl von T Beobachtungen:

$$O = O_1 \dots O_T. \tag{3.11}$$

Jede Beobachtung O_t ist dabei ein Symbol aus der Menge von Beobachtungssymbolen V. Aus den Beobachtungen wird versucht, relevante Informationen zum eigentlichen Markov Prozess herauszufinden.

3.3 Drei Basis Probleme von Hidden-Markov-Modellen

Ist ein *HMM* gegeben, so können bestimmte stochastisch relevante Problemstellungen gelöst werden. Diese werden in drei fundamentale Probleme aufgefasst. Diese, sowie die passenden Algorithmen, werden hier kurz erklärt und benannt.

Problem 1 Gegeben sei ein *HMM* $\lambda = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$ und eine Beobachtungssequenz $O = O_1 \dots O_T$. Im ersten Problem soll herausgefunden werden, wie hoch die Wahrscheinlichkeit dieser Beobachtungssequenz O ist, wenn das gegebene Modell λ gilt. Es ist also $P(O|\lambda)$ zu berechnen.
Der dazu meist genutzte Algorithmus ist der sogenannte Forward-Backward-Algorithmus. Es wird ein sogenannter α-pass bzw. eine *forward*-Variable berechnet. Für jede Beobachtung bzw. Beobachtungssequenz wird die Wahrscheinlichkeit bis zum Zeitpunkt t rekursiv berechnet. Dies geschieht für jeden Zustand q_i zum Zeitpunkt t im Markov Prozess. Danach werden die Wahrscheinlichkeiten summiert.

Problem 2 Gegeben sei ein *HMM* $\lambda = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$ und eine Beobachtungssequenz O. Das Ziel im zweiten Problem ist es, die optimale Sequenz von Zustandsübergange im versteckten Markov Prozess zu berechnen.
Der Viterbi-Algorithmus, der auf dynamischer Programmierung basiert, wird dafür verwendet. Dabei spielt es keine Rolle wie gut ein einzelner Zustand zu einer Sequenz passt. Es wird versucht die globale optimale Zustandssequenz zu finden.

Problem 3 Gegen sei eine Beobachtungssequenz O, die Anzahl der Zustände N und die Anzahl der Beobachtungssymbole M. Es soll ein Modell $\lambda = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$ bzw. die Parameter des Modells so berechnet werden, dass $P(O|\lambda)$ maximiert wird. Es kann als Training oder Lernen des *HMM* angesehen werden.
Eine iterative Methode, das Modell zu adjustieren, ist der Baum-Welch-Algorithmus. Dabei wird von einem initialen *HMM* λ ausgegangen. Dieses ist entweder eine

gute Vermutung oder zufällig gewählt. Es erfolgt eine Schätzung mithilfe von stochastisch relevanten Variablen, welche im Vorfeld berechnet werden. Erhöht sich $P(O|\lambda)$, so werden die Variablen neu berechnet und es erfolgt eine neue Schätzung. Dies geschieht solange bis $P(O|\lambda)$ nicht mehr wächst, ein bestimmter Schwellwert erreicht worden ist oder eine bestimmte Anzahl an Iterationen vollzogen wurde.

3.4 Arten von Hidden Markov Modellen

Im Beispiel 3.2 und in Abbildung 3.1 ist ein ergodisches bzw. ein vollkommen verbundenes *HMM* zu sehen. Das bedeutet, dass jeder Zustand von jedem anderen aus in endlich vielen Schritten erreicht werden kann. Es hat sich herausgestellt, dass andere Typen manchmal besser geeignet sind als das Standardmodell. Eine solche Variante ist das Links-Rechts- bzw. das Bakis-Modell.
Dabei ist der Zustandsübergang nur zu einem Zustand mit einem höheren Index als der aktuelle möglich. Der Übergang ist nur in eine Richtung erlaubt.

$$a_{ij} = 0 \text{ mit } j < i \tag{3.12}$$

Manchmal gibt es zusätzliche Einschränkungen, beispielsweise ist ein Übergang höchstens zum Index größer als 2 möglich. Zudem gibt es nur einen initialen Zustand

$$\pi_i = \begin{cases} 0, & i \neq 1 \\ 1, & i = 1 \end{cases} \tag{3.13}$$

da die Zustandssequenz bei Index $i = 1$ anfängt und bei $i = N$ endet.

Kapitel 4

Android

4.1 Allgemein

Im Zusammenhang mit Smartphones wird der Begriff Android als Betriebssystem verstanden, welches dem Benutzer eine graphische Oberfläche zur Interaktion anbietet. Der Nutzer kann zudem selbst entscheiden welche Applikationen und welche Hardware ihm zur Verfügung stehen soll [GS10].
Ursprünglich entstand Android aus einem Zusammenschluss von Firmen durch Google, welche unter den Namen Open Handset Alliance lief. Das Ziel war und ist es, eine Open-Source Software-Plattform zu entwickeln, wodurch Integration und Wiederverwendung vereinfacht werden soll. Dadurch ist es möglich, verschiedenen Entwicklern die Zusammenarbeit an einem gemeinsamen Produkt zu geben, eigene Ideen zu verwirklichen und das System auf Bedürfnisse anzupassen. Durch die offene Zugänglichkeit trägt die Gesellschaft einen Beitrag zur Verbesserung bei, anstatt das Monopol einer einzelnen Firma zu geben [Inca].

Seit der Ankündigung des Betriebssystems und dem offiziellen Start hat sich Android sehr stark verbreitet. Für Smartphones stieg von 2009 bis 2013 der Marktanteil von Android am Endkundenabsatz von rund 4% auf ca. 80% und überholte somit auch Apples Betriebssystem iOS [Staa]. Allein in Deutschland stieg der Marktanteil von Android Januar 2012 bis Mai 2014 von 60% auf 80% [Stab].

Android wird vor allem attraktiver durch die Möglichkeiten der Individualisierung, den Gedanken als Open-Source Projekt und durch die hohe Portabilität. Letzteres ist durch die Architektur selbst gegeben und wird im kommenden Abschnitt näher erläutert.

4.2 Architektur

Aus der Entwicklersicht ist die Systemarchitektur von Android ein Schichtenmodell. Dabei existieren 4 Schichten, wie in Abbildung 4.1 zu sehen ist. Die Aussagen zur Architektur stammen aus [GS10, Dev11].

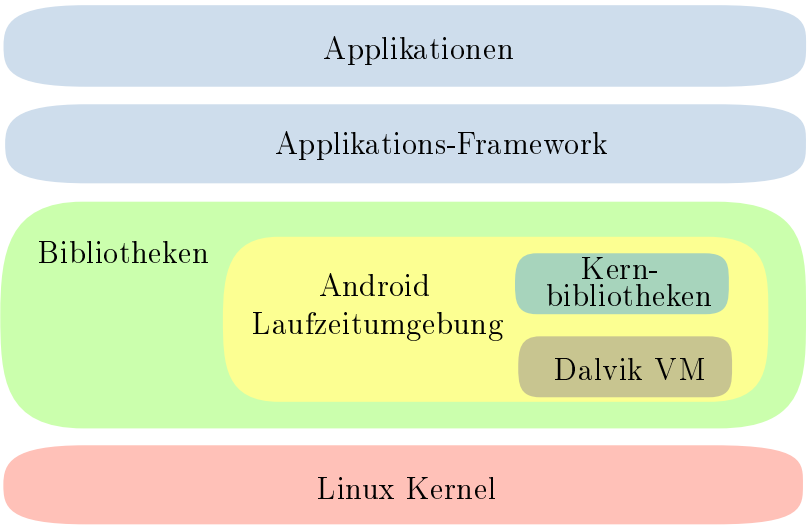

Abbildung 4.1: Die Android Schicht-Architektur [GS10]

Die oberste Schicht bilden die Applikationen bzw. Apps. Darunter zählen Basiselemente, wie etwa Kalender oder E-Mail. Entwickelte Anwendungen von anderen Organisationen oder einzelnen Personen, wie Spiele oder funktionale Apps gehören ebenfalls dazu. Diese werden in der Programmiersprache Java implementiert.
Das Applikations-Framework dient der Wiederverwendung von bereits vorhandenen Applikationen, indem ihre APIs in dieser Schicht veröffentlicht und mithilfe von Managern und Services benutzt werden. Sie erleichtert die Standardisierung der Applikationsstruktur.
Die nächste Schicht besteht einmal aus Bibliotheken, welche in C/C++ geschrieben sind und über Java Interfaces oder das Applikations-Framework genutzt werden können. Zum anderen existiert die Android Laufzeitumgebung, welche aus Basisbibliotheken besteht und die Kernelemente in Java ausmacht. Jede Applikation läuft in einer eigenen Dalvik-Virtual-Maschine, die speziell für geringen Speicheraufwand und parallele Nutzung von mehreren virtuellen Maschinen entwickelt worden ist. Sie baut auf einem Linux-Kernel auf, der die letzte Schicht in der Architektur bildet. Sie dient als Abstraktion von Software zu Hardware und macht eine hohe Flexibilität und Portierbarkeit aus.

4.3 Aktivitäten und Lebenszyklus

Im Folgenden wird der grobe Aufbau und Ablauf einer Android Applikation erläutert [Incb, Incc]. Eine Anwendung besteht im Grundlegenden aus einer oder mehreren Aktivitäten, Layouts und dem Android-Manifest. Eine Aktivität ist eine graphische Oberfläche, die dem Benutzer ermöglicht, bestimmte Aktionen auszuführen. Diese sind beispielsweise das Verfassen und Versenden von Nachrichten. Die graphische Oberfläche für eine Aktivität wird mit einem Layout definiert und kann im *XML*-Format beschrieben werden. Das Android-Manifest besitzt ebenfalls dieses Format und liefert der Applikation wichtige Informationen zum Start. Alle verwendeten Aktivitäten werden darin eingetragen und es werden Zugriffsrechte oder benötigte Bibliotheken für den späteren Gebrauch deklariert.

Es gibt Plattformen, in denen eine Anwendung nur gestartet und beendet werden kann. Dagegen ist der Ablauf einer Android Anwendung ein Lebenszyklus und durchläuft verschiedene Stadien. Dies ist in Abbildung 4.2 zu sehen. Da eine Anwendung aus mehreren Aktivitäten bestehen kann, gilt der Ablauf für jede Aktivität. Es gibt verschiedene Zustände, die den Lebenszyklus verwalten und mit Callback-Methoden korrespondieren.

Es gibt eine Hauptaktivität, die den Einstiegspunkt darstellt. Ein Beispiel wäre, die Übersichtsanzeige von Nachrichten und Kontakten. Dabei werden die Methoden *onCreate* und *onStart* aktiviert. *onResume* ist aktiv während der aktuell laufenden Aktivität. Sie befindet sich im Vordergrund und ist sichtbar.
Von diesem Startpunkt aus, können andere Aktivitäten aufgerufen werden, z.B. das Verfassen einer neuen Nachricht. Sobald eine andere Aktivität gestartet wird, rückt die vorherige in den Hintergrund und *onPause* wird ausgeführt. Die letzte Aktivität kann beendet werden, wenn andere Anwendungen mit höherer Priorität mehr Speicher oder Leistung benötigen.
Danach kann entweder *onResume* folgen, indem der Nutzer zur alten Aktivität zurückkehrt oder es folgt *onStop*. Sowohl im Zustand *onPause* als auch im Zustand *onStop* sollten größere Ressourcen, wie Netwerk- oder Datenbankverbindungen, frei geben werden. Falls die alte Aktivität fortgesetzt werden soll, so werden *onRestart* und *onStart* ausgeführt und der Lebenszyklus wird fortgesetzt. Wird die alte Aktivität jedoch nicht mehr benötigt oder dazu aufgefordert sich zu beenden, wird *onDestroy* aufgerufen. Die Aktivität oder die gesamte Anwendung wird komplett beendet.

