Hochschule
Ravensburg-Weingarten
Technik | Wirtschaft | Sozialwesen

Fakultät für Elektrotechnik und Informatik

Studiengang Informatik Master

**Master-Thesis**

# On-Line Handschriftenerkennung (OLCR) der Deutschen Einheitskurzschrift

zur Erlangung des akademischen Grades

Master of Science der Informatik

vorgelegt von:

Niko Will, B.Sc.

21. Dezember 2010

1. Gutachter: Prof. Dr. rer. nat. Martin Hulin

2. Gutachter: Prof. Dr. rer. nat. Wolfgang Ertel

## Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher, in gleicher oder ähnlicher Form, keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

_____ _____

Unterschrift Ort, Datum

# Zusammenfassung

Mit dem Aufkommen immer kompakter werdenden Computern, sogenannten Tablets, die meist keine physikalische Tastatur mehr besitzen, steigt auch der Wunsch nach einer adäquaten Eingabemethode für solche Geräte. Die Deutsche Einheitskurzschrift (DEK) bietet die Möglichkeit, handschriftliche Notizen in Echtzeit niederzuschreiben. Eine Handschriftenerkennung für diese Kurzschrift kann die Einsatzmöglichkeiten mobiler Computer um viele nützliche Bereiche erweitern.

In der Arbeit werden die Besonderheiten der DEK analysiert und die Anwendung bisheriger Ansätze für die Erkennung anderer Schriften, hinsichtlich der DEK, untersucht. Bisherige Arbeiten für das Erkennen der DEK konnten nicht gefunden werden. Es wird eine Möglichkeit der Segmentierung eines Kurzschriftwortes in einzelne Kurzschriftzeichen aufgezeigt, die diese Besonderheiten berücksichtigt. Die Klassifikation eines segmentierten Zeichens erfolgt anhand eines 8-Direction-Chain-Codes und weiterer Merkmale wie Größe, Schleifen innerhalb des Zeichens und den Übergangswinkeln eines Zeichens zu seinen Nachbarn. Zwei Classifier[1] werden mit den extrahierten Merkmalen trainiert und getestet. Beide erreichen für gelernte Muster von einem Schreiber eine Erkennungsrate von über 98% auf neuen Mustern.

[1] Komponente, die für ein eingegebenes Muster als Ausgabe die Klasse bzw. eine Liste aller möglichen Klassen mit deren Wahrscheinlichkeit liefert.

# Vorwort

An dieser Stelle möchte ich mich bei meinem Erstbetreuer Herrn Prof. Hulin für die inspirierenden Gespräche und die ständige Betreuung während der Arbeit bedanken. So mancher Denkanstoß, vor allem während der Entwicklung der Segmentierung, war überaus nützlich für den Erfolg dieser Arbeit. Ebenfalls großer Dank gebührt Herrn Prof. Ertel, der sich als Zweitbetreuer der Arbeit zur Verfügung stellte. Darüber hinaus stand er mir vor allem in Fragen der Mustererkennung und bei der Auswahl der Algorithmen stets behilflich zur Seite. Außerdem geht ein besonderes Dankeschön an meine Familie, die mich während des Studiums und vor allem während der Master-Thesis immer unterstützten und motivierten. Allen voran meine Verlobte Nadine Zubler, die mir stets den Rücken freihielt und sich so manches Problem anhören musste. Ein Dank auch an meinen Hund Elvis, der das letzte halbe Jahr viel zu kurz kam, mir es aber dennoch nicht übel nahm. Zu guter Letzt möchte ich mich bei Michael Wagner für die Gespräche und das Korrekturlesen der Arbeit bedanken.

# Inhaltsverzeichnis

# 1. Einleitung

Stenografie ist ein Kurzschriftsystem, das es ermöglicht, gesprochene Sprache oder eigene Ideen in Echtzeit zu notieren. Die 1924 verabschiedete Deutsche Einheitskurzschrift (DEK) wurde 1938 und 1968 reformiert und ist seither das Standardstenographiesystem in Deutschland und Österreich. Sie gliedert sich in die drei aufeinander aufbauenden Stufen Verkehrs-, Eil- und Redeschrift mit jeweils höherem Abstraktionsniveau. Bereits bei der niedrigsten Stufe können von geübten Schreibern bis zu 120 Silben pro Minute erreicht werden. Bis 1990 gehörte das Erlernen der DEK zum Standardfach jeder kaufmännischen Ausbildung. Mit dem Aufkommen von Diktiergeräten und Weiterentwicklungen auf dem Gebiet der Spracherkennung verlor die DEK an Bedeutung und wird heute nur noch in speziellen Kursen von Stenographievereinen oder in bestimmten Studiengängen vermittelt.

Dennoch betonte Wolfgang Behm, Parlamentsstenograf und ehemaliger Leiter des Stenographischen Dienstes des Bundestags, dass gerade für die Sitzungsprotokolle es derzeit kaum Alternativen zum klassischen stenographischen Protokoll einer Parlamentssitzung gibt. In solchen Protokollen muss laut Behm auch aufgeführt werden, wenn ein Redner von einer Fliege umkreist wird und nach dieser schlägt oder wenn es Zwischenrufe aus dem Publikum gibt. Dies sind zwei Beispiele, bei denen z.B. eine Spracherkennungssoftware an ihre Grenzen stößt. Ein Wechsel zur Maschinenstenographie schließt Behm aufgrund von fehlendem Personal mit den notwendigen Kenntnissen aus. Auch wenn hier eine schnellere redaktionelle Nachbearbeitung der Protokolle möglich wäre, da die Texte bereits maschinell erfasst wären. Behm kommt zu dem Schluss, dass neue Technologien die Stenographie im Arbeitsprozess der parlamentarischen Protokollierung immer mehr umgeben und ergänzen werden. Völlig wegzudenken sei sie aber gerade für diese besondere Aufgabenstellung nicht. [1]

Weiter zeigen die jüngsten Entwicklungen der PC Branche, dass der Trend in vielen Bereichen weg von teuren und teilweise großen Notebooks hin zu günstigen und überall einsetzbaren Tabletgeräten (wie dem iPad von Apple) geht. Die hohe Mobilität, die solche Geräte bieten, wird durch das Einsparen einer vollwertigen Tastatur erkauft. Diese wird durch eine Bildschirmtastatur ersetzt, die allerdings durch die Geschwindigkeit der Texterfassung nur sehr begrenzt zur Eingabe längerer Texte dienen kann. Software für die Texterkennung von Langschrift gibt es auf dem Tabletmarkt bereits mehrere Jahre, welche mittlerweile auch sehr gute Ergebnisse bei der Erkennung liefert. Doch auch hier ist die Geschwindigkeit bei der Erfassung der Texte für einen produktiven Einsatz meist zu gering. Hier kann die Möglichkeit zur Eingabe mittels der DEK ganz neue Anwendungsgebiete erschließen oder bestehende vereinfachen.

## 1.1. Problemstellung

Das Schriftbild der DEK unterscheidet sich gravierend von anderen Schriften und auch von Kurzschriftsystemen anderer Sprachen (siehe Kapitel 2.1). Daher können bereits vorhandene Verfahren für andere Schriften nicht einfach übernommen werden, sondern es muss eine neue Methode gefunden werden, die die Besonderheiten der DEK berücksichtigt.

Die größte Herausforderung bei Mustererkennungsproblemen ist stets die Extraktion von eindeutigen Merkmalen, anhand derer eine genaue Klassifikation ermöglicht wird. Das Auffinden solch eindeutiger Merkmale eines Wortes oder eines Wortteils stellt damit eine große Herausforderung dieser Arbeit dar.

Bei der Handschriftenerkennung als ein Gebiet der Mustererkennung kann es zusätzlich notwendig sein, die Eingabe in einzelne Zeichen für den eigentlichen Erkennungsalgorithmus zerlegen zu müssen. Diese Aufgabe ist meist ebenso aufwändig wie die Erkennung der Zeichen selber.

Wenn diese Aufgaben erfolgreich gelöst sind, kann auf eine Vielzahl von Algorithmen zurückgegriffen werden, die sich bereits in anderen Systemen zur Handschriftenerkennung bewährt haben (siehe Kapitel 3). Jedoch darf auch die Auswahl der richtigen Methode und die Umsetzung für das eigene Problem hierbei nicht unterschätzt werden.

## 1.2. Zielsetzung

In dieser Arbeit sollen mögliche Lösungswege für die Onlineerkennung (OLCR, siehe Kapitel 2.2) der Verkehrsschrift, also der untersten Stufe der DEK, evaluiert werden. Dabei werden sowohl globale wie auch lokale Ansätze betrachtet und bewertet. Ersteres entspricht einer Worterkennung mittels Wörterbuch, letzteres der Erkennung einzelner Buchstaben eines Wortes. Schließlich wird die vielversprechendste Lösung genauer untersucht und in Form einer Machbarkeitsstudie umgesetzt.

Durch den begrenzten zeitlichen Rahmen dieser Arbeit werden nur Mitlaute und Selbstlaute nach §2 bzw. §3 der Wiener Urkunde [2] (siehe Kapitel 2.1) in dieser Arbeit umgesetzt. Auf das Erkennen von Kürzeln nach §5 und Berücksichtigung von Kontextinformationen (siehe Kapitel 2.1) wird verzichtet. Dennoch wird bei der Analyse und Auswahl der Methode darauf geachtet, dass sich diese später mit der gewählten Methode unter vertretbarem Aufwand umsetzen lassen.

Weiter ist es nicht Bestandteil dieser Arbeit, dass die Implementierung der ausgewählten Methode jegliche Annehmlichkeiten einer vollwertigen Software zur Handschriftenerkennung bietet. D.h., dass weder Funktionen zur Nachbearbeitung einer Eingabe noch besondere Vorkehrungen für eine fehlertolerante Eingabe getroffen werden. Beispiel für letzteres wäre es, wenn z.B. kurz der Stift während der Eingabe eines zusammenhängenden Zeichens angehoben wird, sodass ein einzelnes Zeichen von der Hardware als zwei separate Eingaben erkannt wird.

# 2. Grundlagen

In diesem Kapitel soll eine kurze Übersicht über einige Grundlagen gegeben werden, welche zum Verständnis der Arbeit wichtig sind.

## 2.1. Deutsche Einheitskurzschrift

Wie bereits erwähnt, wurde die DEK so wie sie heute noch verwendet wird, zum letzten Mal 1968 in der Wiener Urkunde [2] festgeschrieben. Im 1. Teil dieser Urkunde geht es konkret um die in der Arbeit aufgegriffene Verkehrsschrift. §1 legt hierbei den Schreibraum mit vier wichtigen, horizontalen Linien (Obergrenze, Oberlinie, Grundlinie, Untergrenze) fest, in denen geschrieben wird. Der Abstand zwischen zwei Linien wird Stufe genannt. Des Weiteren gibt die Urkunde eine Einordnung der Zeichen anhand ihrer Größe an. Zeichen, die eine halbe Stufe hoch oder kleiner sind entsprechen kleinen Zeichen. Zeichen mit einer Höhe von einer Stufe den mittleren Zeichen, alle größeren Zeichen dementsprechend den großen Zeichen. Abbildung 1 zeigt den Schreibraum und Beispiele für die einzelnen Zeichengrößen, wobei die großen Zeichen wiederum weiter unterteilt werden können in zwei- und dreistufige Zeichen. Es gilt zu beachten, dass die Zeichen nicht zwangsweise auf der Grundlinie geschrieben werden (Erläuterung folgt).

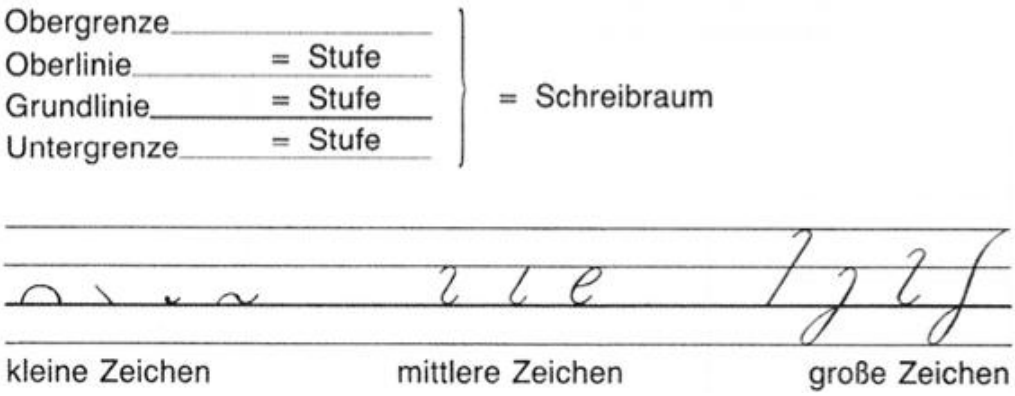



**Abbildung 1: Definition des Schreibraums und Zeichenbeispiele [3]**

Bei den Zeichen kann zusätzlich unterschieden werden nach „Mitlauten“, die in §2 und „Silbenzeichen“ und „Kürzeln“, die in §5 der Urkunde definiert werden. Insgesamt gibt es 55 Mitlaute, 5 Silben und Silbenteile und 103 Kürzel für Vor- und Nachsilben sowie Wörter und Wortstämme. Dabei kommt es vor, dass ein Zeichen sowohl als Mitlaut sowie als Silbenzeichen oder Kürzel definiert ist. Der Kontext, in dem das Zeichen im Wort oder Satz auftritt, gibt dann die eigentliche Bedeutung an.