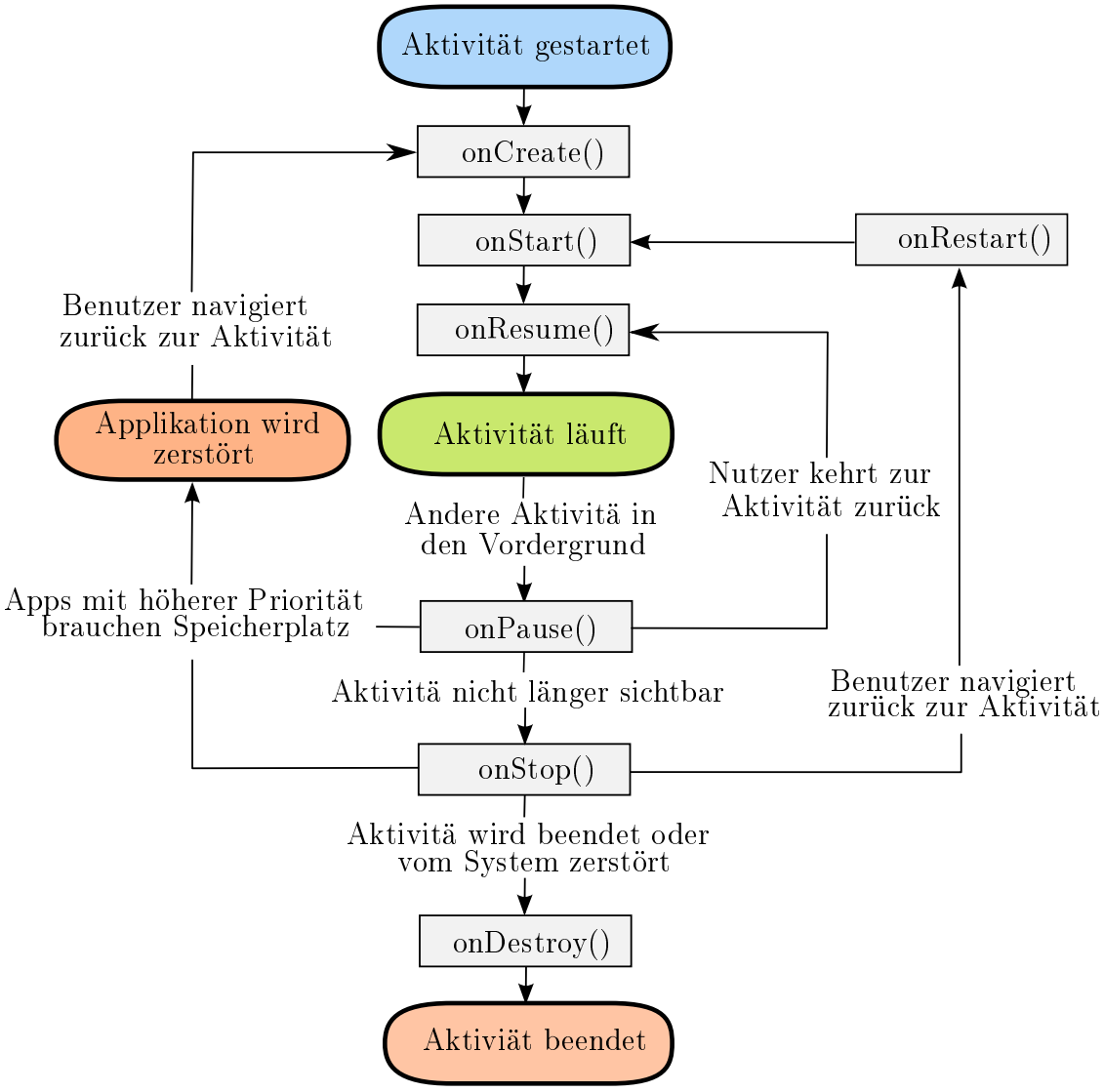

Abbildung 4.2: Der Android Lebenszyklus [Incb]

Kapitel 5

Online Handschrifterkennung

In diesem Kapitel wird über die online Handschrifterkennung im Allgemeinen gesprochen. Es wird der gängige Prozess des Erkennens erläutert sowie einige Beispiele zu jedem Schritt gegeben. Der Prozess gliedert sich in die 4 Hauptaspekte: Vorverarbeitung, Feature-Extraktion, Klassifikation und Nachverarbeitung. Abbildung 5.1 zeigt ihren Ablauf.

Abbildung 5.1: Ablauf eines Erkennungsprozesses

In der Arbeit liegt der Fokus hauptsächlich auf der online Erkennung von chinesische Zeichen bzw. Kanji. Daher werden die genannten Techniken sich besonders auf diese beziehen. Es werden jedoch teilweise Vergleiche zwischen der westlichen und chinesischen Schrifterkennung gegeben, um Unterschiede oder Schwierigkeiten festzustellen.

5.1 Allgemein

Als Input wird meist eine Menge von Punkten gegeben, die wiederum eine Linie darstellen können. Diese werden in der Vorverarbeitung so prozessiert, dass sie für den Erkennungsprozess nutzbar sind. Heraus kommen Merkmale die mit einer Datenbank verglichen werden. Das Merkmal, welches am besten mit einem gegebenen Eintrag übereinstimmt, wird meist als erkanntes Zeichen verwendet. In der Nachverarbeitung wird versucht, dieses zu verifizieren.

Online Erkennung bedeutet, dass eine sofortige Rückmeldung erscheint, sobald ein Benutzer eine Eingabe tätigt. Das heißt der Erkennungsprozess läuft bereits

während des Schreibens. Die offline Erkennung ist das Gegenteil und wartet mit der Erkennung, bis ein Zeichen vollendet ist.
Je nachdem welche Techniken und Algorithmen im Prozess beteiligt sind, ist die Erkennung zeitlich versetzt zum Schreiben. In der Regel werden 0,2 bis 2,5 chinesische Schriftzeichen pro Sekunde gezeichnet. Im Englischen sind es 1,5 bis 2,5 Zeichen pro Sekunde. Somit besteht für die online Erkennung die Anforderung, dass sie mindestens so schnell sein muss, um mit dem Schreiben Schritt zu halten [TSW90].

Das größte Problem stellt die Strichvariation in Anzahl und Reihenfolge dar [TSW90, LJN04, DLX07, ZN12]. Nativen Schreiber wird für jedes Zeichen eine bestimmte Schreibung beigebracht, dennoch kann es zu Variationen kommen. Besonders wenn keine reguläre Blockschrift verwendet wird. Bei Applikationen mit Schreibern ohne Vorkenntnisse der Sprache kann es ebenfalls zu Abänderungen kommen. Beispiel für so eine Anwendung wäre ein mobiles Wörterbuch mit aktiver Eingabe. Es gibt verschiedene Herangehensweisen, dieses Problem zu lösen. Der Trend geht zu online Features, da daraus Anzahl, Reihenfolge und Richtung der Striche filterbar sind [TSW90].

Es ist noch nennenswert, dass es zudem Benutzer abhängige und unabhängige Erkennungssysteme gibt [TSW90]. Nutzer abhängige Systeme kennzeichnen sich durch System Adaption aus. Es werden Schreiber spezifische Eingaben gegeben. Dadurch lernt das System den eigenen Schreibstil des Benutzers. Das kann zu einer besseren Erkennung führen. Das Gegenteil ist ein Benutzer unabhängiges System. Solche sind schwerer zu konstruieren, da starke Variationen zwischen Schreibern auftauchen können. Vorteil von der Unabhängigkeit ist die Nutzung ohne eine Vorkalibrierung.

5.2 Vorverarbeitung

Die Vorverarbeitung ist ein wichtiger Schritt vor der eigentlichen Erkennung eines Zeichens. Unabhängig von späteren Schritten wird immer eine Normalisierung auf den Input Daten durchgeführt. Zudem dient sie dazu, das Zeichen auf eine bestimmte Größe zu vereinheitlichen und mögliche Variationen im Muster zu verringern. Variationen können z.B durch Umwelt Einflüsse entstehen oder sind einfach Benutzer spezifische Kringel. Wichtigkeit der Vorverarbeitung wird sehr betont [JLN03, TSW90, LJN04, YYS90], da sie die Qualität der Erkennung verbessert.

Im Allgemeinen kann die Normalisierung in linear und nicht linear eingeteilt werden [LJN04]. Es wird dabei eher zu nicht linearen Verfahren tendiert [ZN12, JLN03]. Lineare Normalisierung zeichnet sich dadurch aus, dass der Input durch affine Transformationen verändert wird. Am Ende soll sich das Zeichen in einer Standard Box befinden, ähnlich wie in Abbildung 2.2.

Nicht linear Normalisierung berechnet die Punkte neu und verteilt sie bezüglich ihrer Dichte. Das bedeutet, dass dichtere Gebiete erweitert werden und dünn besiedelte Gebiete verkleinert werden. So wird die Eingabe auf eine bestimmte Größe gebracht, wobei stark besiedelte Gebiete mehr Aufmerksamkeit erhalten. Das Stichwort ist hier die Normalisierung der Liniendichte (engl. line density equalization) von [YYS90].

Da die chinesischen Schriftzeichen bereits in einer imaginären Box geschrieben werden, ist die Normalisierung einfacher als beispielsweise für westliche Schriften. Dort ist die Länge des Wortes, als auch die Höhen und Tiefen sehr variabel. Durch die Variation vom Verhältnis der Länge und Größe, kann die Normalisierung schwieriger sein. Es wird beispielsweise versucht, die imaginäre Basislinie und von dort aus die Ober- und Unterlinie zu finden. Dadurch ist es möglich, die Größe oder eventuelle Rotationen im Wort festzustellen [JLN03]. Ansonsten ist es auch üblich, eine Segmentierung durchzuführen. Dadurch reduziert sich das Erkennen auf 26 Buchstaben, was Rechenzeit und -speicher spart [TSW90].

Neben einer Normalisierung gibt es weitere optionale Vorverarbeitungsschritte, die bei beiden Spracharten verwendet werden. Meist erfolgt noch eine Daten Reduktion bzw. Kompression. Dort geht es darum, insignifikante Punkte zu löschen, etwa Punkte die alle auf einer geraden Linie sind [JLN03], z.B. durch Equidistance Sampling. Dabei wird durch Resampling die Distanz zwischen den Puntken einer Linie äquivalent approximiert [LJN04].

Weitere Schritte in der Vorverarbeitung die im allgemeinen noch verwendet werden sind Segmentierung oder Rauschen. Methoden zur Entfernung von Rauschen sind z.B. Weichzeichnen, Filterung oder Korrektur von Ausreißern [TSW90].

5.3 Feature Extraktion und Repräsentation

Ist die Vorverarbeitung mit Normalisierung und Daten Kompression abgeschlossen, folgt eine Extraktion von Features. Diese sind relevante Daten für die Erkennung. Meist werden geometrische Merkmale wie Distanzen, Winkel, lokale Richtungen oder Richtungsänderungen gesucht [JLN03]. Andere Daten wären z.B. Fourier Koeffizienten von x- und y-Koordinaten [TSW90], Gradienten [DLX07] oder Histogramm Features [LJN04]. Letzteres ist dabei eher ein offline Feature. Dabei wird ein bestimmtes Merkmal, wie die Richtung oder die Anzahl bestimmter Linien, gezählt. Noch gibt es keine Standardmenge von Features, dennoch sind die gesuchten Daten im westlichen und chinesischen Zeichen oft gleich [JLN03].

Extrahierte Features und die Features in der Datenbank müssen nicht unbedingt die gleiche Darstellung haben [LJN04]. Meist sind die Input Merkmale auf einem niedrigeren Level als die des Mustermodells eines Zeichens.

Features können weiterhin eingeteilt werden in strukturell, statistisch-strukturell und statistisch [LJN04].

In der strukturellen Repräsentation wird der Strich selbst analysiert. Einzelne Punkte oder Linien können in Segmente eingeteilt werden, woraus Sequenzen entstehen. Diese werden versucht mit anderen zu matchen. Solche simplen Darstellungen sind die niedrigste Ebene von Features.
Ein solches einzelnes Segment könnte mit einem Code versehen werden, sodass jede Art oder Richtung von Strich eine eindeutige Zuordnung besitzt. Diese Darstellung wird Strich-Code-Repräsentation genannt.
Es gibt zudem Features, die besonders auf die Beziehung zwischen den einzelnen Strichen achten. Eine solche relationale Darstellung kann als Graph realisiert werden und eignet sich gut für chinesische Zeichen, da diese einen hierarchischen Aufbau besitzen.

Die stochastisch-strutkurelle Repräsentation misst relevante Daten probabilistisch und stellt sie in einer String, Graph oder Baum Form dar. Im Allgemeinen können alle strukturellen Merkmale stochastisch beschrieben werden. Diese Features können als Sequenz weitergegeben und erkannt werden. Wichtigster Vertreter sind die *HMMs*. Diese sind in der Erkennung von westlichen Zeichen bereits stark vertreten, wie z.B. in [HBT96].

Rein stochastische Darstellung geht so vor, dass alle Daten aus der Trajektorie in ein 2D-Bild abgebildet werden. So kann es als offline Erkennung betrachtet werden und ermöglicht es, Strichvariationen nicht beachten zu müssen.

5.4 Klassifikation

Im Allgemeinen wird die Klassifikation nach [ZN12, LJN04, JLN03] in grob und fein eingeteilt. Bei der Klassifikation geht es darum, die gefundenen Features mit einer gegebenen Datenbank abzugleichen. Darin befinden sich Klassen mit Referenzmodellen zum eigentlichen Zeichen. Es wird versucht das Modell mit der geringsten Abweichung vom Input zu finden. Dieses wird dann als Ergebnis der Erkennung angesehen.
Da versucht wird, mit einer Vorlage zu vergleichen, wird vom *Template-Matching* gesprochen. Es wird auch *Nähester-Nachbar-Klassifizierung* genannt, da versucht wird den kleinsten Abstand vom Input zu einer Referenz zu finden.

Die grobe Klassifikation beschleunigt den Erkennungsprozess selbst, indem versucht wird, wahrscheinlichere Kandidaten oder Klassen von Kandidaten zu finden und unnötige nicht zu beachten. Meist werden simple Distanzmessungen verwendet, um eine kleinere Menge aus der Menge der gesamten Referenzen zu finden. Eine andere Möglichkeit ist, für jede Referenz eine Punktzahl zu vergeben und entsprechend einem Schwellwert die passenden Kandidaten wählen.

Die feine Klassifikation wird auf entsprechend gefundene Kandidaten angewendet. Es gibt nun verschiedene Art und Weisen, wie die Zeichen erkannt werden können. Eine Herangehensweise ist über dynamische Programmierung, auch DP-Matching genannt. Es ist eines der Basiswerkzeuge der japanischen Kanji Erkennung. Dabei wird die ideale Korrespondenz zwischen gegebenen Punkten bzw. anderen Features gesucht. Als passendes Kriterium kann die Levenshtein-Distanz [Lev66] genommen werden. Sind zwei Zeichensequenzen gegeben, so wird die eine in die andere durch Operationen wie Einfügen, Ersetzen oder Löschen umgewandelt. Diese können gezählt oder gewichtet werden. So ist es möglich den Unterschied herauszufinden. Eine Anwendung in der online Schrifterkennung ist beispielsweise die Authentifizierung von Unterschriften [SVD04].

Es gibt weitere Matching Verfahren, die mithilfe von Beziehungen der Striche untereinander versuchen, eine Korrespondenz zu finden. Dies kann als Problem der automatischen Positionierung von Labels angesehen werden. Eine Möglichkeit der Lösung wäre eine heuristische Suche mithilfe des A*-Algorithmus.

Da es probabilistische Features gibt, existieren auch probabilistisches Matching Verfahren. *HMMs* wurden bereits als wichtige Vertreter für die Repräsentation von Features erwähnt. Die Aufgabe hier ist es, die eingehende Sequenz von Merkmalen in die wahrscheinlichste Sequenz von Zustandsübergängen zu übersetzen. Dies entspricht dem zweiten Problem von *HMMs* wie in 3.3 erwähnt. Es wird also für jedes Zeichen ein solches *HMM* verwendet. Ein anderer Ansatz ist, Radikale oder bestimmte Stricharten als *HMM* zu modellieren. Diese Herangehensweise wird in 6 vertieft.

Zum Schluss gibt es statistische Klassifikation. Diese finden besonders Verwendung, wenn mehrere Features in einem Vektor zusammengefasst werden. Dabei werden intelligente Klassifikatoren benutzt. Kandidatenklassen werden als Untervektorraum dargestellt und werden als multivariate Gaußverteilung angenommen. Sie sind sehr genau, haben jedoch hohe Speicher- und Rechnerkosten.
Eine Kombination aus mehreren Klassifikatoren ist natürlich auch möglich.

5.5 Nachverarbeitung

Die Nachverarbeitung wird auch linguistische Verarbeitung genannt [JLN03] [LJN04]. Man kann diesen letzten Schritt als Verifizierung des erkannten Zeichens sehen [LJN04]. Hier wird versucht, die Klassifikation noch zu verbessern. Oft kommen mehrere Zeichen in Frage. Der linguistische Kontext kann helfen die optimalen Kandidaten zu finden. Dabei wird ein Wörterbuch generiert, welches ein bestimmtes Vokabular enthält. Man kann dies als Graphen oder Baum mit N-Grammen darstellen. N-Gramme kommen aus der Sprachverarbeitung und stellen N-Tupel von syntaktischen Einheiten dar, wie z.B. Buchstaben oder Wörter. Es

gibt Kombinationen die wahrscheinlicher auftreten als andere und dadurch eine höhere Priorität erhalten. Bi- und Tri-Gramme werden bereits in der westlichen Schrifterkennung angewendet. Ein Buchstabe oder Gruppe von Buchstaben ist dabei ein Knoten im Graphen. Mit Viterbi oder anderen Such-Algorithmen kann ein erkanntes Wort nochmals linguistisch verifiziert werden.

Die Graph-Darstellung kann auch für chinesische Zeichen in Frage kommen. Dabei wird eine Menge von Schriftzeichen als Graph dargestellt mit möglichen Strichen und Radikalen als Knoten.

Kapitel 6

Wahl des Verfahrens - Substroke-Ansatz

6.1 Auswahl des Verfahrens

In Kapitel 2 wurden die Grundlagen zur chinesischen Schrift erklärt und Unterschiede zur westlichen dargestellt. Beispielsweise besteht die Anzahl der zu erkennenden Zeichen bei dem einen aus 26 Buchstaben und bei dem anderen aus 5000 Zeichen. Auch die Länge bzw. Größe der Zeichen brauchen verschiedene Ansätze zur Schrifterkennung.
Andererseits gibt es auch Gemeinsamkeiten, wie die Zusammensetzung aus elementaren Zeichen. Für Ersparnisse kann in westlichen Sprachen das Wort in Buchstaben segmentiert werden und diese wiederum in bestimmte Linien- oder Kurvensegmente. Das Gleiche ist auf chinesische Zeichen anwendbar. Die kleinsten Elemente hier sind Linien. Es gibt neuere Erkennungsverfahren die diesen Gedankengang mit Hilfe von mehreren *HMMs* verfolgen.
Ein solches Verfahren nennt sich Teilstrich-Ansatz (orig. engl. Substroke Approach) von Nakai et. al. [NASS01]. Da sich der englische Begriff für Teilstrich auch im Deutschen etabliert hat, wird im Folgenden vom Substroke-Ansatz bzw. von Substroke-*HMM* gesprochen. Das Verfahren selbst ist von japanischen Forschern für Kanji entwickelt worden. Da Kanji und chinesische Schriftzeichen essenziell gleich sind, kommt diese Technik auch für letzteres infrage.

In Kapitel 5 wurden verschiedene Verfahren für die Erkennung vorgestellt. Der Substroke-Ansatz ist eine Kombination daraus. Er zeichnet sich durch Speichereffizienz aus, da in dem Verfahren nur 25 Basis Teilstriche angenommen werden. Ein weiterer Aspekt des Verfahrens sind *HMMs*, die für die Substokes trainiert werden. Noch gibt es nur wenige Verfahren, die das Erkennen von chinesischen Schriftzeichen mit *HMMs* wagen [JLN03]. Somit ist diese Idee noch neu und stellt

eine andere Herangehensweise dar.
Zudem ist die Speicherung eines einzelnen Zeichens über ein sogenanntes Wörterbuch realisiert. Dort ist für jedes Symbol eine Sequenz von Substrokes hinterlegt. Dadurch ist es möglich das Problem der Strichvariation in Zahl und Reihenfolge anzugehen. Weiterhin kann der Rechenaufwand gering gehalten werden, beispielsweise in der Suche der wahrscheinlichsten Substroke-Sequenz durch effiziente Darstellung und Suche im Wörterbuch.
Durch diese Kombination und Möglichkeiten wurde der Substroke-Ansatz ausgewählt. Dieses Verfahren stellt ein sehr kompaktes System dar mit dem Wunsch nach geringeren Speicher- und Rechenaufwand. Dies sind ideale Voraussetzungen, um eine Implementation auf einem mobilen Endgerät zu versuchen.

In der Arbeit selbst werden nur Zeichen in regulärer Blockschrift betrachtet und es werden vorerst für den Prototypen nur traditionelle Strichreihenfolgen verwendet. Diese Einschränkung wird durch spätere Kapitel genauer erklärt.
Im Folgenden soll zunächst das Verfahren selbst erläutert werden. Danach werden Vor- und Nachteile deutlich gemacht. Später folgt die Interpretation und praktische Umsetzung in einen Prototypen. Die vorgestellten Ideen stammen aus [NASS01, NSSS02, TIM+02, NSS03].

6.2 Das Verfahren

6.2.1 Feature-Extraktion

Die benötigten Input-Features sind Geschwindigkeiten bzw. Verlagerungen von Stift- oder Fingerposition. Im Grundlegenden werden somit nur die Koordinaten benötigt. Diese werden während des Zeichnens von der Benutzeroberfläche des Tablet oder Handys aufgenommen. Dadurch ist eine Vorverarbeitung des rohen Inputs nicht mehr nötig. Mithilfe der erfassten Koordinaten können nun Geschwindigkeits- und Verlagerungsvektoren berechnet werden.

Eine Stiftposition ist ein Vektor mit x- und y-Koordinaten (x, y). Dann ist $\begin{pmatrix} dx \\ dy \end{pmatrix}$ ein Feature-Vektor, der den Unterschied von der letzten zur neuen Eingabeposition darstellt. Während ein Strich gezogen wird, werden konstant Positionen der Eingabe aufgezeichnet. Das Eingabegerät befindet sich somit auf der Zeichenfläche, daher wird von Pen-Down gesprochen. Der Vektor $\begin{pmatrix} dx \\ dy \end{pmatrix}$ bezeichnet bei Pen-Down die Geschwindigkeit jeder Stiftposition während jeder gezeichneten Linie.

Eine Verlagerung stellt den Wechsel der Stiftposition vom Absetzen des letzten Striches zum Ansetzen des neuen dar. In Abbildung 6.1 ist die rote Linie ein

Beispiel für einen Verlagerungsvektor. Der Stift oder Finger befindet sich während des Wechsels nicht auf der Benutzeroberfläche, daher wird von Pen-Up gesprochen.

Abbildung 6.1: Ein Beispiel einer Pen-Up Verlagerung

6.2.2 Hidden-Markov-Modelle für Substrokes

Jedes Zeichen besteht aus Basiselementen. Die kleinste Einheit sind dabei Striche. Bei diesem Verfahren wird angenommen, dass es 25 Stricharten gibt, aus denen jedes Kanji bzw. chinesische Zeichen besteht. Diese Stricharten können in 8 Richtungsklassen eingeteilt werden. Abbildung 6.2 zeigt die Klassen für gezogene Pen-Down Linienmodelle. Sie werden mit Buchstaben gekennzeichnet, wobei eingeteilt wird in kurze (a-h) und lange (A-H) Striche.

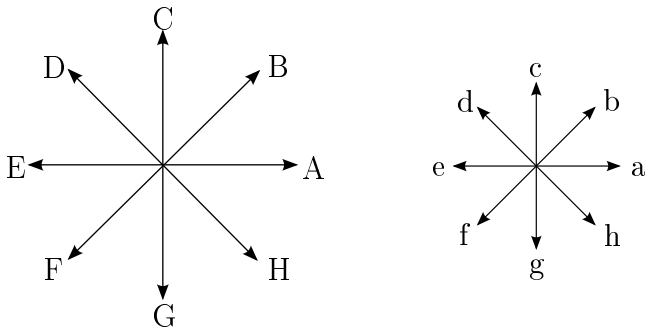

Abbildung 6.2: Die Richtungsklassen a-h und A-H von Pen-Down Substrokes für lange und kurze Striche [NASS01]

Die Pen-Up Modelle haben ebenfalls 8 Richtungsklassen, welche mit Zahlen 1-8 dargestellt werden. Dazu kommt noch eine weitere Richtungsklasse 0. Diese bezeichnet das Absetzen und das erneute Ansetzen an genau der gleichen Stelle.

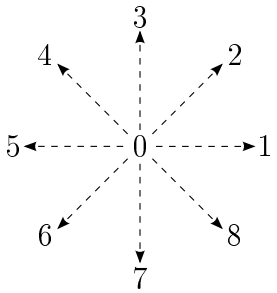

Abbildung 6.3: Die Richtungsklassen 0-8 von Pen-Up Substrokes [NASS01]

Diese verschiedene Pen-Down und Pen-Up Modelle werden als *HMM* dargestellt. Die Anzahl der Zustände im Pen-Down Modelle ist $N = 3$. Jeder Zustand stellt dabei eine unterschiedlich große Geschwindigkeit eines Striches dar. Zudem sollen diese Links-Rechts-*HMMs* sein. Die Pen-Up Modelle haben nur einen Zustand $N = 1$, der eine Verlagerung ausgibt. Da es nur einen Output geben kann, gibt es keine weiteren Zustandsübergänge.

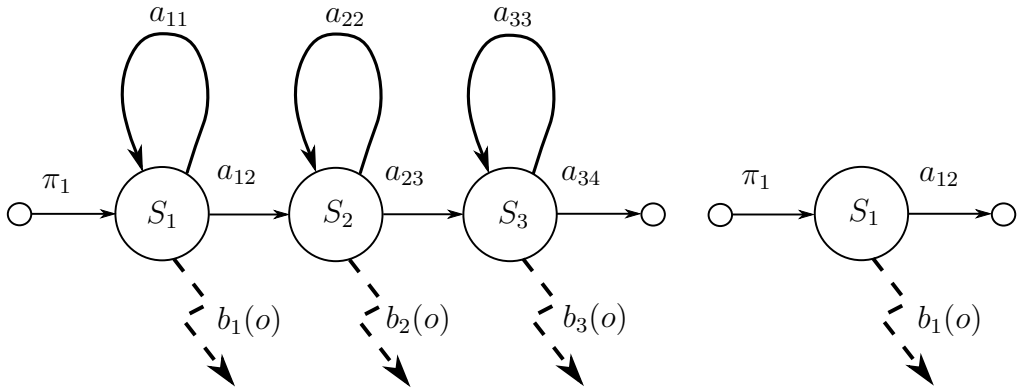

Abbildung 6.4: Substroke *HMMs* für Pen-Down (links) und Pen-Up (rechts)

Für jede Substroke Klasse k gilt das *HMM*

$$\lambda^k = (\boldsymbol{A}^k, \boldsymbol{B}^k, \pi^k). \tag{6.1}$$

Wobei $\boldsymbol{A}^k = \{a_{ij}^k\}$ die Wahrscheinlichkeitsverteilung der Zustandsübergänge von S_i nach S_j, $\boldsymbol{B}^k = \{b_j^k(\boldsymbol{v})\}$ die Wahrscheinlichkeitsverteilung der Beobachtungssymbole $\boldsymbol{v}$ zum Zustand S_j und $\pi^k = \{\pi_i^k\}$ die initiale Zustandswahrscheinlichkeit angibt.