Selbstlaute nach §3 ergeben sich durch sinnbildliche Selbstlautandeutung. Das bedeutet konkret, dass Selbstlaute am folgenden Mitlaut angedeutet werden. Vereinfacht gesagt ergibt sich ein Selbstlaut durch die Verbindung von der Grundlinie zum ersten Zeichen bzw. durch die Verbindung zwischen zwei Zeichen. Eine Unterscheidung kann anhand von drei Eigenschaften getroffen werden: 1. Der Länge der Verbindung, 2. ob der nächste Mitlaut verstärkt geschrieben wird (entsteht durch größeren Druck mit speziellen Stenofüllern oder –bleistiften) und 3. anhand der Stellung des folgenden Mitlauts. Bei letzterem wird zwischen Hoch- und Tiefstellung unterschieden, die in der Regel eine halbe Stufe beträgt. Dabei wird die Grundline für alle folgenden Zeichen sinnbildlich mitverschoben. Abbildung 2 zeigt alle möglichen Selbstlaute mit dem Mitlautzeichen für ein „b“.

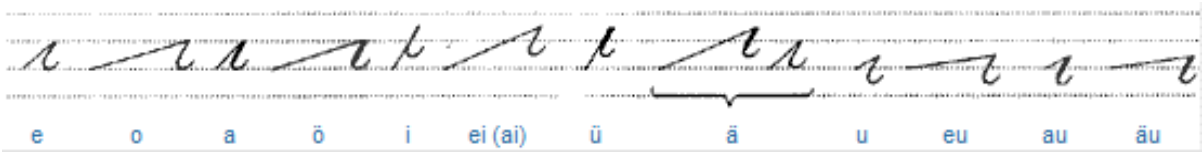



**Abbildung 2: Selbstlaute werden nur angedeutet [2]**

Weiter definiert die Urkunde diverse Schreibregeln für einen zusätzlichen Geschwindigkeitsgewinn beim Schreiben. Zum Beispiel gibt es keine Großschreibung, keine Mitlautverdopplung (außer beim „l“) und keine Silbendehnung durch Selbstlaute oder Dehnungs-h. Dadurch gibt es erneut Fälle, bei denen ein Wort in der DEK mehrere Bedeutungen haben kann wie beispielsweise Beet und Bett. Hier muss ebenfalls wieder der Kontext des gesamten Dokuments berücksichtigt werden um das korrekte Wort erkennen zu können. Weitere Schreibregeln für Sonderfälle wie z.B. Eigennamen, sind für die Handschriftenerkennung irrelevant und somit für das Verständnis der Arbeit nicht notwendig.

Im Vergleich zu Blockschriften bei welchen die Zeichen innerhalb eines Wortes einzeln für sich geschrieben werden, werden bei kursiven Schriften, wie der DEK, die Zeichen verbunden und in einem Strichzug geschrieben. Das Trennen innerhalb eines Wortes kommt bei der DEK nur in bestimmten Sonderfällen vor. Für eine korrekte Erkennung eines getrennten Wortes muss ebenfalls der Kontext des Satzes betrachtet werden. [2][3]

## 2.2. On-line Character Recognition

Bei der Handschriftenerkennung wird generell zwischen Online (OLCR) und Offline (OCR) unterschieden. Bei OCR wird mit eingescannten oder abfotografierten Grafiken gearbeitet, aus denen zunächst die eigentliche Schrift durch Bildverarbeitungstechniken extrahiert werden muss, um im Anschluss die Erkennung vornehmen zu können. Bei OLCR hingegen wird ein spezielles Eingabegerät verwendet, das die Bewegung eines Schreibgeräts erfasst und in kartesische Koordinaten umwandelt. Dabei spielen vor allem die Samplingrate und die Auflösung des Eingabegeräts eine entscheidende Rolle für die spätere Erkennung. Ersteres gibt die Geschwindigkeit an, mit der die Stiftdaten erfasst werden können. Letzteres gibt Aufschluss darüber, wie genau das Eingabegerät die Position des Schreibgeräts erfassen kann. Der Vorteil der Onlineerkennung liegt im geringeren Aufwand für die Erfassung der Daten im Vergleich zur Offlineerkennung. Des Weiteren kann bei OLCR auf die zeitliche Information der Eingabe zugegriffen werden d.h. die Reihenfolge der gezeichneten Striche ist bekannt. OCR Methoden können dabei problemlos auch auf OLCR Probleme angewandt werden. Dazu genügt es die Eingabe in eine Grafikdatei zu schreiben und diese zu verwenden.

## 2.3. Microsoft INK API

Für die spätere Implementierung der ausgewählten Methode wird die Microsoft INK API, als Teil des Tablet PC Platform SDKs, verwendet. Diese API bietet dem Programmierer eine standardisierte Schnittstelle zum Eingabegerät sowie Methoden zur Integration von speziellen Anforderungen in Kombination mit einem Eingabegerät in das Betriebssystem Windows selbst. Dadurch lassen sich Stifteingaben in eigenen Programmen verwenden ebenso wie z.B. die windowseigene Handschriftenerkennung um neue Schriften zu erweitern.

Obwohl es logisch erscheint, die Handschriftenerkennung von Windows zu wählen, um die Kurzschrift in jeder Anwendung nutzen zu können, habe ich mich bei der späteren Umsetzung bewusst dagegen entschieden. Meine Erfahrungen mit der DEK zeigen mir, dass es wenig sinnvoll ist, einen so begrenzten Platz für die Eingabe zur Verfügung zu haben, wie es mir das Input Panel

von Microsoft bietet (siehe Abbildung 3). Schließlich wird die Stenographie verwendet, um längere Texte schnell niederschreiben zu können. Beim Input Panel hingegen müsste der Bediener bereits nach einigen Wörtern die „Einfügen“ Taste verwenden, um den geschriebenen Text in ein Dokument einzufügen. Da dies den Schreibfluss und eventuell den kreativen Prozess des Anwenders stören würde, macht es aus meiner Sicht mehr Sinn, die Möglichkeit zu haben, über mehrere Linien hinweg in ein Dokument zu schreiben, analog zu einem Blatt DIN A4 Papier (ähnlich dem Windows Journal). Leider war es mir nicht möglich, erfahrene Stenographen zu diesem Sachverhalt zu befragen, da mehrere Anfragen an den Deutschen Stenographenbund im Sande verliefen.

Aus oben genanntem Grund wurde die Möglichkeit, eigene Recognizer[2] für das Input Panel zu erstellen, nicht näher betrachtet. Jedoch möchte ich hier den anderen Weg kurz beschreiben um zu zeigen, welche Möglichkeiten mir die API von Microsoft bietet, um Entscheidungen, die im Kapitel 4 getroffen wurden, eventuell besser nachvollziehen zu können.


**Abbildung 3: Input Panel von Microsoft.**

Prinzipiell lässt sich zu jedem Oberflächensteuerelement ein „InkOverlay“ hinzufügen, um Stifteingaben auf diesem Steuerelement abfangen zu können. Möchte man dem Benutzer allerdings eine bestimmte Eingabefläche bieten, kann hierfür direkt das Steuerelement „InkPicture“ verwendet werden. Bei beiden Wegen kann auf Stiftereignisse wie dem Schreiben oder dem Drehen des Stifts (Radiergummifunktion) reagiert werden. Außerdem lässt sich die Stifteingabe über eine „InkCollection“ auslesen. Diese beinhaltet alle „Stroke“ Objekte des jeweiligen Steuerelements. Ein solches Stroke Objekt repräsentiert genau eine Stifteingabe vom Aufsetzen des Stifts auf der Eingabefläche bis zum Abheben. Dies wird auch Pen-Down Pen-Up Zyklus genannt. Die API bietet dem Programmierer die Möglichkeit, die rohen Paketdaten eines Stroke Objekts auszulesen, die vom Eingabegerät erfasst wurden (siehe Abbildung 4). Das sind zum einen die kartesischen Kooridnaten im Ink Space, dessen Auflösung in der Regel dem 27 fachen der Bildschirmauflösung entspricht, zum anderen Informationen über die Druckstärke und weiteren tabletspezifische Eigenschaften. Des Weiteren lassen sich von der API berechnete Eigenschaften wie Schnittpunkte, Hoch- und Tiefpunkte und Größe abfragen und Transformationen wie z.B. eine Scherung, Skalierung, Drehung usw. auf die Eingabe anwenden. Zusätzlich wird die Eingabe in einer Bezier-Darstellung approximiert, die dem entspricht, was der Benutzer tatsächlich sieht. Dies ist wichtig, da die tatsächlich erkannten Punkte des Tablets teilweise starkes Rauschen aufweisen, was einerseits nicht

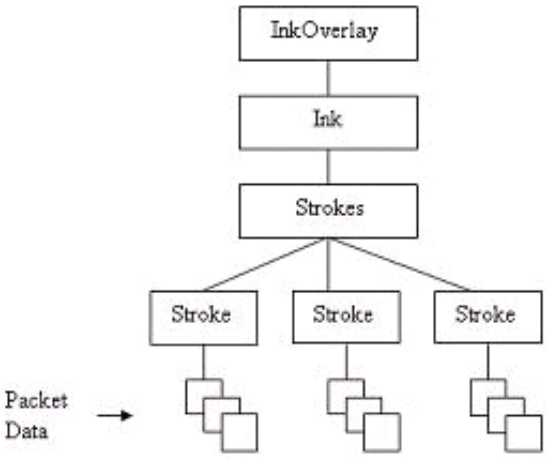



**Abbildung 4: Das Objektmodell für Ink.**

Quelle: http://i.msdn.microsoft.com/dynimg/IC104403.gif)

[2] Begriff von Micorosoft für einen Classifier.

schön anzusehen ist andererseits bei der Weiterverarbeitung ansonsten erst gefiltert werden muss. Mit der Bezier-Darstellung erhält der Programmierer gleich eine weitestgehend geglättete Darstellung der Eingabe.

Trotz der Entscheidung gegen die Erstellung eines Recognizers für das Input Panel ist es doch später problemlos möglich, die implementierte Methode auch als Recognizer umzusetzen, denn diese bieten Zugriff auf dieselben Eigenschaften wie das InkOverlay und das InkPicture. [4]

# 3. Stand der Technik

Da für das Erkennen der DEK keine bisherigen Arbeiten gefunden werden konnten, gebe ich in diesem Kapitel einen Überblick über bisherige Arbeiten im Bereich der Handschriftenerkennung. Da es hierfür mehrere Ansätze gibt gliedert sich dieses Kapitel in die entsprechenden Teildisziplinen. Grundsätzlich gibt es drei Kategorien für die Erkennung einer Handschrift:

- Globale Ansätze: Ein Wort wird als Ganzes erkannt. Die Merkmale für die Klassifikation sind hier z.B. die Anzahl der horizontalen und vertikalen Striche, Ober- und Unterlängen sowie Schleifen des gesamten Wortes.
- Lokale Ansätze: Hier wird das Wort zuerst anhand von bestimmten Merkmalen in einzelne Segmente zerlegt, die anschließend einzeln erkannt werden.
- Hybride Ansätze: Hierbei werden globale und segmentierende Ansätze kombiniert.

Im ersten Teil dieses Kapitels gehe ich auf einzelne Ansätze aus diesen Bereichen näher ein. Hybride Ansätze werden dabei jedoch nur begrenzt behandelt und in den anderen Kapiteln aufgeführt, je nachdem wo es sinnvoll erschien. Es wurden zwar weitaus mehr Publikationen gelesen als hier aufgeführt, doch da es aus meiner Sicht wenig Sinn macht, Ansätze zu durchleuchten, die von vorneherein keinen Beitrag zum Erfolg der Arbeit liefern konnten, wurden diese hier nicht aufgeführt. Dazu zählen z.B. auch Methoden für die Erkennung von chinesischen Schriftzeichen, da solche Ansätze aufgrund der Schrift sehr speziell sind. Da es sich bei der DEK um eine kursive Schrift handelt (siehe Kapitel 2.1), werden hauptsächlich Ansätze für solche Schriften betrachtet. Auch Arbeiten für die Erkennung anderer Kurzschriftsysteme werden in Teilen angeführt.