Die Wahrscheinlichkeit einer Beobachtung ist hierbei nicht diskret, sondern wird mit einer multivariate Gaußverteilung angenommen mit

$$b_j(\boldsymbol{v}) = \frac{1}{\sqrt{(2\pi)^n \mid \boldsymbol{\Sigma}_j \mid}} \exp\left(-\frac{1}{2}(\boldsymbol{v} - \boldsymbol{\mu}_j)^t \boldsymbol{\Sigma}_j^{-1} (\boldsymbol{v} - \boldsymbol{\mu}_j) \right). \tag{6.2}$$

Die Substroke Modelle werden vor der Verwendung mit entsprechenden Daten trainiert. Dies entspricht dem Problem 3 der *HMMs* wie in 3.3 erwähnt.

6.2.3 Hierarchisches Wörterbuch

Die Substroke Modelle alleine reichen nicht, um ein Zeichen zu erkennen, da die Definition für Schriftzeichen fehlt. Diese Definitionen werden in einem Wörterbuch

realisiert. Da die chinesische Schrift hierarchisch aufgebaut ist, kann dieses für das Wörterbuch nützlich gemacht werden.

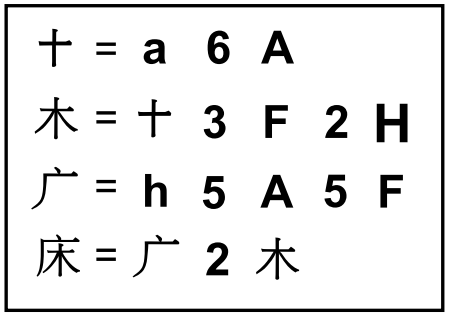

Abbildung 6.5: Möglicher Wörterbucheintrag für 4 Zeichen

In Abbildung 6.5 ist ein solches Nachschlagewerk abgebildet. Auf der linken Seite sind jeweils die Zeichen selbst. Die Regel in der zweiten Zeile bedeutet, dass das Zeichen für *Holz* 木 sich aus einem anderen Zeichen und vier weiteren Substrokes zusammen setzen lässt. Um Redundanzen zu vermeiden, kann eine Sequenz von Substrokes durch ein bereits definiertes Zeichen ersetzt werden.

Noch gibt es keinen automatischen Algorithmus, der ein solches Wörterbuch definiert. Es wird manuell erstellt. Laut [NASS01] wird dies eines der zukünftigen Forschungsthemen sein.

6.2.4 Klassifikation

Der Input ist auf einer niedrigeren Darstellungsstufe als die Referenzmodelle, wie in 5.3 erwähnt. Als Features zum Erkennen werden Geschwindigkeits- bzw. Verlagerungsvektoren eingegeben. Diese werden mit Referenzmodellen als *HMMs* verglichen.

Ein sogenannter Decoder übernimmt den Prozess des Erkennens. Für jedes Schriftzeichenmodell aus dem Wörterbuch gibt es eine Sequenz von Substrokes. Die passenden *HMMs* für jede Substrokesequenz werden vom Decoder konkateniert. Danach wird berechnet, wie wahrscheinlich der Input auf die jeweilige Sequenz zutrifft. Es wird ein idealer Pfad gesucht, der am besten zur Eingabe passt. Diejenige mit der höchsten Wahrscheinlichkeit wird als erkanntes Zeichen ausgegeben.
Um nicht jedes mal eine Sequenz aus dem Wörterbuch zu generieren bzw. alle Einträge komplett zu durchlaufen, wird dieses als Graph dargestellt. Dadurch können effiziente Suchalgorithmen angewendet werden.
In der zugrundeliegenden Arbeit werden jedoch keine genaueren Angaben zu der Suche oder der eigentlichen Pfadfindung gemacht.

6.3 Vor- und Nachteile des Verfahrens

Wie zu Beginn des Kapitels erwähnt, eignet sich das Verfahren nur für reguläre Blockschrift. Die Erkennungsrate bei Schreibschrift liegt viel niedriger als die der Systeme, welche ein *HMM* für ein ganzes Zeichen verwenden [TIM+02]. Bei kursiver Handschrift werden einzelne Linien verbunden. Dadurch kann es sehr schnell zu einer falschen Zuordnung der Substrokes kommen.

Es gibt Zeichen mit der gleichen Sequenz von Substrokes. Das Wort *Brunnen* 井 und das Wort *öffnen* 开 haben beide die Sequenz *(A 6 A 4 F 2 G)*.
Ein menschliches Auge erkennt, dass die Schriftzeichen sich unterscheiden, da die Linienpositionen jeweils verschieden sind. Bei der Erkennung nur mit Substrokes ist es klar, dass es zu solchen Mehrdeutigkeiten kommen kann.
Durch die manuelle Erstellung des hierarchischen Wörterbuchs kann es auch direkt zu einer falschen Erkennung eines Zeichens führen. Beispielsweise wird ein kurzer Strich als lang interpretiert und falsch in das Wörterbuch eingetragen. Genauso kann es schnell zu einer Verwechslung der Verlagerungsklassen kommen.

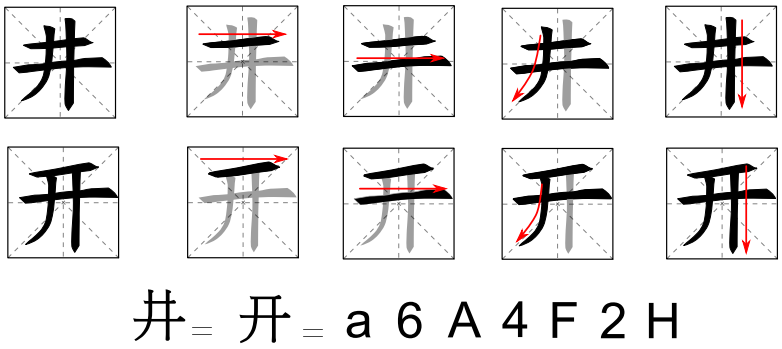

Abbildung 6.6: Das Wort für *Brunnen* und *öffnen* besitzen die gleiche Sequenz von Substrokes. Die gezogenen Arten der Linien sind gleich, nur die Position ist anders.

Ein weiterer Nachteil könnten Poly-Striche darstellen. Bei diesen muss die Richtungsänderung bzw. der Beginn des nächsten dazugehörigen Striches erkannt werden.

Ein Problem bei der chinesischen Schrifterkennung ist die Reihenfolge der Striche. Jedes Zeichen hat zwar eine gegebene Schreibregel, aber es kann zu einer falschen Schreibung kommen. Mithilfe des Wörterbuchs kann dieses Problem umgangen werden. Für einen Zeicheneintrag werden mehrere Sequenzen mit unterschiedlicher Substroke Reihenfolge definiert. Das absehbare Problem hierbei sind die Kombinationsmöglichkeiten. Besteht ein Zeichen aus n Strichen, so gibt es auch $n!$ mögliche Zusammensetzungen. In Kapitel 9.5 wird versucht, diese Schwierigkeit zu überwinden.

Für ein Zeichen sind jedoch nur die 25 Substroke-*HMMs* und die dazugehörige Definition im Wörterbuch nötig. Dieser Eintrag kann um ein neues Zeichen jederzeit erweitert werden. Es muss kein neues Schriftzeichen trainiert werden, da die Basis Elemente bereits existieren.
Ein weiterer Vorteil ist die Speichereffizienz. Da es nur ein Wörterbuch und 25 *HMMs* gibt, kann der benötigte Platz gering gehalten werden. Die verwendete Speicherkapazität unterbietet die von Systemen, die *HMMs* von ganzen Schriftzeichen verwenden [NASS01]. Durch einen kleineren Platzbedarf eignet sich das Substroke Verfahren besonders für kleine Geräte.
Wird eine schnelle und effiziente Suche für das Wörterbuch implementiert, so ist die Laufzeit der Erkennung sehr kurz.

Kapitel 7

Training der Substroke Hidden-Markov-Modelle

7.1 Datenbank für chinesische Zeichen

Vor der Verwendung der Substroke-*HMMs* müssen diese trainiert werden. Die Trainingsdaten werden aus der Datenbank CASIA [CLLa] vom Institut für Automatisierung der chinesischen Wissenschaftsakademie entnommen. Diese und andere Datenbanken für online und offline Daten werden in [LYWW11] erklärt und dargestellt. Für die Erhebung der benötigten Daten wurde ein Anoto-Stift von Anoto AB verwendet. Es ist ein Stift mit eingebauter Kamera, welcher auf Papier mit bestimmten Muster schreibt und Druck, Winkel oder Position des Stiftes bestimmen kann. Die Daten können an einen Rechner geschickt werden für weitere Verarbeitungen. [AB].

Für das Training der Substroke-*HMMs* wurde speziell CASIA-OLHWDB1.1 [CLLb] ausgewählt, da diese einzelne Schriftzeichen beinhaltet. Diese steht für Forschungszwecke von Vorverarbeitung, Feature-Extraktion und Klassifizierung frei zur Verfügung. Zudem stehen die reinen Koordinaten ohne Vorverarbeitung in den Dateien. Dies entspricht dem Input von Android.

Der Datensatz ist eingeteilt in Trainingsdaten mit 240 Schreibern und Testdaten mit 60 Schreibern. Die Mengen der Probanden ist disjunkt. Es wurden die 3755 Zeichen auf Level 1 von Zeichensatz GB2312-80 aufgenommen. Die Samples wurden sequenziell für jeden Schreiber in eine Datei gespeichert. Diese liegen in binärer Form vor und ihr Aufbau ist in [CLLc] genauer erläutert. Sie sind so aufgebaut, dass ein repräsentativer Code aus dem GB2312-80 und die Anzahl der gezeichneten Striche übersichtlich extrahiert werden kann.

Die Daten werden von nativen Schreiber erstellt. Im Gegensatz zu nicht-nativen Schreiber tendieren diese dazu, unsauberer zu schreiben, obwohl sie in regulärer

Blockschrift schreiben. Es kann also zu Unterschieden in der Linienführung kommen.
In der chinesischen Schrift gibt es zudem einige der Richtungen des Substroke-Ansatzes nicht oder sie treten sehr selten auf. Diese falschen oder raren Striche sind in der Datenbank nicht vorhanden. Somit kann nur für die richtigen Striche ein *HMM* trainiert und validiert werden.

7.2 Parsen der Trainings- und Testdaten

Das Parsen kann bereits als Teil der praktischen Umsetzung gezählt werden, da für den eigentlichen Prototypen trainierte *HMMs* gebraucht werden. Die Implementation des Parsens wird hier kurz erläutert.

Zunächst werden alle Samples, die zu einem Zeichen gehörend, gemeinsam abgespeichert. Durch diese Sortierung kann auf alle Samples eines Zeichens direkt zugegriffen werden.
Für die Substroke-*HMMs* werden bestimmte Striche benötigt. In der Datenbank wird nicht zwischen Block- oder Schreibschrift unterschieden und es gibt keinen Vermerk welche Stricharten in welchem Zeichen stehen. Es wird eine Heuristik angewendet, um die passenden Substrokes finden. Die Annahme ist, dass die Blockschrift genau die richtige Anzahl von Strichen und die richtige Strichreihenfolge besitzt. Da in jedem Sample der passende GB2312-80 Code und die Anzahl der verwendeten Striche abgespeichert ist, wurde die Filterung vereinfacht. Die Recherche nach der richtigen Anzahl von Strichen geschieht jedoch manuell. Dies war ein langwieriger Prozess, da zunächst passende Zeichen und deren Schreibweise nachgeschlagen werden musste.
Durch diese Heuristik passiert es, dass für die Striche unterschiedlich viele Samples für Training und Test entstehen. Denn verschiedene Schreiber können ein Zeichen regulär und das andere wiederum kursiv schreiben. Natürlich kann es auch zu Unterschieden in der Reihenfolge kommen, da auch native Schreiber fehlerhaft schreiben.

Für das Filtern wurde eine eigene Datenstruktur angelegt, welche später in der Android Applikation ebenfalls verwendet wird. Es handelt sich um eine Klasse *Character*, welche die entsprechenden Striche und Koordinaten speichert. Das UML-Diagramm in Abbildung 7.1 zeig diese Struktur.

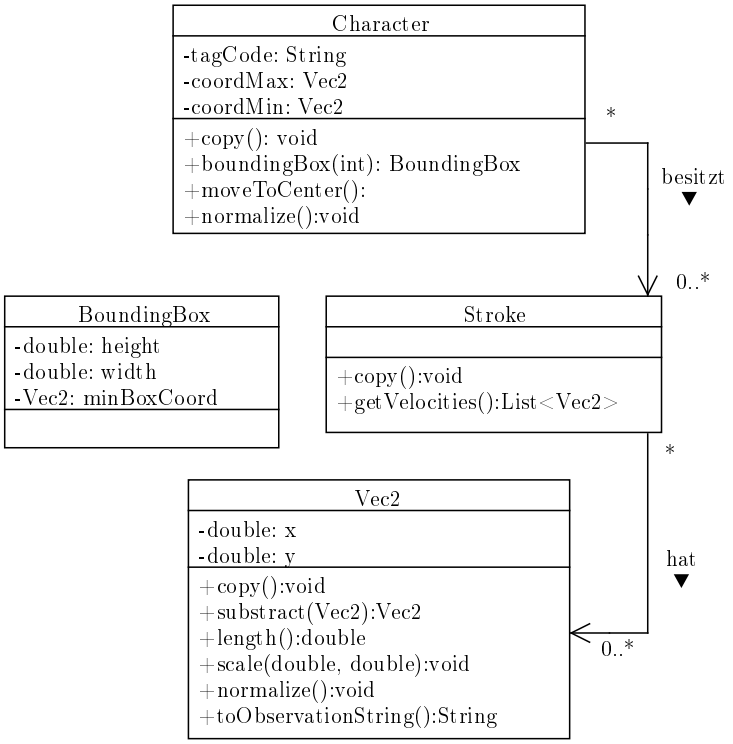

Abbildung 7.1: UML-Diagramm für eine Character-Klasse

Alle gezogenen Linien eines jeden gefilterten Samples werden sequenziell abgespeichert. So können einzelne Striche mit einem Index erreicht und die benötigten Vektoren herausgefunden werden.
Für die Pen-Down Modelle wird jeweils der Strich an der spezifische Stelle i genommen. Die Geschwindigkeitsvektoren werden jeweils paarweise zwischen zwei aufeinander folgenden x- und y-Koordinatenpaaren gebildet.
Für die Pen-Up Modelle muss der Strich i angegeben werden, der nach dem Absetzen und wieder Aufsetzen des Stiftes die benötigte Verlagerung besitzt. Es wird der Vektor gebildet zwischen der letzten Koordinate des letzten Striches und der ersten Koordinate des nächsten Striches.

7.3 Training der Hidden-Markov-Modelle

7.3.1 Bibliothek für Hidden-Markov-Modelle

Da das Hauptaugenmerk nicht auf *HMMs* liegt, wurde eine existierende Bibliothek verwendet. Diese nennt sich *jahmm* von Jean-Marc François und wurde in Java implementiert [Fra]. Veröffentlicht ist sie unter der BSD-Lizenz [Ini].
Sie enthält *HMM* spezifische Algorithmen, wie beispielsweise den Baum-Welch

Lernalgorithmus oder den Forward-Backward Algorithmus. Zudem ermöglicht sie es, für ein *HMM* diskrete Wahrscheinlichkeiten oder eine Wahrscheinlichkeitsverteilung für Beobachtungen zu verwenden. Somit ist es auch möglich, Vektoren als einzelne Beobachtung zu benutzen.

Da *jahmm* in Java implementiert wurde und Android ebenfalls Java verwendet, fiel die Wahl sehr schnell auf diese Bibliothek.

7.3.2 Ablauf des Trainings

Mithilfe der *jahmm* Bibliothek werden die Substroke-*HMMs* trainiert. Dabei werden die gefilterten Samples in einem für *jahmm* passendes Format eingelesen. Diese werden als Beobachtungssequenz für ein Baum-Welch-Training verwendet. Es muss jedoch ein initiales *HMM* gegeben sein. Durch verschiedene Testläufe hat sich ergeben, dass das initale *HMM* zwar die Einschränkungen eines Links-Rechts-Modells nicht genau einhält, aber die besseren Ergebnisse erzielt. Dieses wird wie folgt initialisiert:

$$\boldsymbol{A} = \begin{pmatrix} 0.4 & 0.3 & 0.3 \\ 0.0 & 0.5 & 0.5 \\ 0.1 & 0.3 & 0.6 \end{pmatrix} \text{ und } \boldsymbol{\pi} = \begin{pmatrix} 0.8 & 0.1 & 0.1 \end{pmatrix}$$

Die Wahrscheinlichkeitsverteilung der Emissionen von möglichen Beobachtungen wird für jeden Zustand auf eine multivariate Gaußverteilung gesetzt.

Nach dem Training wird eine Textdatei erstellt, welche trainierte *HMM* darstellt. Sie wird für die Applikation verwendet und kann von der *jahmm* Bibliothek gelesen werden.

Ein *HMM* ist abhängig von den verwendeten Samples. Je nachdem wie gut oder schlecht Probanden für das Training sind, so fällt die Erkennung gut oder schlecht aus. Daher werden mehrere *HMMs* für ein Substroke trainiert. Diese werden dann validiert. Die Beschreibung dafür ist in Abschnitt 9.2 zu finden.

Kapitel 8

Implementation des prototypischen Erkennungssystems

8.1 Vorbereitung und Einschränkungen

Die trainierten Substroke-*HMMs* aus Kapitel 7 werden für den Prototypen verwendet. Dieser funktioniert bisher nur mit bestimmten Substrokes, da es auch nur bestimmte Samples gab. Zudem wird nur die richtige Strichreihenfolge beachtet, da es sich in der Datenbank nur um native Schreiber handelt. Es kann natürlich sein, dass vereinzelt falsche Stricharten und -reihenfolgen in der Datenbank existieren. Jedoch gibt es keine Möglichkeit schnell an diese Daten zu kommen.

Der Substroke-Ansatz verläuft so, dass für jeden Wörterbucheintrag alle dazugehörigen *HMMs* konkateniert werden. Für die eingegebene Strichsequenz von Vektoren wird der wahrscheinlichste Eintrag ausgewählt, indem ausgerechnet wird, wie wahrscheinlich die einzelnen Zustände der *HMMs* sind. Aufgrund fehlender Bibliotheken, welche solch eine Suche bzw. Matching effizient gestalten können, kann der eigentliche Substroke-Ansatz nicht ganz vervollständigt werden. Da der Ansatz nur Andeutungen von verwendeten Algorithmen vorschlägt, wird eine eigene Herangehensweise gewählt.
Dabei werden mehrere Sequenzen von Substrokes erkannt, welche anschließend im Wörterbuch nachgeschlagen werden. Die Idee dabei ist, dass mehrere Kandidatensequenzen die Möglichkeit der Erkennung erhöhen. Mithilfe von Beam-Search werden diese Sequenzen erstellt. Diese Herangehensweise kann als Greedy zählen und wird in den nächsten Abschnitten genauer erläutert.

Zu Beginn wurde ein Server-Client-Modell für die Erkennung entwickelt. Der Client stellte dabei ein Android-Smartphone dar. Für jeden gezogenen Strich auf dem Gerät wurde eine JSON-Objekt mit den jeweiligen Koordinaten auf einen Server übertragen. Es wurde getestet, wie schnell ein einzelner Strich erkannt wer-

den konnte. Nach mehreren Durchläufen wurde klar, dass ein Prototyp für die Erkennung auf dem Android-Gerät selbst lauffähig war.

8.2 Aufbau und Oberfläche des Systems

Für den Prototypen wurde eine möglichst einfache Struktur verwendet. Die Anwendung besteht aus einer einzelnen Aktivität. Die Oberfläche besitzt zunächst die nötigsten GUI-Elemente. Ein Screenshot der Applikation ist in Abbildung 8.1 zu sehen.

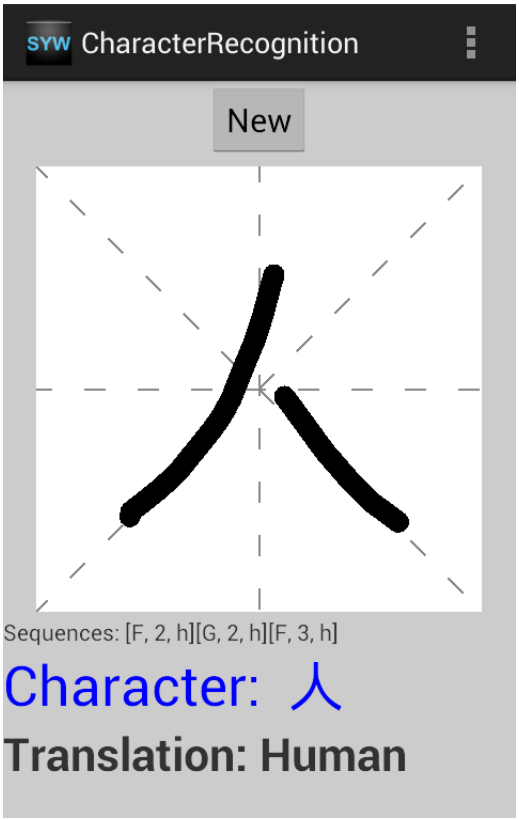

Abbildung 8.1: Screenshot des Prototypen

Ein *Button* ermöglicht das Löschen des eingegebenen Zeichens, sodass ein neues gezeichnet werden kann. Darunter befindet sich die quadratische Zeichenfläche mit einem Muster, wie es in der chinesischen Schrift üblich ist. Dies erleichtert dem Benutzer die Eingabe in eine Box zu schreiben. Unter der Zeichenfläche befinden sich drei *TextViews*. Eines stellt die erkannten Sequenzen dar und die anderen beiden zeigen jeweils das erkannte Zeichen bzw. dessen Übersetzung. Die Übersetzung eines Zeichens erfolgt, wenn auf dieses geklickt wird.

Der Ablauf der Erkennung ist im Aktivitätsdiagramm in Abbildung 8.2 dargestellt. Beim Start der Applikation werden nötige Informationen und Objekte geladen. Dazu gehört das Wörterbuch für die Zeichendefinitionen, die trainierten Substroke-*HMMs* und die graphische Oberfläche. Ist die Applikation sichtbar, wird auf eine Eingabe gewartet.

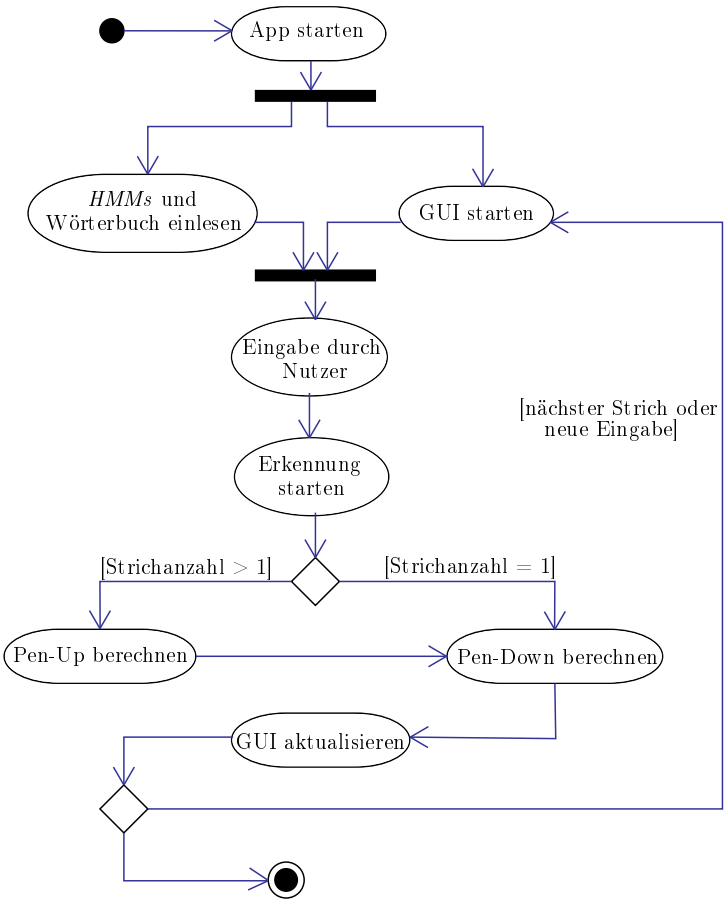

Abbildung 8.2: Aktivitätsdiagramm des Prototypen zur Erkennung von chinesischen Schriftzeichen

Erfolgt eine Eingabe auf der Zeichenfläche, indem Striche gezogen werden, so werden Koordinaten aufgenommen. Sobald ein Strich beendet wurde, startet der Erkennungsprozess. Dabei werden Berechnungen mithilfe der Android-Klasse *AsyncTask* im Hintergrund vollzogen, ohne das User-Interface zu blockieren.
Der Erkennungsprozess verläuft komplett in *AsyncTask*. Es werden die aufgenommen Koordinaten übergeben und prozessiert. Die Erkennung wird in den nächsten Abschnitten genauer erläutert.