Im Anschluss daran werden bisherige Arbeiten auf dem Gebiet der Kontexterkennung gezeigt, welche normalerweise genutzt wird, um die Erkennungsrate der oben genannten Ansätze zu verbessern. Bei der DEK ist die Kontexterkennung jedoch aufgrund der Mehrdeutigkeit einzelner Zeichen (siehe Kapitel 2.1) zwingend erforderlich. Abschließend wird noch auf eine Arbeit eingegangen, die mittels Latex DEK Zeichen erzeugt und einige Ideen für die spätere Erkennung geliefert hat.

## 3.1. Globale Ansätze

Die Erkennung anhand eines kompletten Wortes ist meist sehr aufwändig. Das liegt zum einen daran, dass es schwierig ist, Eigenschaften zu finden, die einzelne Wörter untereinander eindeutig identifizieren. Zum anderen muss für diese Form der Erkennung ein umfangreiches Wörterbuch der jeweiligen Sprache aufgebaut werden. Das Durchsuchen eines solchen Wörterbuchs ist ebenfalls sehr zeitintensiv und benötigt jede Menge Speicher zum Vorhalten des gesamten Wörterbuchs. Ein weiterer Nachteil bei globalen Ansätzen ist, dass falsch geschriebene Wörter entweder nicht oder komplett falsch erkannt werden, wohingegen bei segmentierenden Ansätzen der Fehler einfach in die Ausgabe übernommen werden würde. Dieses Verhalten zeigt sich ebenso bei Eigennamen, die in einem Wörterbuch normalerweise nicht vorkommen. [5]

W. Cho et al. zeigen in [6] eine Möglichkeit zur Erkennung von kursiven Wörtern mit Hidden Markov Modellen (HMM) anhand von gescannten Briefköpfen. Da HMMs mit zeitlichen Abhängigkeiten arbeiten, die bei OCR Anwendungen nicht vorhanden sind, setzen die Autoren ein so genanntes „Sliding Window“ ein. Damit wird die Schrift in Schreibrichtung „gescannt“, um somit einen annähernden zeitlichen Verlauf der Schriftzeichen zu erhalten. Dieses Scannen erfolgt, indem die Grafik des Wortes in eine Sequenz von schmalen vertikalen Bereichen geteilt wird (siehe Abbildung 5). Diese Bereiche können dabei auch leicht überlappend und in diversen Breiten ausgeführt werden, um den Algorithmus besser an die Aufgabenstellung anzupassen. Anhand der

lokalen Merkmale eines Bereichs werden mittels der „Principal Component Analysis“ (PCA) Featurevektoren extrahiert. Anschließend wird die „Vector Quantization Technique“ angewendet, um die stetigen Featurevektoren in ein sogennantes „Codeword“ mit diskreten Werten umzusetzen. Dies geschieht durch ein „Codebook“, welches einen Satz von Codewords enthält und mit speziellen Trainingsbeispielen zuvor erstellt wurde. Das Auffinden des richtigen Codewords erfolgt durch eine Abstandsberechnung der Eingabe zu den einzelnen Codewords des Codebooks. Nach dem erfolgreichen Scannen des Wortes erhält man so eine Sequenz von Codewords. Mit dieser Sequenz wird anschließend ein Netz von mehreren Hidden Markov Modellen durchlaufen, genau ein Modell für jeden Buchstaben des Alphabets und zusätzlich noch eine Gruppe von Modellen um die Verbindungen zwischen Buchstaben zu modellieren. Zu jedem Codeword der Sequenz wird der wahrscheinlichste Pfad eines Modells berechnet, der dann anschließend mit dem nächsten Codeword getestet wird. Da der Pfad mit der höchsten Wahrscheinlichkeit aber nicht unbedingt der richtige sein muss, grenzen die Autoren den Suchraum durch Pruning von nicht möglichen Wörtern anhand eines Lexikons bereits während der Suche ein. Sie räumen zwar ein, dass so nicht der optimale Pfad gefunden wird, jedoch gab es zur Zeit der Arbeit scheinbar keinen Algorithmus, der linguistisch korrekte Wörter liefert ohne aufwändig den gesamten Suchraum durchgehen zu müssen. Am Ende der Codewordsequenz erhält man so das wahrscheinlichste Wort des Lexikons. Trainiert mit 5000 und getestet mit 2700 Städtenamen, die aus einer Datenbank mit gescannten Briefköpfen stammen, erreichten die Autoren bei 10 Einträgen im Lexikon eine Erkennungsrate von 98,12%, welche jedoch bei steigender Lexikongröße rapide abnimmt und bei 10000 Einträgen bereits nur noch 67,09% als richtig erkennt. Für den Test verwendeten sie eine Sliding Window Größe von drei Pixeln. Die einzelnen Frames wurden mittels PCA in einen 30 dimensionalen Vektor zerlegt und anschließend mit einem 50 Codewords umfassenden Codebook quantisiert.

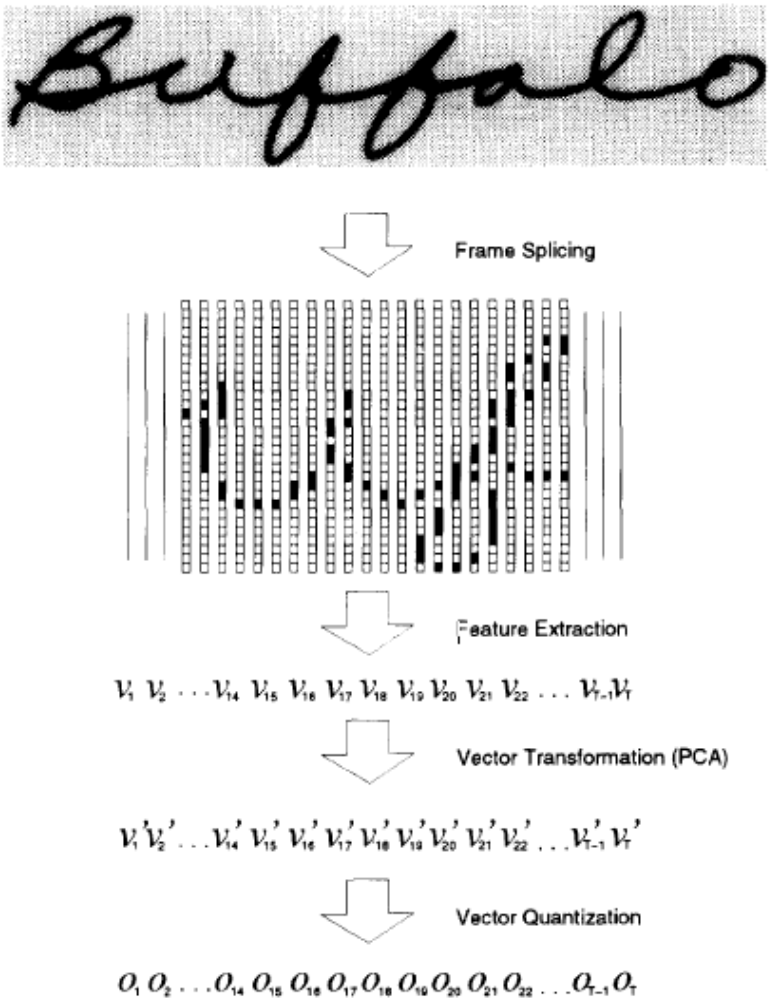



**Abbildung 5: Feature Extraction mittels "Sliding Window" [6].**

Duneau und Dorizzi stellten in [7] eine Erweiterung ihres Systems „Adresy“ (ADaptive REcognition SYstem) vor. Das System wurde speziell für Handheld PCs designt und verwendet deshalb nur wenig zeitaufwändige Algorithmen. In Bezug auf die Worterkennung stellen die Autoren zwei Behauptungen auf: Erstens sei es unmöglich, ein Wort als ein großes Symbol zu betrachten und mit einem Wörterbuch abzugleichen, da aufgrund der Fluktuationen beim Schreiben eines Wortes zu viele Wortmodelle je Wort nötig wären, was bei großem Vokabular den Suchraum sprengen würde. Zweitens sei es unmöglich einen Buchstaben innerhalb eines Wortes genau zu

lokalisieren, bevor der Buchstabe erkannt wurde. Um beiden Problemen zu begegnen, wählen die Autoren einen „analytischen Ansatz mit interner Segmentierung“. Ihr System kann in drei Bereiche untergliedert werden (siehe Abbildung 6): Die Lerneinheit, die Erkennungseinheit und die Anpassungseinheit. In der Lerneinheit wird anhand von vorher eingegebenen Worten für jeden Buchstaben ein Prototypenset erstellt. Dies geschieht auf dieselbe Weise, wie die Erkennungseinheit anfänglich verfährt: Die Eingabe eines Grafiktablets wird auf

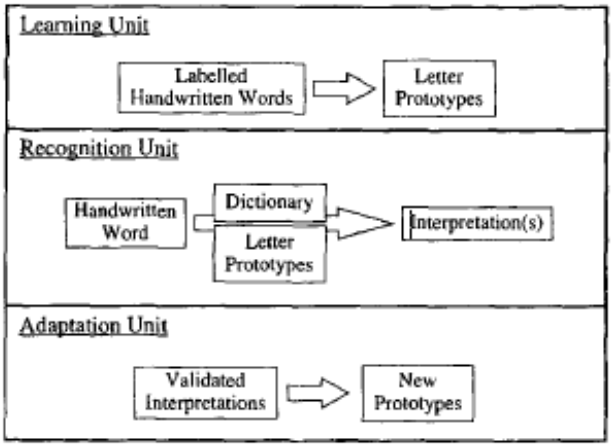



**Abbildung 6: Aufbau von Adresy [7].**

Transitionen, lokalen Extremas und bestimmten Segmentierungspunkten untersucht und anhand definierter Regeln in Teile zerlegt. Aus jedem Teil werden dann vier Merkmale extrahiert: die euklidische Distanz zwischen Anfangs- und Endpunkt, der Winkel zwischen der Verbindung von Anfangs- und Endpunkt und der horizontalen Achse sowie zwei Fourierkoeffizienten. In der Erkennungsphase werden die Merkmalsvektoren mit den Prototypen aus der Lerneinheit verglichen, um somit die passenden Buchstaben zu finden. Anschließend werden die gefunden Buchstaben als Ganzes mit dem Wörterbuch abgeglichen. Der Prozess zwischen Abgleich mit dem Prototypenset und dem Wörterbuch wird dabei solange wiederholt, bis eine oder mehrere zufriedenstellende Interpretationen der Eingabe gefunden wurde. Um eine gewisse Robustheit zu erreichen, führen die Autoren eine „virtuelle Hypothese“ ein. Wenn keine Übereinstimmung zwischen einem Buchstaben und einem Prototyp gefunden wird, wird versucht, diesen fehlenden Buchstaben durch statistische Berechnungen anhand des Wörterbuchs zu finden. Dies funktioniert nur, wenn maximal ein Buchstabe nicht erkannt wurde. Am Ende der Erkennung werden bis zu maximal fünf Interpretationen, gewichtet nach ihrer Wahrscheinlichkeit, an den Benutzer ausgegeben. Ist die wahrscheinlichste Interpretation nicht richtig, wählt der Benutzer entweder unter den anderen vier die Richtige aus, oder gibt das Wort erneut ein. Wenn eine der gefunden Interpretationen ausgewählt wird, wird die Eingabe zusammen mit der richtigen Interpretation in einem speziellen Trainingset abgelegt. Dieses Set wird bei der nächsten manuell angestoßenen Lernphase von der Lerneinheit zusätzlich zu den mitgelieferten Trainingsdaten bei der Erstellung der Prototypen berücksichtigt. So lernt das System auf Dauer die Handschrift seines Hauptverwenders. Interessant an diesem Ansatz ist, dass keine klassischen Lernalgorithmen wie HMMs oder neuronale Netze verwendet werden, sondern heuristische Suchen mit selbstdefinierten Vergleichsregeln. Mit 50 vorgelernten Wörtern erreicht das System eine Erkennungsrate von 90,3%. Bereits nach 500 gelernten Wörtern innerhalb des Adaptionsprozesses wird eine Rate von 95,9% erreicht. Getestet auf einem SUN Sparc funktionierte das System gut bei kleinen Wörterbüchern mit 1000 Wörtern. Bei größerem Vokabular von 10000 Wörtern benötigte es jedoch über zehn Sekunden, um ein Wort zu erkennen.

R.K. Powalka et al. beschreiben in [8], wie sie ihren „multiple interactive segmentation recognizer” (SEG), der einen segmentierenden Ansatz verwendet, um zwei globale Ansätze zu einem Hybridsystem erweitern. Die Erkennung bei der Segmentierung liefert gute Ergebnisse, wenn die Buchstaben in einem Wort klar trennbar sind. Kommt es jedoch zu Überschneidungen von Buchstaben aufgrund einer nicht idealen Handschrift, sinkt die Erkennungsrate. Im Durchschnitt erreicht SEG Erkennungsraten von 48%. Die Erweiterung sieht vor, dass dort, wo die Segmentation keine klaren Ergebnisse liefert, zusätzlich eine globale Methode bei der Entscheidungsfindung hilft. Dafür haben die Autoren zwei Worterkennungsalgorithmen entwickelt und getestet. Für beide muss aber zuerst eine Zonen Klassifikation durchgeführt werden, die erkennt, ob es nur eine mittlere Zone für kleine Zeichen, eine obere und mittlere für Zeichen mit Oberlängen, eine untere und

mittlere für Zeichen mit Unterlängen oder eine obere, mittlere und untere für Zeichen mit Ober- und Unterlängen gibt (siehe auch Abbildung 11 b auf Seite 17). Hierfür verwenden die Autoren eine Kombination von Histogramm- und Extrema-Methoden. Mittels Histogramm lässt sich das Vorhandensein von Zonen zwar besser auffinden, dafür lassen sich die Zonengrenzen nur schwer feststellen. Mit den Extremas hingegen eine bessere Abgrenzung zwischen zwei Zonen möglich ist, dafür ist aber das Erkennen von Zonen als Ganzes schwerer. Sind die Zonen eines Wortes bestimmt, sucht der erste Ansatz nach Ober- und Unterlängen (Ascender/Descender, AD) von Buchstaben, die über eine berechnete mittlere Zone hinausgehen wie z.B. beim „b“. Die zweite Methode berechnet Schnittpunkte von Abstrichen[3] mit einer imaginären Linie (Vertical Bars, VB), die durch die Mitte der mittleren Zone gezogen wird. Ein zusätzliches Verifikationsmodul für Buchstaben (Letter Verification, LV) nutzt die Ausgabe des SEG zur besseren Bewertung der extrahierten Merkmale der beiden globalen Methoden (siehe Abbildung 7). Anschließend wird das Wörterbuch nach Übereinstimmungen der extrahierten Merkmale durchsucht und die besten Ergebnisse zurückgeliefert. Das Aufbauen der Datenbank kann hierbei automatisch durch Festlegung der Werte jedes Buchstabens bei idealer Schrift erfolgen. Getestet haben die Autoren ihre Methoden mit 16 Schreibern, die jeweils die gleichen 200 Wörter erfassten. Diese Wörter waren wiederum Teil eines 4107 Wörter umfassendem Wörterbuchs. Im Vergleich aller Methoden einzeln, schnitten die globalen Ansätze wesentlich schlechter ab als SEG, wobei die Ansätze mit LV bereits nahe an den Erkennungsraten von SEG liegen. Bei Betrachtung der besten fünf gelieferten Ergebnisse verbesserte sich SEG nur um 5%, wohingegen sich die globalen Ansätze um 20% verbesserten und damit fast an die Erkennungsrate von SEG heranreichten. Kombiniert mit LV überstiegen die globalen Methoden SEG sogar. Werden immer mehr gelieferte Wortalternativen als Ergebnis des Erkennungsprozess zugelassen, steigt die Erkennungsrate der globalen Ansätze sogar auf über 75% bei 25 Wortalternativen. Jedoch scheint mir diese Zahl wenig praktikabel, denn ein System, das dem Benutzer 25 Wörter liefert, die evtl. richtig sein könnten, aber die Erkennungsrate, dass das erste Wort der Liste richtig ist, nur bei 48% liegt, würde mich wenig zufrieden stellen. Die Kombination von SEG mit VB, ADLV und VBLV in einen hybriden Ansatz liefern hingegen bereits bei fünf Wortalternativen eine Erkennungsrate von etwa 75%. Nur SEG in Verbindung mit AD ohne LV liefert immerhin noch knapp 66%. Bei nur einer Wortalternative verbesserte der hybride Ansatz mit der besten Methode SEG und VBLV die Erkennungsrate immerhin auf 61,9% im Vergleich zu SEG alleine mit 48,5%.

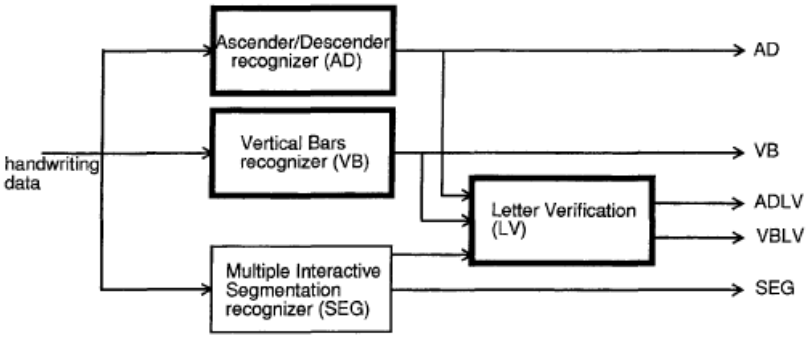



**Abbildung 7: Zusammenhang der einzelnen Recognizer [8].**

In [9] wird von Dehghan et al. ein globaler Ansatz für die Erkennung von Farsi aufgezeigt. Farsi ist eine Erweiterung der arabischen Schrift, somit ist laut Autoren auch eine Anwendung auf andere arabische Schriften möglich. Um die Merkmale zu extrahieren, wird auch hier ein Sliding Window eingesetzt. Ein Wort wird dabei vertikal in mehrere Bereiche geteilt, die sich gegenseitig zu 50% überlappen. Durch die großen Überlappungen bei diesem Ansatz sollen Besonderheiten der arabischen Schrift besser

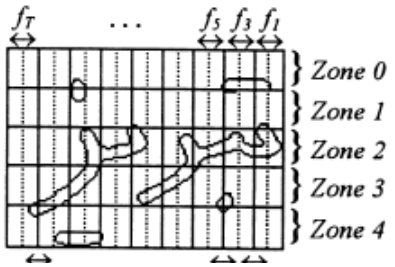



**Abbildung 8: Zerlegung einer Wortgrafik [9].**

[3] Striche, die von oben nach unten führen im Gegensatz zu Aufstrichen, die von unten nach oben geschrieben werden.

berücksichtigt werden. Jeder Bereich wird anschließend horizontal in fünf Zonen aufgeteilt (siehe Abbildung 8). Anschließend wird für jede Zone anhand der Strichrichtung innerhalb der Zone ein Histogramm gebildet und dieses mithilfe eines Codebooks quantisiert. Die Autoren verwendeten für das Codebook ein speziell trainiertes Kohonen-Netz. Dadurch sollen die Trainingszyklen des Hidden Markov Modells, das bei dieser Arbeit als einzelnes Netz ausgeführt ist, reduziert werden. Die Autoren erreichten bei der Anwendung auf handgeschriebenen Städtenamen von 198 arabischen Städten eine Erkennungsrate von 65% auf den Testdaten bei 17.000 Datensätzen, wovon 60% Trainings- und 40% Testdaten zufällig gewählt wurden.

Auch Günter und Bunke arbeiten in [10] mit einem Sliding Window zur Merkmalsextraktion und HMMs zur Erkennung. Im Gegensatz zu den anderen gezeigten Arbeiten, die auf ein Sliding Window setzten, geht es hierbei jedoch nicht um Wordspotting Aufgaben[4], sondern um normale Sprache. Das Fenster zum Scannen setzen die Autoren auf eine Breite von einem Pixel und der Höhe des Wortes. Außerdem verwenden sie keine Überlappung zwischen den Bereichen. Für jeden Bereich und somit jeden Pixel in der Breite eines Wortes werden neun Merkmale extrahiert. Zur Erkennung verwenden sie wie Cho et al. ein HMM je Buchstabe jedoch ohne extra HMMs für die Verbindungen zwischen diesen (siehe Abbildung 10). Dieser Ansatz wurde bis dahin nur bei Wordspotting Aufgaben verwendet, jedoch nicht für das Erkennen von normaler Sprache. Hier war es eher üblich ein HMM für alle Worte oder ein HMM pro Wort zu verwenden. Die Autoren begründen die Entscheidung, dass dadurch weniger Trainingsdaten benötigt würden. Um damit nun ganze Wörter bilden zu können, werden auch hier die einzelnen Modelle der Buchstaben zu einem großen Modell für jedes Wort des Wörterbuchs verbunden (siehe Abbildung 9). Die Klassifikation eines Wortes ergibt sich schließlich durch die beste Übereinstimmung des jeweiligen Modells mit einem der zusammengesetzten Modelle aus dem Wörterbuch. Mit 9861 Trainingsdaten und 3850 Testdaten von über 80 verschiedenen Personen auf einem Wörterbuch mit 2296 Wörtern erreichte das System eine Erkennungsrate von knapp 82%.

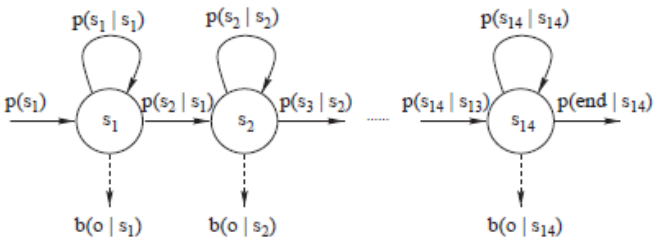



**Abbildung 10: HMM für einen einzelnen Buchstaben [10].**

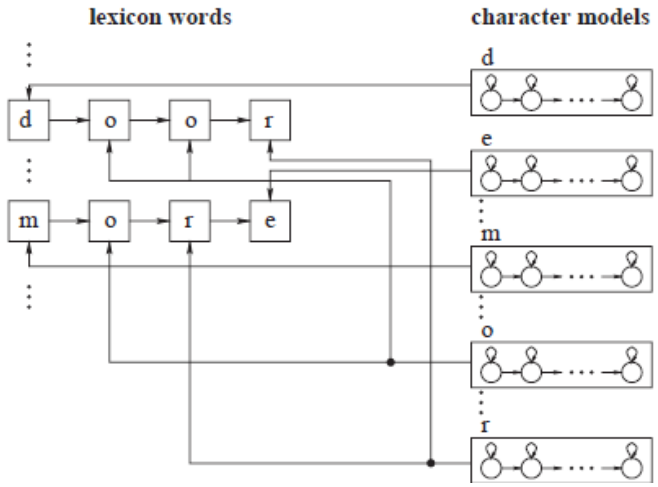



**Abbildung 9: Verbindung der Buchstabenmodelle führt zu Wortmodellen [10].**

Wie die genannten Arbeiten zeigen, werden Globale Ansätze meist nur bei OCR Aufgaben verwendet, bei denen eine Segmentierung ebenfalls sehr aufwändig wäre. Gerade sogenannte Wordspotting Aufgaben, wie sie z.B. bei Postsortierstellen notwendig sind, verwenden globale Ansätze. Bei diesen Aufgaben handelt es sich um begrenzte Lexika und meistens ist eine Segmentierung aufgrund unterschiedlicher Schreibstile und stark verrauschtem Eingangsmaterial

[4] Beim Wordspotting enthält das Wörterbuch nur wenige Einträge, dafür kann die Schreibweise sehr stark variieren. Solche Methoden werden z.B. zur automatischen Sortierung von Briefen anhand der Briefköpfe verwendet. Hierbei werden nur wenige Städtenamen im Wörterbuch gehalten, jedoch unterscheidet sich die Handschrift für jede Stadt zwischen unterschiedlichen Schreibern sehr stark.

sehr schwer bis gar nicht möglich. Problematisch ist jedoch immer der Suchraum bei großen Lexika, weshalb solche Systeme für eine normale Sprache, bei welcher das Vokabular schnell über 50000 Wörtern beinhaltet, wenig geeignet sind. Leider machen die wenigsten Autoren Angaben zu der Geschwindigkeit ihrer Algorithmen auf Wörterbüchern solcher Größe.

## 3.2. Segmentierende Ansätze

Bei diesen Ansätzen geht es darum, ein Wort in mehrere Bereiche zu unterteilen und die Bereiche separat zu erkennen. Die Bereiche müssen dabei nicht zwangsläufig den Buchstaben entsprechen, sondern sie müssen auf den Erkennungsalgorithmus abgestimmt werden. Im Vergleich zu den globalen Algorithmen aus Kapitel 3.1 benötigen die Algorithmen zum Segmentieren mehr Wissen über die Schrift der zu segmentierenden Wörter. Solche Algorithmen können in drei Kategorien eingeteilt werden:

- Regionsbasierte Methoden: Es wird versucht, Bereiche mit bestimmten Merkmalen zu finden und anhand dieser zu segmentieren.
- Konturbasierte Methoden: Anhand der Kontur werden dominante Punkte gesucht, an denen segmentiert werden soll.
- Erkennungsbasierte Methoden: Diese sind meist stark mit dem Erkennungsalgorithmus verwoben, sodass eine Segmentierung ohne Erkennung alleine nicht möglich ist.