8.3 Feature Extraktion

Wird der Erkennungsprozess gestartet, werden zunächst die Feature-Vektoren berechnet. Der übergebende Parameter ist ein *Character*-Objekt, wie bereits in Kapitel 7.2 beschrieben. Da die Erkennung nach jeder Linie startet, muss nur jeweils die letzte Verlagerung bzw. Geschwindigkeiten berechnet werden. Die Ergebnisse nach jedem Linienzug werden in der *State*-Klasse gespeichert.

Die Vektoren werden jeweils nach folgender Formel berechnet

$$\vec{\boldsymbol{v}} = \overrightarrow{AB} = \begin{pmatrix} x_B \\ y_B \end{pmatrix} - \begin{pmatrix} x_A \\ y_A \end{pmatrix} \text{ mit } A \text{ und } B \text{ als gegebene Punkte.} \tag{8.1}$$

Für jeden gezogenen Strich werden die Geschwindigkeitsvektoren von aufeinander folgenden Koordinaten berechnet. Für die Verlagerung werden jeweils die letzte Koordinate des letzten Striches und die erste Koordinate des folgenden Striches verwendet. Es wird eine Abfrage ausgeführt, ob es bereits möglich ist, eine Verlagerung zu berechnen. Dies ist der Fall, wenn es mehr als einen Strich gibt.

8.4 Klassifikation

8.4.1 Erkennung einer Substrokesequenz

Bei der Klassifikation werden n mögliche Sequenzen als Kandidaten gebildet. Für jeden möglichen Pen-Down oder Pen-Up Strich des Benutzers werden die Wahrscheinlichkeiten jedes Substroke-*HMMs* berechnet. Dies entspricht dem Problem 1 der *HMMs* aus Kapitel 3.3. Die *jahmm* Bibliothek verwendet den Forward-Backward-Algorithmus.
Die Pen-Down Substrokes sind noch einmal in lang und kurz unterteilt. Um diese zu unterscheiden wurde ein Schwellwert verwendet. Dieser ist ein Prozentsatz von der Diagonalen der Zeichenfläche. Diese wird zuvor bei der Initialisierung des User-Interfaces berechnet und in der *State*-Klasse abgelegt.

Die Kandidaten für n mögliche Sequenzen werden mithilfe eines Beam-Search berechnet. Für jede Linie oder Verlagerung wird berechnet, welche Substroke-*HMMs* welche Wahrscheinlichkeiten liefern. Dadurch entsteht ein Entscheidungsbaum. Der Ansatz geht nun so vor, dass die n besten Substrokes gewählt, welche die höchste Wahrscheinlichkeit liefern. Dadurch entstehen n mögliche Sequenzen. Für jede der n möglichen Kandidatensequenzen werden die Wahrscheinlichkeiten der berechneten Substrokes multipliziert. Danach wird getestet, welcher aktuelle Substroke die Wahrscheinlichkeit der Kandidatensequenz maximiert. Die n Sequenzen mit der höchsten Wahrscheinlichkeit werden dann ausgewählt und ersetzen die alten.

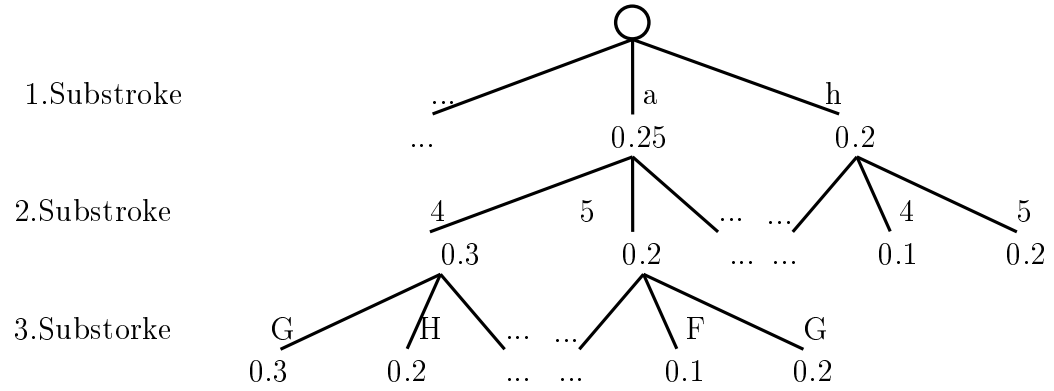

Abbildung 8.3: Visualisierung von Beam-Search bei der Berechnung von Kandidaten von Substrokesequenzen

Durch diesen Greedy-Ansatz kann es jedoch sein, dass nicht das globale Optimum erreicht wird. Da für den Moment das beste Ergebnis berechnet wird, ist dieses nur ein lokales Maximum. Greedy-Ansätz sind dafür in der Regel schnell und brauchen wenig Speicherplatz, weswegen dieser auch ausgewählt wurde.

8.4.2 Aufbau und Suche im Wörterbuch

Die Kandidatensequenzen werden in einem Wörterbuch nachgeschlagen. Dieses wurde als eigene Klasse *Dictionary* implementiert, welche durch folgendes UML-Diagramm beschrieben wird:

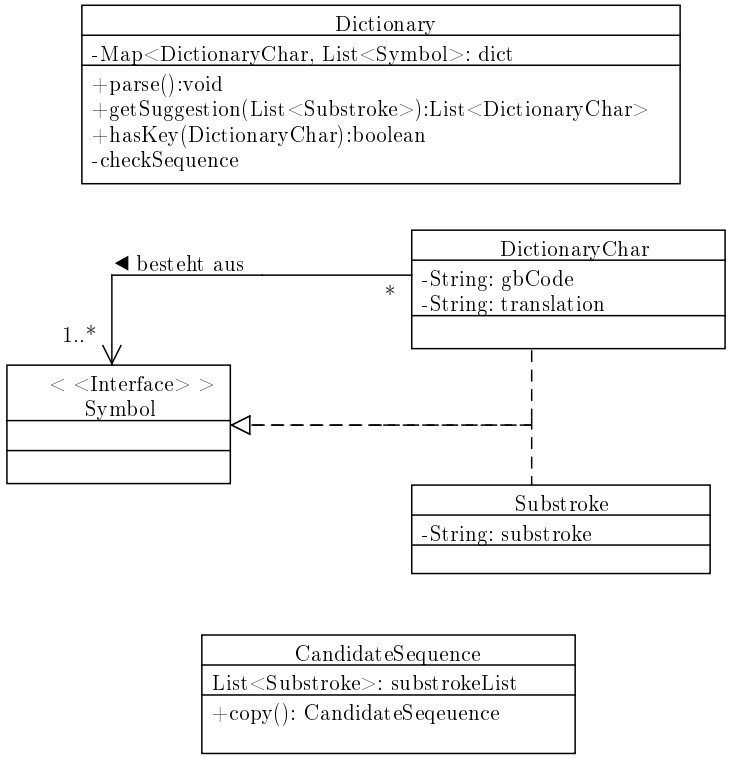

Abbildung 8.4: UML-Klassendiagramm der Klasse Dictionary

Das Wörterbuch liegt als Textdatei vor und wird zum Start der App eingelesen. Dieses wird manuell erstellt und ist hierarchisch Aufgebaut. Dabei ist für jedes Schriftzeichen *DictChar* eine Sequenz von *Symbolen* möglich. Ein *Symbol* kann dabei ein anderes Schriftzeichen oder ein *Substroke* sein. Da es sich um einen Prototypen handelt, sind in der Implementation nur 27 Schriftzeichen eingetragen.

Die Suche nach einem passenden Wörterbucheintrag für die Liste von Kandidatensequenzen ist einfach gehalten. Für jeden Kandidat wird Substroke für Substroke abgeglichen, ob diese vorhanden sind. Bisher ist somit nur ein exakter Abgleich möglich. Da es zusätzlich auch Zeichen gibt, welche die gleiche Strichanzahl und -art haben, wird das Wörterbuch komplett durchlaufen. Dadurch werden mehrere Vorschläge einer Erkennung möglich, falls es mehrere Zeichen mit der gleichen Definition gibt.

Ist ein Schriftzeichen erkannt, so wird der Nutzer über das User-Interface benachrichtigt. Dort werden die *TextViews* entsprechend dem Zeichen und dessen Übersetzung gesetzt.

Kapitel 9

Ergebnisse

9.1 Allgemein

In diesem Kapitel werden Ergebnisse der Implementation aus 8 vorgestellt. Diese werden im Anschluss ausgewertet. Da die Ergebnisse im Vergleich zu [NASS01] eine hohe Diskrepanz aufweisen, wird versucht diese zu erklären und weitere Verbesserungen vorzuschlagen.

9.2 Validierung

Für die Validierung wurden sowohl einzelne Substrokes als auch ganze Zeichen verwendet. Diese Validierung geschieht auf einem Rechner. Bei den Substrokes bzw. den ganzen Zeichen sind die jeweiligen Test-Daten aus dem CASIA Datensatz CASIA-OLHWDB1.1, wie in 7.1 erwähnt, gewählt worden. In 7.2 wurde gezeigt, wie die Zeichen und Striche extrahiert wurden.

Des Weiteren wurde eine Zeitmessung durchgeführt. Getestet wurde auf einem Android-Smartphone. Dieses ist ein Alcatel One Touch M'Pop mit der Version 5020D von TCL Communication. Es hat die Android Version Jelly Bean 4.1. Der eingebaute Prozessor hat 1 GHz und besitzt einen 512 MB großen RAM [Incd].

9.2.1 Erkennungsrate eines Substrokes

Die Validierung jeweils einer Substroke-Art verläuft mit unterschiedlich vielen Samples, da bei der Extraktion bereits unregelmäßig viele gefunden worden.
Jedes Substroke-*HMM* wird mit sieben verschieden großen Mengen von Samples trainiert. Dadurch entstehen für jede Substroke-Art sieben *HMMs*. Für die langen Pen-Down Substrokes sind beispielsweise von 200 bis 1400 in 200er Schritten

Trainings-Daten verwendet worden. Nach dem Training werden extrahierte Test-Daten dagegen validiert. Es wird zu sehen sein, mit wie vielen Trainings-Daten ein *HMM* eine hohe Erkennungsrate erhält. Dieser Vorgang nennt sich auch Kreuzvalidierung. Dabei wurden die Pen-Up Substrokes, die langen Pen-Down und die kurzen Pen-Down Substrokes in diesen drei Gruppen validiert.

In den folgenden Tabellen 9.1, 9.2 und 9.3 werden die einzelnen Erkennungsraten der jeweils trainierten Substroke-*HMMs* angezeigt. In den beiden Diagrammen 9.4 sind die Substroke Modelle 0 und b gesondert dargestellt, da diese eine sehr geringe Anzahl an Samples haben.

Die Tabellen sind zeilenweise zu lesen. Dabei steht nach jedem Substroke die Anzahl der Test-Daten, gegen die validiert wurde. Danach folgen die Erkennungsraten für die *HMMs*, welche mit einer unterschiedlichen Anzahl an Trainings-Daten trainiert worden sind. Beispielsweise hat der Substroke A mit 200 Trainings-Samples beim Validieren gegen 2038 Test-Samples eine Erkennungsrate von 94%.

		Anzahl Trainingssamples						
Substroke	Teststriche	200	400	600	800	1000	1200	1400
A	2038	0.94	0.94	0.88	0.89	0.89	0.90	0.90
F	1060	0.74	0.74	0.71	0.71	0.72	0.75	0.73
G	812	0.76	0.79	0.83	0.80	0.78	0.78	0.79
H	762	0.77	0.83	0.76	0.81	0.82	0.81	0.81

Tabelle 9.1: Erkennungsraten der langen Pen-Down Substrokes

		Anzahl Trainingssamples						
Substroke	Teststriche	25	50	75	100	125	150	175
a	718	0.8	0.5	0.46	0.57	0.58	0.57	0.70
f	220	0.42	0.51	0.35	0.55	0.56	0.54	0.58
g	96	0.40	0.40	0.27	0.44	0.56	0.52	0.60
h	178	0.53	0.56	0.60	0.72	0.71	0.70	0.70

Tabelle 9.2: Erkennungsraten der kurzen Pen-Down Substrokes

		Anzahl Trainingssamples						
Substroke	Teststriche	100	200	300	400	500	600	700
2	540	0.52	0.57	0.57	0.59	0.63	0.64	0.65
3	438	0.43	0.52	0.72	0.74	0.73	0.74	0.74
4	590	0.47	0.51	0.56	0.61	0.62	0.61	0.61
5	652	0.83	0.85	0.87	0.87	0.88	0.88	0.88
6	822	0.32	0.55	0.62	0.64	0.66	0.66	0.67

Tabelle 9.3: Erkennungsraten der Pen-Up Substrokes

		Anzahl Trainingssamples	
Substroke	Teststriche	4	
0	4	1.0	
		Anzahl Trainingssamples	
Substroke	Teststriche	25	50
b	30	0.67	0.80

Tabelle 9.4: Erkennungsraten des Pen-Up Substrokes 0 und des Pen-Down Substrokes b

In der folgenden Abbildung 9.1 sind die besten Erkennungsraten für jeden Substroke abgebildet.