Lu und Shridhar geben in [5] einen Überblick über diverse Algorithmen, die bis zum Erscheinen der Arbeit 1996 erfolgreich von anderen Autoren umgesetzt wurden. Im Bereich der segmentierenden Algorithmen nennen sie z.B. den Algorithmus von Boinovic und Srihari [11]. Sie zerteilen ein Wort vertikal in mehrere Bereiche, wobei jeder Bereich einen Buchstaben oder einen Teil davon beinhaltet. Um dies zu erreichen, werden zuerst die horizontalen Bereiche für die obere, mittlere und untere Zone identifiziert. Lokale Minimas an der unteren Kontur innerhalb der mittleren Zone deuten auf Verbindungen zwischen Buchstaben hin und werden daher als Segmentierungspunkte (PSP) vermerkt. Schleifen in der unteren Zone besitzen in dieser ebenfalls ein Minimum, der PSP wird dann an der Stelle gesetzt, an welcher die Linie von der unteren Zone wieder in die mittlere übergeht. Dabei ist ein PSP nur gültig, wenn es innerhalb einer festgelegten Distanz keinen weiteren PSP gibt und die vertikale Projektion aller Pixel unter einer bestimmten Schranke ist. Aus jedem Segment werden anschließend topologische und geometrische Merkmale extrahiert, um Buchstabenvorschläge und Wortformationen zu bilden. Das System erreichte 77% bei Daten, die von einer Person trainiert und getestet wurden und bis zu 63%, wenn die Daten von unterschiedlichen Personen stammten.

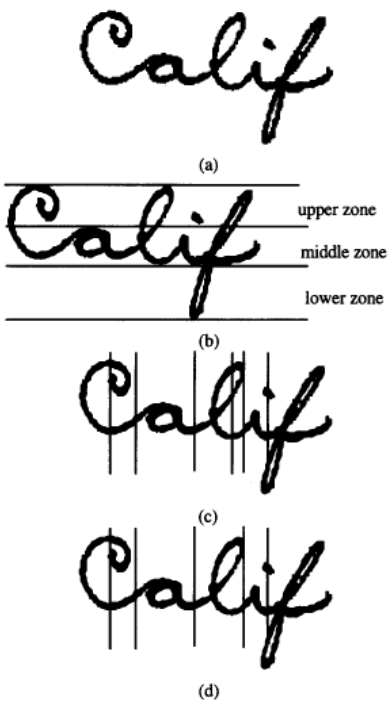



**Abbildung 11: Wortgrafik (a), eingeteilt in drei Zonen (b), suchen der PSPs (c + d) [5].**

Ein weiterer in [5] genannter Algorithmus stammt von Chen at al. [12] und basiert auf Singualrities (Einzigartigkeiten) und Regularities (Regelmäßigkeiten). Nach einer Reihe von Preprocessing Techniken wird jedes schwarze Pixel der Wortgrafik in eine der zwei Klassen „Islands“ und „Bridges“ eingeteilt. Islands sind dabei vertikale Strichführungen wohingegen Bridges zwei Islands miteinander verbinden. PSPs dürfen dabei nur auf den Bridges liegen und werden mit dem Viterbi [13] Algorithmus mit einer speziellen Kostenmatrix gesucht, bei der die Kosten auf den Bridges minimal sind. Die Autoren können zeigen, dass dabei in einem

Segment nie mehr als zwei Buchstaben liegen. Außerdem kann ein Buchstabe maximal in drei Segmente geteilt werden. Die erkannten Segmente werden anschließend einem HMM zugeführt. Auf einer USPS[5] Datenbank mit 1583 Wörtern, wovon 1489 als Trainings- und 94 als Testdaten verwendet wurden, erreichte das System eine Erkennungsrate von 68% bei Betrachtung nur des besten Wortes und 84% bei den besten zwei erkannten Wörtern.

Balestri und Masera [14] betrachten das Finden der optimalen PSPs als Suchraumproblem. Wenn man zunächst jeden Punkt innerhalb eines Wortes als PSP betrachtet, benötigt man nur eine Reihe von Funktionen, die diese Hypothese für jeden einzelnen Punkt bestätigen oder verwerfen. Da dies nicht in annehmbarer Zeit möglich wäre, verwendeten die Autoren den A*-Algorithmus für eine heuristische Suche. Die Hauptaufgabe besteht darin, die heuristischen Kostenfunktionen g(n) und h(n) zu definieren. Dafür müsse man sich den Satz aller möglichen PSPs als gewichteten Graph vorstellen, indem der Pfad mit den kleinsten Gewichten bzw. den niedrigsten Kosten gesucht wird. Die Kostenfunktionen geben die Autoren wie folgt an:

$$g(start) = 0$$

$$g(n) = g(n-1) + \frac{w}{p[width = w] * p[pos = s]}$$

$$h(start) = N$$

$$h(n) = h(n-1) - w$$

$$f(n) = g(n) + h(n)$$

Wobei N die Breite des Wortes, w die Distanz zwischen dem letzten angenommenen PSP und dem nächsten, s die Position des letzten PSPs, p[width = w] die Wahrscheinlichkeit eines w-Breiten Buchstabens und p[pos = s] die Wahrscheinlichkeit, dass der PSP an Stelle s einen tatsächlichen PSP darstellt. Wobei letzteres einen geschätzten Wert aufgrund einiger einfacher Merkmale darstellt. Die Wahrscheinlichkeit für die Breite eines Buchstabens wurde anhand statistischer Werte für die englische Schrift berechnet. Um die Berechnungsdauer weiter zu verbessern, entwickelten die Autoren eine weitere Bedingung, um manche Zweige die der A*-Algorithmus erweitert von vorneherein auszuschließen. Dies geschieht dann, wenn die Anzahl der geöffneten Knoten einen bestimmten Wert übersteigt. Damit ist die Zulässigkeit zwar nicht mehr gegeben, die Autoren behaupten aber, keine praktischen Probleme dabei beobachten zu können. Leider werden in der genannten Quelle keine Ergebnisse für den Algorithmus berichtet.

Bhowmik et al. trainieren ein neuronales Netz, um Wörter in Bangla in einzelne Buchstaben im Rahmen einer OCR Aufgabe zu zerlegen [15]. Banlga ist eine indische Schrift, die (wie alle indischen Schriften) kursiv geschrieben wird. Ähnlich der normalen deutschen Langschrift hat Bangla eine obere, mittlere und untere Zone. Buchstaben werden normalerweise entlang der „Matra“ miteinander verbunden. Dies ist die Grenze zwischen der oberen und der mittleren Zone. Wenn die Matra und die Baseline, welche die Grenze zwischen mittlerer und unterer Zone darstellt, erkannt werden, können die Zonen eindeutig bestimmt werden. Deshalb werden Segmentierungsalgorithmen für Bangla maßgeblich an ihren Fähigkeiten für die Erkennung der Matra gemessen. Die Autoren gehen jedoch einen anderen Weg und versuchen die Baseline und Headline anhand der horizontalen Projektion und des horizontalen Histogramms für schwarze Pixel zu erkennen. Dadurch lässt sich ebenfalls die mittlere Zone bestimmen.

[5] United States Postal Service, also der Deutschen Post in Amerika.

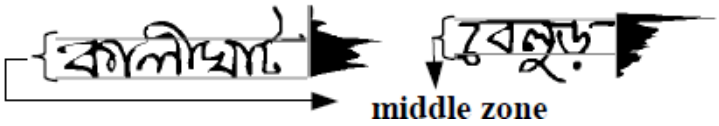



**Abbildung 12: Mittlere Zone durch die horizontale Projektion[15].**

Anschließend werden für jeden Konturpixel die Position relativ zur mittleren Zone, die Werte des vertikalen und horizontalen Histogramms und die Richtungen der L Nachbarpixel der Konturlinie jeweils vor und nach dem aktuellen Pixel als Merkmale extrahiert. Ein neuronales Netz entscheidet dann anhand der Merkmale, ob es sich um einen Segmentierungspunkt handelt oder nicht. Das Netz wurde zuvor mit manuell bestimmten Beispielen für beide Klassen trainiert. Tests ergaben, dass das Netz zum Übersegmentieren neigt, was aber in der Beschaffenheit mancher Buchstaben in Bangla zu begründen ist. Daher führen die Autoren ein Postprocessing durch, um die Ergebnisse des Netzes zu verfeinern. Getestet wurde der Algorithmus auf einer Datenbank mit berühmten Städtenamen aus dem westlichen Bengalen. Bei L = 8, was einer Größe des Merkmalsvektors von 20 entspricht, erzielten die Autoren mit 88,1% korrekt erkannten Segmentierungspunkten die besten Ergebnisse. Jedoch lagen auch die Ergebnisse für L = 5, 10, 15 mit 87,4%, 87,3% und 87,5% nur unwesentlich darunter.

Ma und Leedham zeigen in [16] einen Ansatz für die Erkennung von Renqun, eine Art Kurzschrift für die Chinesische Sprache, mit der geübte Schreiber bis zu 200 Wörtern pro Minute erfassen können. Da für Renqun die üblichen geometrischen Techniken um typische OLCR Features wie Ober- und Unterlängen, Schleifen, Scheitelpunkte oder Wölbungen zu erkennen aufgrund des Schriftbilds nicht möglich sind, nutzen die Autoren die „region of support“ Methode, die von Teh und Chin [17] erstmals vorgestellt wurde. Sie dient der „dominant point detection“, also dem Auffinden von markanten Stellen aus denen sich anschließend entsprechende Merkmale extrahieren lassen. Die „region of support“ wird an einem Punkt gebildet, indem die Nachbarn des Punktes betrachtet werden. Es werden solange Nachbarpunkte hinzugezogen, bis bestimmte Regeln zutreffen. Die Menge der Nachbarn die nötig sind, um die Bedingungen wahr werden zu lassen, sind die „region of support“ des Punktes. Innerhalb einer „region of support“ wird anschließend der Punkt mit der größten Krümmung als dominierender Punkt selektiert. Danach werden Schleifen, Haken und „short primitives“ gesucht und der Liste der dominanten Punkte hinzugefügt. Letzteres sind kurze Strichzüge am Anfang oder Ende der Eingabe.

Besonders interessant hierbei, auch im Hinblick auf die spätere Erkennung von DEK, ist der Ansatz zur Schleifenerkennung. Die Eingabe wird hierfür in einen 16-direction Freeman chain code umgesetzt, bei dem zwischen jeweils zwei Punkten die Steigung gemessen und in 16 Werten quantisiert wird. Aus einer Eingabe entsteht somit eine Sequenz der Richtungen von einem Punkt der Eingabe zum jeweils nächsten. Diese wird anschließend in Bereiche geteilt, immer an denen Stellen, an denen sich der Wert der Richtung in der Sequenz ändert. Alle Richtungen innerhalb eines Bereichs sind also gleich und können daher als gerade Linie betrachtet werden. Als nächster Schritt werden alle Bereiche mit den jeweilig vorhergehenden Bereichen auf Überlappungen untersucht. Dies kann allein aufgrund der Höhe, Breite und Lage der Bereiche geschehen. Zu guter Letzt werden für alle sich überlappenden Bereiche die Schnittpunkte berechnet. Liegen diese jeweils in beiden Bereichen, dann wird eine Schleife angenommen (siehe Abbildung 13). So können alle geschlossenen Schleifen erkannt werden. Für nicht geschlossene Schleifen, die es bei Requn ebenfalls wie bei DEK nur am Anfang und Ende geben kann, bedienen sich die Autoren eines einfachen Tricks: die Eingabe wird am Anfang in entgegengesetzter Strichrichtung und am Ende in Strichrichtung um ein Stück ergänzt, um eine offene Schleife zu einer geschlossenen werden zu

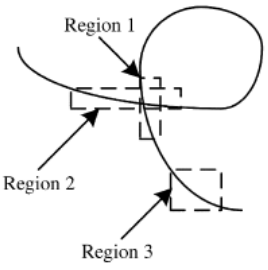



**Abbildung 13: Schleifenerkennung [18].**

lassen (siehe Abbildung 14). Daraufhin kann dann derselbe Algorithmus für geschlossene, als auch für offene Schleifen verwendet werden.

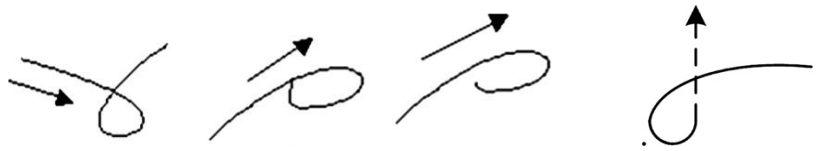


**Abbildung 14: Drei Arten von Schleifen und Erkennung durch Erweiterung [16].**

Im Anschluss daran werden die gefundenen „dominant points“ mithilfe von „polygonal approximation“ nachjustiert. Dabei werden dominante Punkte, die in einer region of support erkannt wurden aber innerhalb einer Schleife liegen, gelöscht. Hierfür werden für jeweils zwei der dominanten Punkte zwei Parameter berechnet. Ist einer der beiden Parameter unter einer bestimmten Schranke, wird der zweite Punkt gelöscht. Ansonsten wird einer der beiden Punkte anhand der Parameter ausgewählt und mit dem nächsten dominanten Punkt verglichen. Dies wird solange wiederholt, bis alle dominanten Punkte den Bedingungen genügen. Die Autoren wählten die Schranken hierbei so, dass eher etwas übersegmentiert wird damit, keine wichtigen Merkmale der Kurve verloren gehen. Aus jedem Segment zwischen zwei dominanten Punkten werden neun Features extrahiert und zwei Backpropagation Netzen zugeführt. Das erste liefert dabei eine Aussage darüber, ob es sich um einen „Stroke“ handelt oder nicht. Nur im Falle eines erkannten Strokes wird das zweite Netz herangezogen, um den genauen Typ aus neun möglichen zu identifizieren. Anschließend werden fälschlich angenommene dominante Punkte, die zu Übersegmentierung führten, gelöscht. Dazu werden, angefangen beim zweiten dominanten Punkt $p_i$, der jeweils sichere Vorgänger $p_{i-1}$ und die jeweils nächsten zwei Punkte $p_{i+1}$ und $p_{i+2}$ betrachtet. Ist der Abstand zwischen $p_i$ und $p_{i+1}$ kleiner als die minimale Stroke Länge und der Abstand zwischen $p_{i-1}$ und $p_{i+2}$ kleiner als eine maximale Stroke Länge, wird einer der beiden Punkte $p_i$ oder $p_{i+1}$ anhand gewisser Regeln gelöscht. Am Ende haben alle dominanten Punkte die geforderten Abstände und können als sichere Segmentierungspunkte angesehen werden. Abbildung 15 stellt den gesamten Ablauf nochmals grafisch dar. Tests zeigten eine korrekte Segmentierung in 99% der Fälle bei einer nicht schräggestellten Schrift, hingegen nur knapp 72% bei einer Schrägstellung von 15° bis 25° und knapp 84% bei circa 45°.

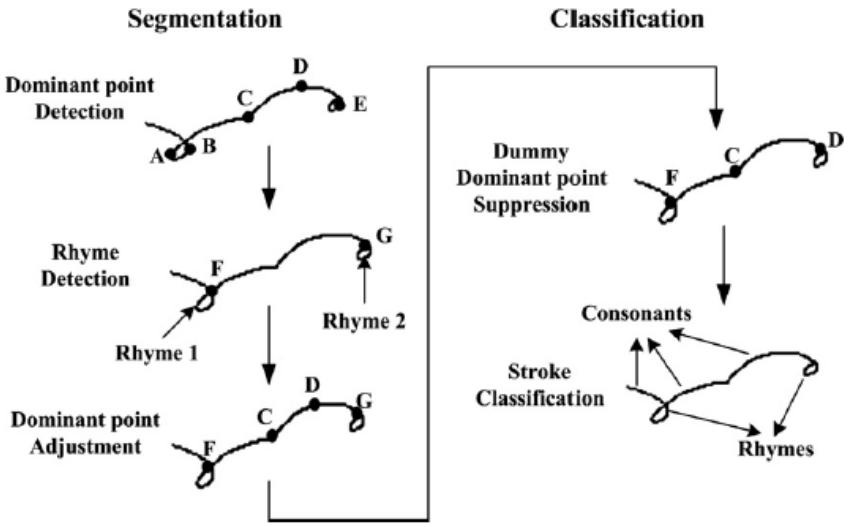



**Abbildung 15: Grafische Darstellung des Prozesses [16].**

Den gleichen Algorithmus wie in [16] verwenden Ma und Leedham zusammen mit Higgins und Htwe in [18] zur Erkennung von Pitman, einer Kurzschrift für die englische Sprache. Requn und Pitman unterscheiden sich dabei nur geringfügig. Für Pitman wurden lediglich einige Anpassungen der Parameter vorgenommen und ein zusätzlicher Schritt für die Erkennung von Vokalen und Diphthongen hinzugefügt. Diese werden in Pitman durch kurze Striche oder Punkte entlang der Konsonantenzeichen angedeutet. Die Erkennung ist dabei recht einfach gelöst. Wenn die Eingabe in ein 1,5mm² großes Raster passt und weniger als 100ms dauerte, dann wird ein Punkt erkannt. Liegen alle Punkte einer Eingabe im Abstand von 0,5mm zu der Linie, die Anfangs- und Endpunkt verbindet, so wird ein Strich angenommen. Für Diphthonge wird die Unterscheidung anhand eines 8-direction Freeman chain codes der Eingabe getroffen. Die Position eines Vokalzeichens bestimmt dabei die genaue Bedeutung. Hierzu wird der dominante Punkt K mit dem geringsten euklidischen Abstand zu einem Vokal oder Diphthong gesucht. Anschließend wird der Punkt H zwischen den dominanten Punkten K-1 und K+1 mit dem geringsten Abstand berechnet. Anschließend bestimmt die Position von K und H die genaue Art des Vokals oder Diphthongs. Außerdem gehen die Autoren auf das besonders schwere Erkennen von sogenannten „smoothed junctions“ ein. Dies sind Verbindungen, die quasi gerade verlaufen, also von der „region of support“ Methode nicht und laut Autoren auch mit anderen mathematischen Mitteln nur schwer erfasst werden können. Daher empfehlen sie, diese speziell zu kennzeichnen um die Maschinenlesbarkeit zu erhöhen. Durch einen Kreis von beliebiger Größe in der Nähe der smoothed junction könnte diese einfacher als dominanter Punkt erkannt und weiterverarbeitet werden. Tests des Systems mit drei Schreibern führten für „normales“ Pitman (ohne die eingeführte Regel) zu einer durchschnittlichen Erkennungsrate von 75,33%. Bei einem zweiten Experiment, bei dem nur die neu eingeführte Regel für die Maschinenlesbarkeit der smoothed junctions evaluiert werden sollte, stellten die Autoren eine um 55,88% (von 37,53% zu 93,41%) höhere Erkennungsrate mit der neuen Regel fest. Die Schreiber benötigten für die Eingabe dabei 14,42% mehr Zeit als ohne die Regel.

Eine Segmentierungstechnik für Sindhi anhand von Height Profile Vectors (HPV) zeigen Shaikh et al. in [19]. Sindhi ist eine arabische Schrift, die kursiv auf einer gedachten Grundlinie (Baseline) geschrieben wird. Die Autoren weisen auf die Ähnlichkeit zu anderen arabischen Schriften hin und somit auf die Übertragbarkeit ihres Algorithmus auf andere arabische Schriften. Da es sich um eine OCR Aufgabe handelt, werden zunächst Preprocessing Techniken angewendet und die Grundlinie erkannt. Für die Erkennung von PSPs werden zunächst die HPVs gebildet. Der Wert an der Stelle i im HPV wird bestimmt durch den größten Abstand eines Pixels in einer ein Pixel breiten Spalte an der Stelle i zu der erkannten Grundlinie. Bei einem Wert von 0 für i im HPV liegt der Punkt i also genau auf der Grundline. Weisen eine bestimmte Anzahl von aufeinanderfolgenden Werten des HPV einen Wert von 0 auf, wird hier ein PSP angenommen. Die Anzahl n der angenommenen PSPs bestimmt damit grob die Anzahl der Buchstaben. Ist $n > 1$, wird jeweils der letzte gefunden PSP einer weiteren Prüfung unterzogen, um sechs Spezialfälle zu unterscheiden, denn scheinbar können bei der Segmentierung von Sindhi mit dem genannten Algorithmus nur hier Unsicherheiten auftreten. Leider werden auch in dieser Arbeit nur sehr unvollständig formulierte Testergebnisse geliefert. Der Algorithmus scheint bis auf zwei Spezialfälle zu funktionieren. Wobei einer dieser Fälle wohl nicht als Fehler angesehen werden kann. Für den anderen schlagen die Autoren eine genauere Betrachtung in der nächsten Phase der Erkennung vor.

Eine OCR Methode um Jawi (ebenfalls eine arabische Schrift) in gescannten Dokumenten zu segmentieren, haben Razak et al. in [20] gezeigt. Dabei legen die Autoren vor allem Wert auf eine Implementierung, die leicht in Hardware gegossen werden kann. Dazu wird zunächst ein gescanntes Dokument in Textzeilen zerlegt. Für jede Zeile wird dann ein Histogramm gebildet, indem die schwarzen Pixel einer Spalte gezählt werden. Der größte Wert des Histogramms wird für die Normalisierung des Histogramms in der nächsten Stufe gespeichert und die Gradienten für jede Stelle des Histogramms gebildet. Bei der Normalisierung werden nun die aufeinanderfolgenden negativen Gradienten gezählt. Übersteigt die Anzahl negativer Gradienten die vorher festgelegte

Schwelle, werden alle Gradienten dieses Blocks von negativen in positive konvertiert. Die normalisierten Gradienten des Histogramms werden nun verglichen. Findet ein Übergang von negativ nach positiv bei zwei aufeinanderfolgenden Werten statt, wird ein PSP gesetzt. Danach werden alle gefunden PSPs durchlaufen und überprüft, ob diese innerhalb eines bestimmten Bereichs (sliding window) liegen. Wenn ja, wird an der Stelle des PSPs segmentiert, wenn nicht, wird am Ende des Bereichs segmentiert. Die Breite des Bereichs ergibt sich aus der berechneten Höhe der aktuellen Textzeile. Abbildung 16 veranschaulicht den gesamten Prozess nochmals. Probleme soll dieser Ansatz nur bei sich überlappenden Buchstaben haben. Angewendet auf ein in Jawi geschriebenes Dokument aus dem 17. Jahrhundert erreichte der Algorithmus eine 98% Genauigkeit bei der Segmentierung der Buchstaben.

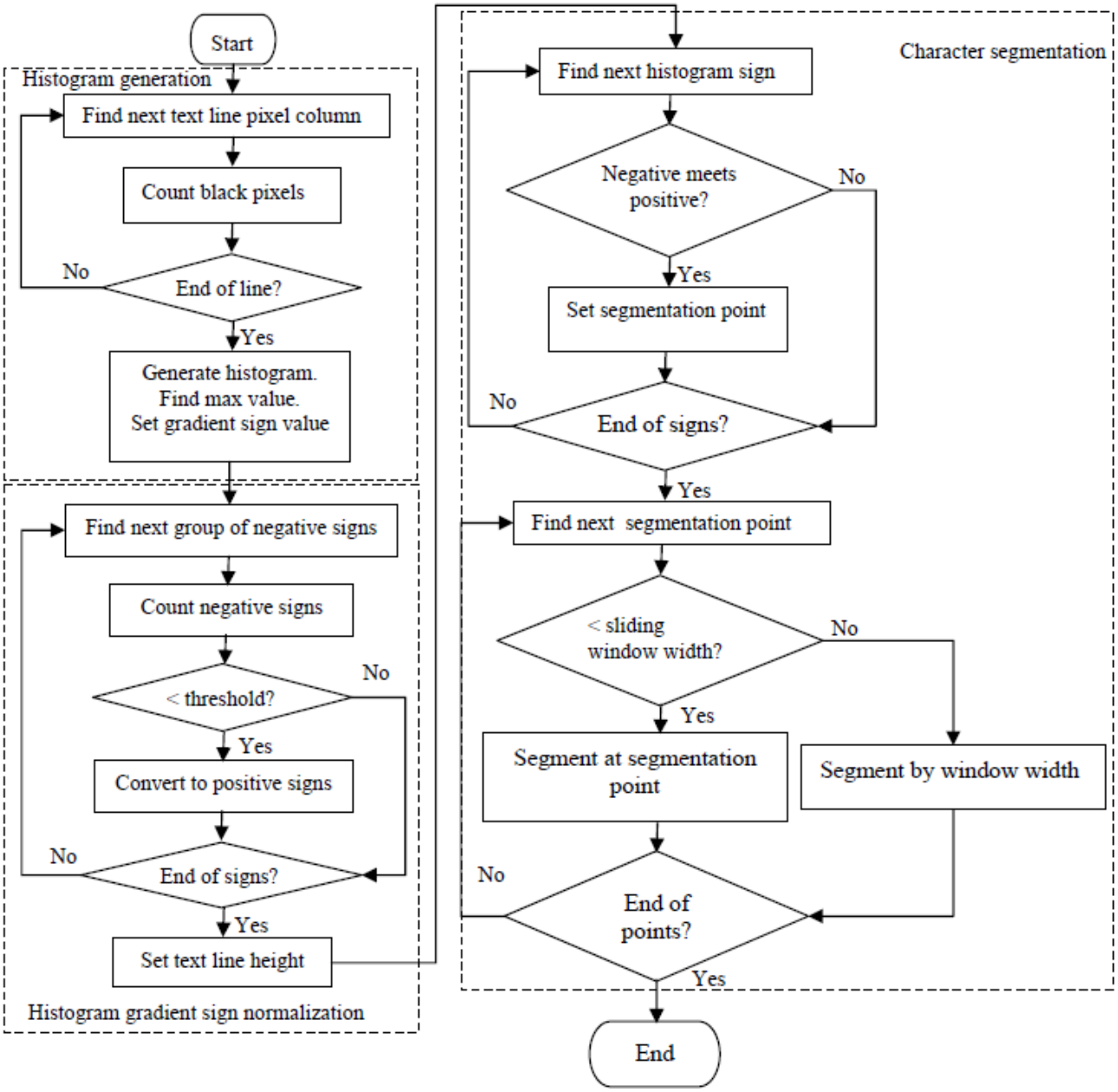



**Abbildung 16: Flow Chart der Buchstabensegmentierung [20].**

Rehman et al. [21] vergleichen einen impliziten mit einem expliziten Ansatz zur Segmentierung von Schriften, welche auf dem lateinischen Alphabet beruhen. Ideale PSPs liegen bei kursiven Schreibstilen dabei auf den Ligaturen (Verbindungen zwischen zwei Buchstaben). Beim impliziten