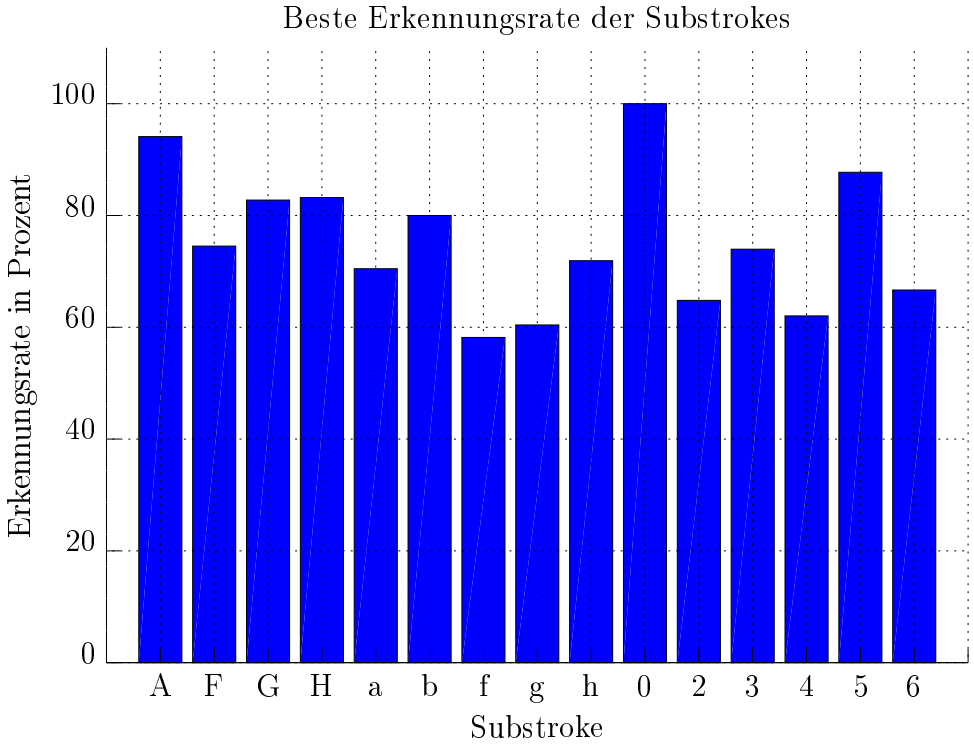

Abbildung 9.1: Beste Erkennungsrate für jeden trainierten Substroke

Es ist zu sehen, dass die Erkennungsraten sich bei ca. 70% einpendeln.

Einige Substrokes sind nicht validiert worden, da wie in 7.1 erwähnt einige Stricharten nicht in der Datenbank vorhanden sind.

9.2.2 Erkennungsrate eines Zeichens

Für die Validierung ganzer Zeichen wurden die *HMMs* aus 9.2.1 ausgewählt, welche die besten Erkennungsraten aufweisen. Es wurden 28 Zeichen zum Testen ausgewählt. Davon konnten insgesamt 1438 Samples extrahiert werden.
Jedes Zeichen wurde in eine Strich- bzw. Koordinatensequenz aufgeteilt. Danach wurde für jeden Pen-Down bzw. Pen-Up Substroke der Erkennungsprozess eingeleitet. Die Funktionsweise ist dabei wie in der Implementation auf Android in 8.

Gesamtanzahl der Zeichen	1438
erkannte Zeichen	512

Tabelle 9.5: Angabe, wie viele ganze Zeichen erkannt worden sind

In der Tabelle 9.5 ist zu sehen, dass von insgesamt 1438 Zeichen nur 512 erkannt worden sind. Dies entspricht etwa 35%.

9.3 Zeitmessung für die Erkennung

Wie bereits in 5.1 erwähnt, werden im Allgemeinen 0.2 bis 2.5 chinesische Schriftzeichen geschrieben [TSW90]. Die Erwartung an den Prototypen, wie viele Zeichen pro Sekunde erkennen muss, liegt somit in diesem Bereich.
Zunächst wurde die Zeit für die Erkennung eines Striches gemessen, da für jeden Substroke der komplette Erkennungsprozess gestartet wird. Dabei wurden 100 Samples von verschiedenen Strichen auf dem Smartphone eingelesen und die Zeit für einen Durchlauf der Erkennung gemessen. In Tabelle 9.6 ist zu sehen, dass die Erkennungszeit im Durchschnitt 0.005 s beträgt.

	einzelner Pen-Down Substroke	einzelnes Schriftzeichen
Anzahl der Samples	100	136
mittlere Zeit in s	0.005	0.076

Tabelle 9.6: Zeit für die Erkennung eines Pen-Down Substrokes bzw. eines Schriftzeichens

Des Weiteren wurde für ganze Zeichen ebenfalls die Zeit gemessen. Dabei sind Samples von den Wörtern Zwei 二 , Erde 土 und Bett 床 ausgewählt worden, da diese eine unterschiedliche Strichanzahl besitzen. Insgesamt waren es 136 Samples. Diese wurden im Durchschnitt in 0.076 s erkannt.

9.4 Bewertung der Ergebnisse

Die Erkennungsrate eines ganzen Zeichens liegt nur bei ca. 35%. Dieses Ergebnis war zu erwarten, da die Erkennungsrate eines einzelnen Substrokes bereits bei ca. 70% lag. Dieser Zusammenhang entsteht, da durch Beam-Search die Wahrscheinlichkeiten der einzelnen Substrokes multipliziert und gegeneinander abgewägt werden.
Nakai et. al. [NASS01] berichten von einer Erkennungsrate von 95%. Diese ist bereits gleichauf mit *HMMs* für ganze Zeichen. Dieser Unterschied von 60 Prozentpunkten liegt möglicherweise an verschiedenen Stellen.

Zunächst wurde ein abgewandelter Ansatz bzgl. des Erkennungsprozesses gewählt, da eine genaue Gliederung durch [NASS01] nicht gegeben war. Dadurch

ist der selbst gewählte Ansatz verschieden von dem vorgeschlagenen durch Nakai et.al. Beispielsweise ist die eigene Greedy-Suche nach der besten Sequenz nur beschränkt auf ein lokales Maximum. Dadurch ist die Suche im Wörterbuch nach Schriftzeichen bereits eingeschränkt.
Weiterhin kann die Heuristik für die Erkennung von langen bzw. kurzen Pen-Down Modellen zu Diskrepanzen führen.

Eine weitere Fehlerquelle könnte das Training und das vorangegangene Extrahieren von Samples sein. Das Sammeln der Trainings- und Testdaten ist eingeschränkt, da mit einer Heuristik passende Striche und Reihenfolgen nur angenommen werden konnten. Zudem ist die Handauslese sehr zeitaufwändig. Das anschließende Training ist abhängig von den verwendeten Daten. Falls unpassende oder falsche Samples extrahiert worden sind, so spiegelt sich der Fehler in den Substroke-*HMMs* wieder. Gleichzeitig kann der Schreibstil von einzelnen Schreibern aus der Datenbank, das Training stark beeinflussen. Wie in 7.1 erwähnt, können starke Variationen auch in der Blockschrift auftauchen.
Möglicherweise kam es unter anderem zu einer Überanpassung. Sind zu viele Daten gewählt worden, können irrelevante Formen bzw. Variablen in das Training eingeflossen sein.

Das beschriebene Training und Testen soll eine Unabhängigkeit vom Schreiber darstellen, da verschiedene Samples von verschiedenen Subjekten stammt. Möglicherweise würde ein besseres Ergebnis bei Schreiberabhänigkeit entstehen. Für die gegebene Datenbank müssten jedoch per Hand passende Zeichen und vor allem ausreichend Samples gefiltert, um diese verwenden zu können.

Des Weiteren könnten verschiedene Substrokes falsch kategorisiert worden sein. Beispielsweise gibt es senkrechte Linien, die einen Haken am Ende aufweisen. Der eigentliche Substroke-Ansatz könnte evtl. durch solche Feinheiten erweitert werden. Mögliche Verbesserungen der Fehlerquellen oder Erweiterungen von eigenen und vom Substroke-Ansatz werden im nächsten Abschnitt 9.5 weiter ausgeführt.

Was die Geschwindigkeit der Erkennung betrifft, ist sie zufriedenstellend. Es ist zu sehen, dass die Anforderung 2,5 Zeichen pro Sekunde zu erkennen erfüllt werden kann. Auch wenn es ein simpler Ansatz für den Prototypen ist.

9.5 Mögliche Verbesserungen und Erweiterungen

9.5.1 Training

Wie bereits angemerkt, sind die verwendeten Substroke-*HMMs* vom Training abhängig. Mit schlechten oder wenigen Werten verändert sich auch die Erkennungsrate. Es könnte versucht werden bessere Datensätze zu finden oder zu erstellen. Dadurch könnten klarere und vor allen Dingen mehr Samples gesammelt werden,

was die Erkennungsrate steigen lässt. Besonders geeignet wäre eine Datenbank mit sauberer Blockschrift.

Für die Entwicklung einer Anwendung mit dem Substrok-Ansatz sollte zudem beachtet werden, ob diese für native oder nicht-native Schreiber ist. Je nachdem werden vor allem die einzelnen Striche anders geschrieben. Daher sollte darauf geachtet werden, dass gelernte Daten von nativen Schreibern nicht in Applikationen für nicht-native Schreiber gegeben werden und umgekehrt.

Für das Training selbst wäre ein aktives Lernen eine weitere mögliche Herangehensweise. Dabei wird für jeden Input nach einem Label gefragt, welches ein Trainer für jeden Strich angeben muss. Somit würde jede Linie in einen Substroke eingeordnet werden. Diese manuelle Gestaltung ist jedoch teuer.

Eine andere mögliche Erweiterung wäre, weitere Features hinzufügen. Beispielsweise wäre die Druckempfindlichkeit der Zeichenoberfläche zu nutzen [NSSS02].

9.5.2 Kalibrierung

Ein Schreiber unabhängiges System, wie der Prototyp oder von Nakai et.al. führt zu einer geringeren Erkennungsrate. Es kann immer passieren, dass ein Schreiber von der Norm abweicht. Die Erkennungsrate bei Nakai et.al. sinkt auf ca. 79% wenn unabhängige Schreiber aus einem anderen Datensatz stammen. Der Unterschied ist relativ gering, jedoch unerwünscht.
Es könnte in Betracht gezogen werden, ein Schreiber abhängiges System zu modellieren. Eine Nutzerkalibrierung vor der ersten Verwendung der Applikation könnte charakteristische Merkmale des Schreibers erfassen und in ein seperates Training einfließen lassen.

9.5.3 Sequenzerkennung

Die Erkennung einer eingegebenen Strichsequenz könnte durch Trigramme erweitert werden. In Kapitel 5.5 wurden die N-Gramme bereits in der Nachverarbeitung vorgestellt. Anstatt eines Greedy-Ansatzes könnten N-Tupel von syntaktisch zusammengehörenden Substrokes verbunden werden. Jedes Tupel besteht aus einer Sequenz von Strichen und besitzt eine Wahrscheinlichkeit. Die Substrokes im Tupel stehen somit in einem Kontext im Zusammenhang. Ein solcher Ansatz wird bereits in [TIM+02] diskutiert.
Dabei werden die kontextabhängigen Modelle trainiert. Hier würden die Möglichen Kombinationen des Kontextes sehr hoch werden. Der Ansatz geht nun weiter so vor, dass gleiche Elemente von Tupel verbunden werden und dadurch ein sogenanntes Hidden-Markov-Netzwerk entsteht.
Der Erkennungsprozess für eine Eingabesequenz würde dann mit den Kontextabhängigen *HMMs* ablaufen.

9.5.4 Wörterbuch

Der Substroke-Ansatz eignet sich gut, um auch Zeichen in einer falschen Strichreihenfolge zu erkennen. Das Problem dabei sind jedoch die in Anzahl der infrage kommenden Kombinationen. Bei n Strichen ergeben sich $n!$ Zusammensetzungen. Ein möglicher Ansatz ist es, ein Wörterbuch zu trainieren [NSS03]. Dabei werden aus verschiedenen Datensätzen mit unterschiedlicher Strichordnung genau diese Unordnungen herausgefiltert. Es werden Wahrscheinlichkeiten verteilt, welche Zeichen wie oft vorkommen. Danach werden die n wahrscheinlichsten falschen Strichordnungen verwendet.
Das Wörterbuch mit den verschiedenen Kombinationen kann als gerichteter azyklischer Word-Graph (engl. *DAWG* directed acyclic word graph) dargestellt werden. Dabei werden in dieser Datenstruktur gleiche Präfixe und Suffixe zusammengefügt. Dadurch kann der nötige Speicherplatz verringert werden.

Der Ansatz kann genauso gut für eine Datenbank mit nicht-nativen Schreibern genutzt werden um falsche Strichordnungen zu erkennen.