Ansatz ist die Segmentierung eng mit der Erkennung verbunden. Alle möglichen Segmente werden dabei dem Erkennungsalgorithmus übergeben und dieser entscheidet, ob evtl. mehrere Segmente zusammengehören. Trotzdem ist es dabei wichtig, die Segmente sorgfältig zu wählen. Eine Übersegmentierung ist dabei besser als zu wenig Segmente, jedoch bedeutet jedes überschüssige Segment zusätzliche Berechnungsdauer, um falsche PSPs zu verwerfen. Außerdem kann es bei Übersegmentierung dazu kommen, dass Teile eines Buchstabens als eigenständiger Buchstabe erkannt werden. Daher wählten die Autoren einen heuristischen Ansatz in der ersten Phase, der das Wort übersegmentiert aber gleichzeitig unnötige PSPs versucht zu eliminieren. Dazu wird zuerst das Wort in Segmente der Breite x = h / 18 geteilt, wobei h die Höhe des Wortes angibt und 18 einen empirisch festgelegten Wert darstellt. Anschließend werden Schleifen durch Schnittpunkte von zwei Linien erkannt und die Teilung im Bereich der Schleifen aufgehoben. Im nächsten Schritt werden alle übrigen Segmente, die durch dieselbe Linie durchlaufen werden und in einem Abstand von x zueinander stehen, verworfen und zu einem Segment verbunden. Zuletzt werden alle dann noch übrigen Segmente zusammengefasst, die im Abstand von x zueinander stehen. Die Autoren können zeigen, dass aufgrund der getroffenen Hypothesen, ein Buchstabe maximal in zwei Teile geteilt wird.

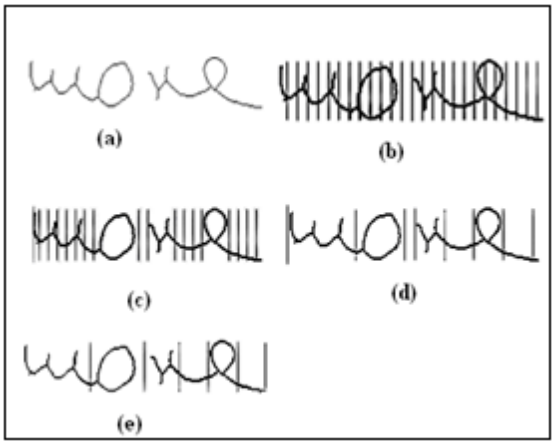



**Abbildung 17: Impliziete Segmentierung nach [21].**

Beim expliziten Ansatz werden PSPs zunächst überall dort gesetzt, wo das vertikale Histogramm einen Wert von 0 oder 1 aufweist. Anschließend werden alle PSPs verworfen, die in ihrer näheren Nachbarschaft bereits einen PSP besitzen. Alle dann noch übrigen PSPs werden von einem trainierten neuronalen Netz anhand von Merkmalen der Dichtefunktion validiert.

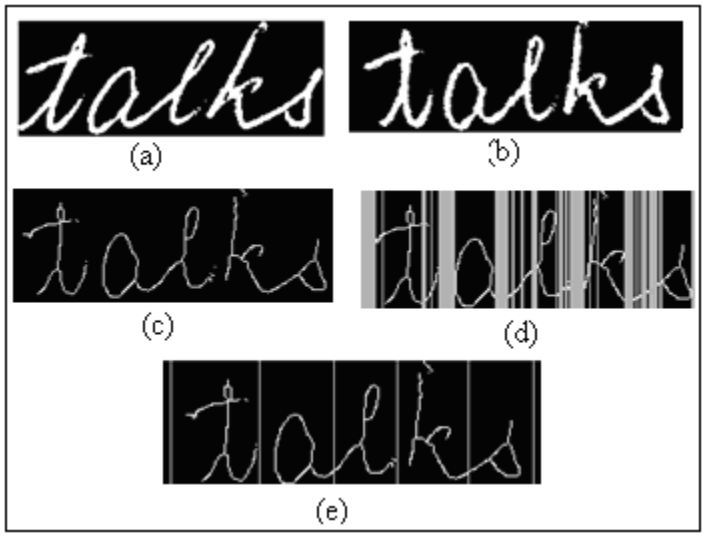



**Abbildung 18: Explizite Segmentierung nach [21].**

Für die Erkennung der mit beiden Ansätzen gefundenen Segmente extrahierten die Autoren die Transition- und Profilprojektionsmerkmale. Ersteres wird anhand der Übergänge von Vodergrund- zu Hintergrundpixeln und umgekehrt berechnet, letzteres teilt das Segment in zwei gleiche Bereiche und berechnet die Distanz der oberen und unteren Projektion von der Mittellinie. Diese Features

wurden einem neuronalen Netz zugeführt und hinterher noch mit einem Lexikon abgeglichen. Der Test auf 350 Wörter einer Datenbank ergab für impliziten Ansatz eine Erkennungsrate von 79,23% und für die explizite Variante 80,91%.

Viele Arbeiten, auch basierend auf anderen Kurzschriftensystemen, liefern bereits gute bis sehr gute Ergebnisse. Wobei bei der Vielfalt der Arbeiten es festzustellen gilt, dass gerade die Segmentierung immer eine Aufgabe darstellt, die stark auf die zu segmentierende Schrift angepasst werden muss. Ohne zusätzliches Wissen über die Besonderheiten einer Schrift ist keine vernünftige Segmentierung möglich. Das Studieren der Publikationen zeigt bereits jetzt deutlich, dass keine der Methoden einfach auf DEK anzuwenden ist, sondern lediglich vereinzelte Impulse für das Lösen von Teilproblemen bei der Segmentierung von DEK liefern können. Keine der gefunden Schriften gleicht dabei der DEK in besonderem Maße. Eine genauere Analyse diesbezüglich wird in Kapitel 4 gezeigt.

## 3.3. Kontexterkennung

Die Kontexterkennung ist zwar nicht Bestandteil dieser Arbeit, dennoch ist es für das Erkennen von DEK unerlässlich, den Kontext, in dem gewisse Zeichen auftreten zu betrachten, um zwischen mehreren Interpretationen eines Zeichens die richtige Alternative wählen zu können (siehe Kapitel 2.1). Untersuchungen in diese Richtung wurden unternommen, um bei der in Kapitel 4 gewählten Methode sicher zu gehen, dass eine spätere Erweiterung um Kontextsensitivität problemlos möglich ist. Daher sollen hier kurz Publikationen in diesem Bereich vorgestellt werden.

Hanson et al. zeigen bereits 1976 einen vom restlichen Erkennungssystem unabhängigen Kontext Postprocessor (CPP), basierend auf binären n-Grammen [22]. In ihrer Arbeit analysieren sie bisherige Konzepte, die teilweise stark in den Erkennungsprozess eingebunden sind. Diese seien jedoch sehr komplex und nur theoretisch interessant, doch in der praktischen Anwendung bleiben die meisten den Erwartungen hinterher. Daher sollen Erkennungssysteme mit klar getrennten Subsystemen nicht nur einfacher zu implementieren, sondern auch leistungsfähiger sein, da Rückkopplungen aus einer späteren Phase leichter möglich sind als in komplex verwobenen Systemen. Die Speicherung der Kontextinformationen erfolgt in binären n-Grammen. Diese werden anhand von binären Trigrammen erläutert. Ein Trippel von Buchstabenpositionen (i,j,k) spezifizieren ein 26 x 26 x 26 (im Falle eines lateinischen Alphabets) Array $D_{ijk,}$ das stellungsabhängiges binäres Trigramm (positional binary trigram) genannt wird. Der (l,m,n)-te Eintrag von $D_{ijk}$ ist 1 genau dann, wenn es ein Wort gibt, welches die Buchstaben l, m, n an der i-ten, j-ten und k-ten Position besitzt, ansonsten 0. Durch diese Datenstruktur kann sehr schnell abgefragt werden, ob eine Buchstabenkombination möglich ist. Der Satz aller $\binom{m}{3}$ stellungsabhängiget binärer Trigramme kann dazu verwendet werden, Fehler in Wörtern mit m Buchstaben zu erkennen und in bestimmten Fällen sogar zu beheben. Da ein Wort immer Buchstabe für Buchstabe validiert wird, wird beim Auftreten eines Fehlers davon ausgegangen, dass der zuletzt hinzugenommene Buchstabe der Fehlerhafte ist. Anhand der beiden korrekten Buchstaben gibt es im Satz der Trigramme 26 Kombinationen, die 0 oder 1 sein können. Ist genau eine dieser Kombinationen 1, so kann der Buchstabe dieser Kombination angenommen werden. Gibt es mehrere Möglichkeiten, wird das Wort verworfen. Tests auf Wörtern mit sechs Buchstaben zeigten, dass der CPP 99% der Wortfehler erkennt und 50% davon korrigieren kann. Allerdings produzierte das Verfahren auch neue Fehler, bei denen auch Wörter korrigiert wurden, in denen weniger Fehler gefunden wurden als diese eigentlich enthielten. Das Verhältnis der höheren Korrekturen zu den erhöhten Fehlern lag jedoch bei 64:1 und 44:1 für Wörter mit einem bzw. zwei Fehlern. Die gesamte Wortfehlerrate lag bei weniger als 1% und die der Buchstabenfehler betrug 0,23%. Diese beeindruckenden Ergebnisse erzielten die Autoren allerdings auf Kosten von 21% verworfenen Wörtern mit drei oder mehr Fehlern. Sie begründen dies aber damit, dass auch der Mensch Probleme hat, ein sechsstelliges Wort mit drei Fehlern zu korrigieren.

Goshtasby und Ehrich schlagen ein System auf Basis von Relaxation Prozessen vor [23]. Dies habe den Vorteil, das nicht nur direkte Nachbarn eines Buchstabens, sondern prinzipiell jeder Buchstabe eines Wortes Einfluss auf dessen Erkennung haben kann. Der Prozess nutzt dabei Informationen über Buchstaben, deren Erkennung sehr sicher ist, für die Korrektur von unsicheren Buchstaben. Der Ansatz benötigt also einen Classifier, der eine Liste möglicher Klassen mit deren Wahrscheinlichkeiten für jeden Buchstaben eines Wortes liefert. Die Autoren definieren ihren CPP auf Basis der Relaxation Prozessen von Rosenfeld et al. [24] wie folgt: Sei $P_i^{(0)}(\lambda)$ die Anfangswahrscheinlichkeit mit $P_i^{(0)}(\lambda) \equiv P(\theta_i = \lambda | X_i)$ dafür, dass ein Buchstabe $X_i$ dem Label λ entspricht und $r_{ij}(\lambda, \lambda')$ die Wahrscheinlichkeit für das Label $\lambda'$ des Buchstaben $X_j$ wenn das Label $\lambda$ des Buchstaben $X_i$ bekannt ist, dann entspricht $r_{ij}(\lambda, \lambda') \equiv P(\theta_j = \lambda' | \theta_i = \lambda)$. Verwendet man nur die direkten Nachbarn eines Buchstaben, um die Wahrscheinlichkeiten für ein Label zu verfeinern, dann ist $j = i \pm 1$. Sollen zu jeder Seite $N$ Nachbarn verwendet werden, entspricht $j \pm t, t = 1, \dots, N$. Die Faktoren für den Beitrag des Relaxation Process werden definiert durch

$$q_i^{(k)}(\lambda) = \sum_j \delta_{ij} r_{ij}(\lambda, \lambda') P_j^{(k)}(\lambda')$$

wobei $\lambda' = \{\lambda_i | P_j(\lambda_i) > P_j(\lambda_l), \forall l \neq i\}$. $\delta_{ij} \in [0,1]$ sind Gewichte, sodass gilt $\sum_j \delta_{ij} = 1$. Mit ihnen können z.B. nahe Nachbarn eines Buchstaben stärker berücksichtigt werden als weiter entfernte. Damit gibt $q_i^{(k)}(\lambda)$ den Einfluss der Nachbarlabels für das Label $\lambda$ des Buchstabens an Stelle $i$ nach $k$ Iterationen an. Die Anzahl der Nachbarn wird dabei durch $j$ angegeben, wie stark das Label von Buchstabe $i$ durch das Label von Buchstabe $j$ beeinflusst wird durch $\delta_{ij}$. $\lambda'$ steht dabei für das Label mit der höchsten Wahrscheinlichkeit an Stelle $j$. Die Autoren wenden diese Definitionen anschließend auf die Relaxation Formel von Rosenfeld et al. an und erhalten

$$P_i^{(k+1)}(\lambda) = \frac{P_i^{(k)}(\lambda)\, q_i^{(k)}(\lambda)}{\sum_\lambda [P_i^{(k)}(\lambda)\, q_i^{(k)}(\lambda)]}, k = 1,2, \dots$$

$P_i^{(k+1)}(\lambda)$ gibt damit die Wahrscheinlichkeit für das Label $\lambda$ des Buchstabens an Stelle $i$ nach $(k+1)$ Iterationen an, basierend auf der Wahrscheinlichkeit, dass es das Label $\lambda$ nach $k$ Iterationen hatte und dem Einfluss der Labels der Nachbarbuchstaben. Der Nenner dient der Normalisierung und sorgt dafür, dass die Ausgabe Wahrscheinlichkeiten darstellen. Wenn der Prozess konvergiert, erreicht genau ein Label jedes Buchstabens eine 100%ige Wahrscheinlichkeit. In diesem Fall wurde ein Wort eindeutig erkannt. Üblicherweise sei es aber so, dass sich der Status eines Labels nicht mehr verändert, sollte die Wahrscheinlichkeit über 95% betragen. Für den praktischen Einsatz schlagen die Autoren daher eine Schranke vor. Sollte die höchste Wahrscheinlichkeit für ein Label diese Schranke übersteigen, sollte der Prozess gestoppt werden. Leider liefern die Autoren keine genauen Daten ihrer Testergebnisse, sondern lediglich die Aussage, dass der CPP viele Fehler des Classifiers korrigieren konnte.