9.5.5 Suche

Bisher arbeitet der Prototyp mit einer Eins zu Eins Suche im Wörterbuch nach der passenden Sequenz. Dadurch können Kandidaten verloren gehen, die nur eine kleine Abweichung besitzen. Es könnte versucht werden die Levenshtein-Distanz anzuwenden, welche in Kapitel 5.4 bereits erläutert wurde. Verschiedene Kandidaten bekommen dabei ein Gewicht zugeordnet, das den Unterschied zur eigentlichen erkannten Sequenz zeigt. Danach würden die n Einträge mit den besten Gewichte im Wörterbuch ausgewählt.

Eine andere Möglichkeit, die ähnlich funktioniert, wäre eine Hamming-Distanz. Diese würde bei zwei gleich langen Sequenzen nachsehen, an welchen Stellen Unterschiede existieren. Die Anzahl der Fehler ist somit die Güte, wie viel eine Kandidatensequenz von der erkannten Sequenz abweicht.

9.5.6 Ansatz

Eine andere Möglichkeit der Erweiterung liegt im Substroke-Ansatz selbst. Es wäre denkbar für bestimmte Linien z.B. Poly-Linien weitere Modelle hinzuzufügen. Aus dem Ansatz geht nicht klar hervor, wie solche Striche gehandhabt werden. Durch die Konkatenation von *HMMs* im Decoder, können Poly-Striche, die erkennbar aus Substrokes bestehen einer Linienart zugeordnet werden. Beispielsweise besteht ⅂ eindeutig aus Substrokes *a* und *g*. Dennoch können bestimmte Poly-Striche nicht in eine Kategorie passen, wie an der Abbildung 2.3 zu sehen war. Das letzte Beispiel  , kann die einzelnen Richtungen nicht in die Substrokes G oder A

einteilen. Da der senkrechte Strich einen Haken besitzt und der waagerechte Strich gewellt ist.

Kapitel 10

Zusammenfassung und Ausblick

10.1 Zusammenfassung

In der Arbeit geht es darum, ein prototypisches System auf Android zu implementieren, welches chinesische Schriftzeichen online erkennen kann. Dabei wurde ein Überblick über den Forschungsstand der online Handschrifterkennung gegeben. Insbesondere wurde dabei auf chinesische und westliche Schrift im Vergleich eingegangen. Gleichzeitig wurde auf Android und die chinesische Schrift selbst eingegangen, um Grundlagen zu schaffen.

Für die Implementation wurde eine aktuelle Idee verwendet, der Substroke-Ansatz. Dieser wurde erklärt und die mathematischen Hintergründe, insbesondere Hidden-Markov-Modelle, dargelegt. Die Annahme ist dabei, dass jedes Schriftzeichen sich aus 25 Stricharten zusammensetzen lässt. Das spart Speicherplatz, führt zu einem schnellen Verfahren und kann hohe Erkennungsraten liefern.
Dieser Ansatz wurde untersucht und in einer prototypischen Anwendung für Android praktisch umgesetzt. Dabei wurden verschiedene Modifikationen vorgenommen aufgrund fehlende Informationen des Substroke-Ansatzes. Das System wurde mit einer entsprechenden Datenbank trainiert und getestet. Dabei sind starke Diskrepanzen zum originalen Werk festgestellt worden. Diese beziehen sich auf die Erkennungsrate. Was die Geschwindigkeit einer Erkennung angeht, so unterbietet sie die Anforderung an ein mobiles System. Dadurch ist es möglich mehr Rechenleistung auszunutzen, wenn nötig. Dies ist vor allem ein Vorteil für die möglichen Erweiterungen oder Verbesserungen für die Applikation. Beispiele dafür sind bessere Distanz-Metriken zu verwenden, eine andere Datenbank für das Training zu beziehen oder die 25 Substroke-Modelle zu erweitern. Kombinationen von einzelnen Techniken können getestet werden.
Für den Substroke-Ansatz gibt es ebenfalls fehlende oder fehlerhafte Ideen und es wurden Vorschläge gegebene für eine Verbesserung. Der Ansatz selbst ist nur

geeignet für reguläre Blockschrift. Je nachdem, welchem Verwendungszweck die Applikation dient, kann dies ein Vorteil sein. Nicht-native Schreiber tendieren dazu, ein Zeichen Linie für Linie abzuzeichnen. Dadurch entsteht zwangsweise eine Blockschrift und kann somit ausgenutzt werden. Natürlich muss dabei beachtet werden, dass zeichnende und schreibende Striche sich Unterscheiden in beispielsweise Schnelligkeit.

Da es sich um einen Prototypen handelt, können in Zukunft weitere Untersuchungen in diese Richtung stattfinden. Im Großen und Ganzen eignet sich der gezeigte Ansatz auf einem mobilen Endgerät.

10.2 Ausblick

Die gezeigten Verbesserungen und Erweiterungen beziehen sich besonders auf den Erkennungsprozess des Prototypen beziehungsweise des Substroke-Ansatzes selbst. Diese Verfeinerungen werden bereits teilweise im aktuellen Forschungsstand diskutiert. Die prototypische Implementation auf Android kann um diese ausgebaut werden, um eine höhere Trefferquote zu erzielen.
Besonderes Augenmerk wird in Zukunft auf die Strichvariationen eines Zeichens gelegt werden, da dieses noch eines der Hauptprobleme in chinesischer Schrifterkennung darstellt. Ein weiterer interessanter Aspekt ist, ob eine Nutzerkalibrierung zu besseren Ergebnis führen würde.

Der Trend wandert mehr zu kleinen und handlichen Geräten, dadurch lohnt es sich, die Forschung für Erkennungssysteme in diese Richtung weiter zu lenken. Die Arbeit hat gezeigt, dass es möglich ist, Erkennungsalgorithmen auf diesen Geräten laufen zu lassen.

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[AB] AB, Anoto: *Anoto, How it works.* http://www.anoto.com/products/how-it-works/. – Eintrag vom 04.07.2014

[CLLa] C.-L. Liu, D.-H. Wang Q.-F. W. F. Yin Y. F. Yin: *CASIA Online and Offline Chinese Handwriting Databases.* http://www.nlpr.ia.ac.cn/databases/handwriting/Home.html. – Eintrag vom 20.12.2013

[CLLb] C.-L. Liu, D.-H. Wang Q.-F. W. F. Yin Y. F. Yin: *CASIA Online and Offline Chinese Handwriting Databases - Download Feature Data.* http://www.nlpr.ia.ac.cn/databases/handwriting/Download.html. – Eintrag vom 20.12.2013

[CLLc] C.-L. Liu, D.-H. Wang Q.-F. W. F. Yin Y. F. Yin: *CASIA Online and Offline Chinese Handwriting Databases - Online Database.* http://www.nlpr.ia.ac.cn/databases/handwriting/Online_database.html. – Eintrag vom 20.12.2013

[Dev11] Developers, Android: *What is Android.* 2011

[DLX07] Dai, Ruwei ; Liu, Chenglin ; Xiao, Baihua: Chinese character recognition: history, status and prospects. In: *Frontiers of Computer Science in China* 1 (2007), Nr. 2, 126-136. http://dx.doi.org/10.1007/s11704-007-0012-5. – DOI 10.1007/s11704–007–0012–5. – ISSN 1673–7350

[Fra] Francois, Jean-Marc: *jahmm - An implementation of Hidden Markov Models in Java.* https://code.google.com/p/jahmm/. – Eintrag vom 03.03.2014

[GS10] Gandhewar, Nisarg ; Sheikh, Rahila: Google Android: An emerging software platform for mobile devices. In: *International Journal on Computer Science and Engineering* 1 (2010), Nr. 1, S. 12–17

[HBT96] Hu, Jianying ; Brown, Michael K. ; Turin, William: HMM based online handwriting recognition. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 18 (1996), Nr. 10, S. 1039–1045

[Inca] Inc., Google: *Android SDK*. `http://source.android.com/index.html`. – Eintrag vom 14.07.2014

[Incb] Inc., Google: *Android SDK*. `http://developer.android.com/guide/components/activities.html`. – Eintrag vom 29.06.2014

[Incc] Inc., Google: *Android SDK*. `http://developer.android.com/training/basics/activity-lifecycle/starting.html`. – Eintrag vom 29.06.2014

[Incd] Inc., Google: *Android SDK*. `http://www.alcatelonetouch.com/global-en/products/smartphones/one_touch_mpop.html`. – Eintrag vom 17.07.2014

[Ini] Initiative, Open S.: *The BSD 3-Clause License*. `http://opensource.org/licenses/BSD-3-Clause`. – Eintrag vom 03.03.2014

[JLN03] Jaeger, S. ; Liu, C.-L. ; Nakagawa, M.: The state of the art in Japanese online handwriting recognition compared to techniques in western handwriting recognition. In: *Document Analysis and Recognition* 6 (2003), Nr. 2, S. 75–88. – ISSN 1433–2833

[KKKL97] Kim, Hang J. ; Kim, Kyung H. ; Kim, Sang K. ; Lee, Jong K.: On-line recognition of handwritten Chinese characters based on hidden Markov models. In: *Pattern Recognition* 30 (1997), Nr. 9, S. 1489–1500

[Lev66] Levenshtein, Vladimir I.: Binary codes capable of correcting deletions, insertions and reversals. In: *Soviet physics doklady* Bd. 10, 1966, S. 707

[LJN04] Liu, Cheng lin ; Jaeger, Stefan ; Nakagawa, Masaki: Online Recognition of Chinese Characters: The State-of-the-Art. In: *IEEE TRANS. PATTERN ANAL. MACH. INTELL* 26 (2004), Nr. 2, S. 198–213

[LYWW11] Liu, Cheng-Lin ; Yin, Fei ; Wang, Da-Han ; Wang, Qiu-Feng: CASIA Online and Offline Chinese Handwriting Databases. In: *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, 2011. – ISSN 1520–5363, S. 37–41

[NASS01] Nakai, Mitsuru ; Akira, N. ; Shimodaira, Hiroshi ; Sagayama, S.: Substroke approach to HMM-based on-line Kanji handwriting recognition. In: *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, 2001, S. 491–495

[NSS03] Nakai, Mitsuru ; Shimodaira, Hiroshi ; Sagayama, Shigeki: Generation of hierarchical dictionary for stroke-order free kanji handwriting recognition based on substroke hmm. In: *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on* IEEE, 2003, S. 514–518

[NSSS02] Nakai, Mitsuru ; Sudo, Takashi ; Shimodaira, Hiroshi ; Sagayama, Shigeki: Pen pressure features for writer-independent on-line handwriting recognition based on substroke HMM. In: *Pattern Recognition, 2002. Proceedings. 16th International Conference on* Bd. 3 IEEE, 2002, S. 220–223

[Rab89] Rabiner, Lawrence: A tutorial on hidden Markov models and selected applications in speech recognition. In: *Proceedings of the IEEE* 77 (1989), Nr. 2, S. 257–286

[Staa] Statista: *Marktanteile der Betriebssysteme am Endkundenabsatz von Smartphones weltweit von 2009 bis 2013.* `http://de.statista.com/statistik/daten/studie/12885/umfrage/marktanteil-bei-smartphones-nach-betriebssystem-weltweit-seit-2009/`. – Eintrag vom 13.07.2014

[Stab] Statista: *Marktanteile der mobilen Betriebssysteme am Absatz von Smartphones in Deutschland von Januar 2012 bis Mai 2014.* `http://de.statista.com/statistik/daten/studie/225381/umfrage/marktanteile-der-betriebssysteme-am-smartphone-absatz-in-deutschland-` – Eintrag vom 13.07.2014

[Sta04] Stamp, Mark: A revealing introduction to hidden Markov models. In: *Department of Computer Science San Jose State University* (2004)

[SVD04] Schimke, Sascha ; Vielhauer, Claus ; Dittmann, Jana: Using adapted levenshtein distance for on-line signature authentication. In: *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on* Bd. 2 IEEE, 2004, S. 931–934

[TIM+02] Tokuno, Junko ; Inami, Nobuhito ; Matsuda, Shigeki ; Nakai, Mitsuru ; Shimodaira, Hiroshi ; Sagayama, Shigeki: Context-

dependent substroke model for HMM-based on-line handwriting recognition. In: *Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on* IEEE, 2002, S. 78–83

[TNS92] Takami, Jun-ichi ; Nagai, Akito ; Sagayama, Shigeki: Speaker Adaptation of the SSS (Successive State Splitting)-Based Hidden Markov Network for Continuous Speech Recognition. In: *Proc. SST92 (Brisbane)* (1992), S. 437–442

[TSW90] Tappert, C. C. ; Suen, C. Y. ; Wakahara, T.: The State of the Art in Online Handwriting Recognition. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 12 (1990), aug, Nr. 8, S. 787–808. – ISSN 0162–8828

[YYS90] Yamada, Hiromitsu ; Yamamoto, Kazuhiko ; Saito, Taiichi: A nonlinear normalization method for handprinted Kanji character recognition-line density equalization. In: *Pattern Recognition* 23 (1990), Nr. 9, S. 1023–1029

[ZN12] Zhu, Bilan ; Nakagawa, Masaki: *Online Handwritten Chinese/Japanese Character Recognition*. `http://dx.doi.org/10.5772/51474`. Version: 2012