Wells et al. bemängeln in ihrer 1990 erschienenen Arbeit [25], dass die bis dahin üblichen Methoden, Buchstaben anhand der Nachbarbuchstaben zu erkennen, eine gewisse Verbesserung bringen, aber auch Fehler produzieren können. Denn es sei damit nicht sichergestellt, dass die Wörter, die erkannt werden, auch tatsächlich Wörter der spezifischen Sprache sind. Sie empfehlen daher den Einsatz eines Wörterbuchs. Nur dadurch würden ausschließlich Wörter akzeptiert, welche es auch tatsächlich gibt, auch wenn die Reject Rate der nicht gültigen Wörter höher sei als bei anderen Ansätzen. Das von Forschern angesprochene Problem beim Einsatz von Wörterbüchern, dass diese bei einem ausreichend großen Vokabular (50.000 Wörter und mehr) zu viel Speicher und Suchzeit benötigen würden, ist mittlerweile gelöst, da Speicher immer günstiger und Prozessoren immer leistungsfähiger werden. Um die Suche dennoch so effizient und speicherschonend wie möglich zu machen, stellen die Autoren eine neuartige Datenstruktur für das Speichern von

Wörterbüchern vor. Hash Tabellen und Baumstrukturen würden diesen Ansprüchen genügen. Doch bei Hash Tabellen kämen Kollisionen bei großem Vokabular häufiger vor und es wäre nur möglich, ganze Wörter abzugleichen und nicht nur Teile eines Worts wie z.B. den Anfang. Bei Schreibfehlern würde somit ein Wort komplett verworfen und es gäbe keine Möglichkeit, Teile des Wortes zu validieren und dadurch Fehler des Recognizers zu korrigieren. Außerdem kann ein Wörterbuch auf Basis einer Baumstruktur bereits bei der Erkennung verwendet werden, nicht mögliche Buchstabenkombinationen auszuschließen, da eine Validierung Buchstabe für Buchstabe möglich ist. Daher haben sich die Autoren für die Baumstruktur entschieden. Bäume mit beliebiger Verzweigung können schneller durchsucht werden als Binärbäume, weshalb sie für ihren Fall einen Baum mit 26-facher Verzweigung (ein Zweig für jeden Buchstaben des Alphabets) wählten. Um jedoch unnötige Pfade nicht speichern und später auch durchsuchen zu müssen, verbesserten sie die normale Implementierung wie folgt: Ein Knoten speichert einen 32 Bit Integerwert und einen Zeiger auf den Anfang eines Arrays. Dabei geben die ersten 26 Bit des Integerwerts an, ob ab diesem Knoten eine Verbindung zu den jeweiligen Buchstaben des Alphabets führen (siehe Abbildung 19). Beginnt der Wert beispielsweise mit 1101… heißt das also, dass auf den Buchstaben des aktuellen Knotens ein „a“, „b“ und „d“ folgen können, aber kein „c“. Innerhalb des Arrays, auf das der Zeiger der Datenstruktur zeigt, kann an erster Position nun der Knoten für den Fall des Buchstaben „a“ an nächster Stelle gefunden werden. An zweiter Position der im Fall des Buchstaben „b“ und an dritter der für „d“. Ein Zeiger auf „c“ als nächster Buchstabe ist nicht notwendig, da dieser bereits ausgeschlossen wurde. Dadurch wird kein Platz für den Zeiger eines Pfades reserviert, den es nicht geben kann. An welcher Position im Array der gesuchte Zeiger für den nächsten Buchstaben steht, kann durch Zählen der Einser im Integer Wert bis zum gewünschten Buchstaben erreicht werden. Das letzte der 32 Bit gibt dabei noch an, ob der aktuelle Buchstabe das Ende eines Wortes darstellt. Es muss also nur der Baum mit der gesuchten Buchstabenkombination durchlaufen werden und das Wortende geprüft werden, um die Korrektheit eines Wortes festzustellen. Die übrigen 5 Bit könnten für weitere Optimierungen genutzt werden. Der Buchstabe zu dem ein Knoten gehört, muss dabei nicht gespeichert werden, da sich dieser aus dem Pfad innerhalb des Baums ergibt.

Tests im Vergleich zu einer herkömmlichen Implementierung eines 26-fach verzweigenden Baumes zeigten bei einem Wörterbuch mit knapp 15.000 Wörtern, dass die Suchzeit für einen Satz etwa doppelt so lange dauert (6 s zu 12 s), dafür aber nur ein Zehntel des Speichers benötigt (4.9 MB zu 0.4 MB). Interessanterweise war die Methode bei einem Wörterbuch mit 60.000 Wörtern wiederum doppelt so schnell (33 s zu 16 s) und benötigte weiterhin nur etwa 10% des Speichers (15,1 MB zu 1,3 MB). Da ein Recognizer normalerweise für jede Position eines Wortes eine Liste der möglichen Buchstaben und deren Wahrscheinlichkeit liefert, muss jede Kombination dieser Wörter mit dem Wörterbuch abgeglichen werden und die Wahrscheinlichkeit für das gesamte Wort berechnet werden. Hierbei verwenden die Autoren den Mittelwert über die vom Recognizer gelieferten Wahrscheinlichkeiten aller Buchstaben eines im Wörterbuch vorhandenen Wortes. Der CPP liefert anschließend eine anhand der Wahrscheinlichkeiten sortierte Liste aller möglichen Wörter.

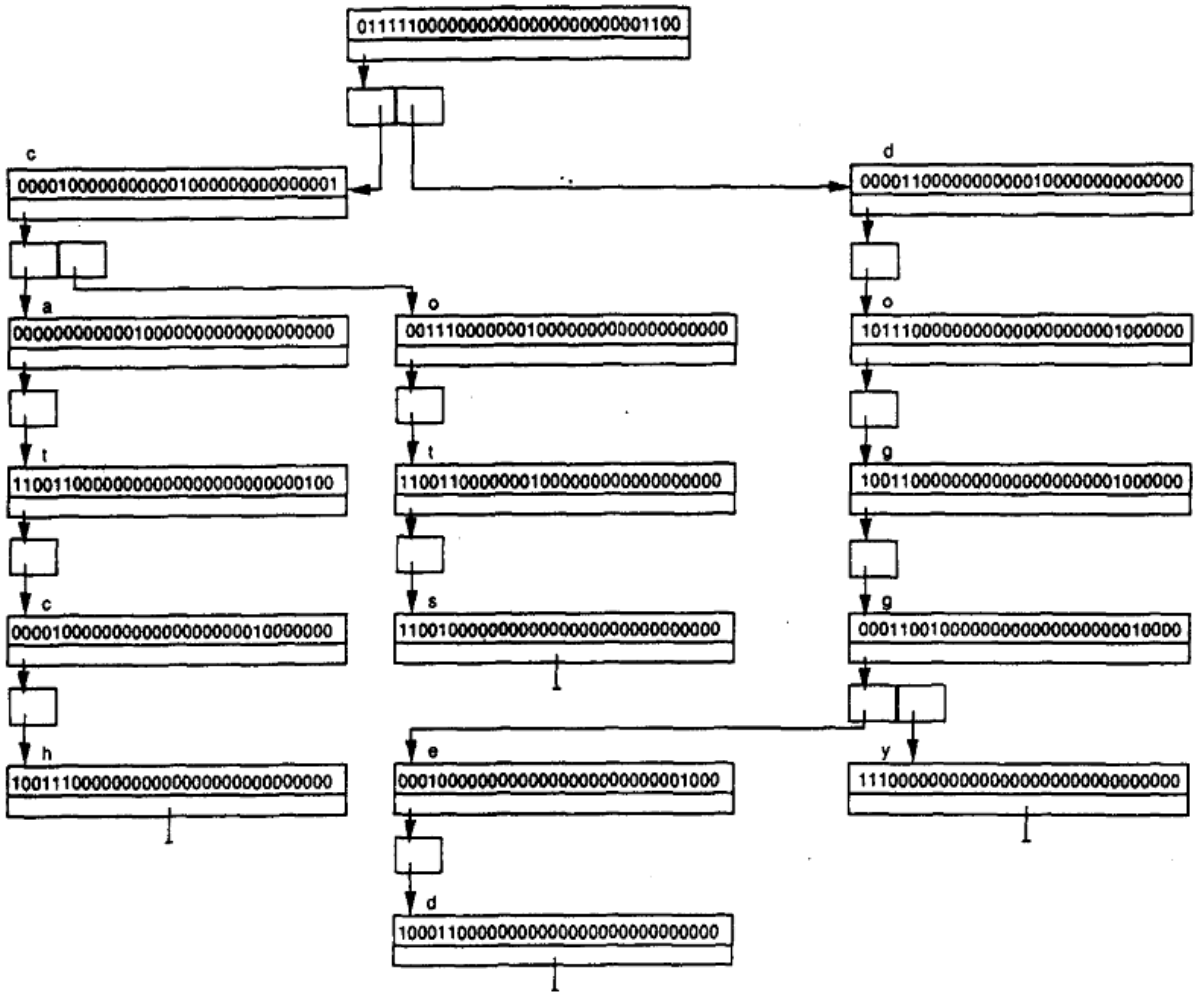



**Abbildung 19: Baum mit reduziertem Speicherverbrauch nach [25].**

Einen vielversprechenden Ansatz liefern Iwayama und Ishigaki mit einem Adaptiven Context Processor (ACP) [26]. Die Methode baut auf einem Classifier auf, der eine Liste der besten zutreffenden Kandidaten für die Eingabe liefert. Als effektive Methode für die Kontexterkennung setzen die Autoren auf ein Context Dictionary (CD). Damit die Erkennung gute Ergebnisse liefern kann, muss das CD eine gute Abdeckung einer großen Anzahl an Eingaben eines Schreibers aufweisen. Die Bandbreite der enthaltenen Eingaben eines CD sei dabei weniger entscheidend. Es ist also wichtiger, im CD Eingaben vorzufinden, die von einem Schreiber regelmäßig eingegeben werden. Dies ist nur mit einem adaptiven System möglich, da nur so neue Eingaben hinzugefügt werden können. Daher haben die Autoren ein Adaptive Context Dictionary (ACD) entwickelt, das während der Laufzeit gebildet wird. Damit das Wörterbuch kompakt bleibt, wird jede gespeicherte Eingabe mit einem Zeitstempel versehen. Ein Least Recently Used Algorithmus löscht regelmäßig alle Eingaben, die nach einer bestimmten Zeit nicht mehr verwendet wurden. Beim Postprocessing wird versucht, eine passende Eingabe im ACD zu finden, ist hier kein passender Kandidat vorhanden, wird eine normale Kontexterkennung, basierend auf einem allgemeinen CD auf Basis von n-Grammen, durchgeführt. Dabei muss der Schreiber nichts beachten, denn es werden automatisch Eingaben zum ACD hinzugefügt, die der Benutzer validiert. Eine Validierung erfolgt automatisch, wenn der Benutzer die Eingabe nach der Erkennung nicht verändert. Ist die Ausgabe falsch, wird der Benutzer diese verändern, indem er mit dem System interagiert. Eine vom Benutzer korrigierte Ausgabe ist somit ebenfalls valide.

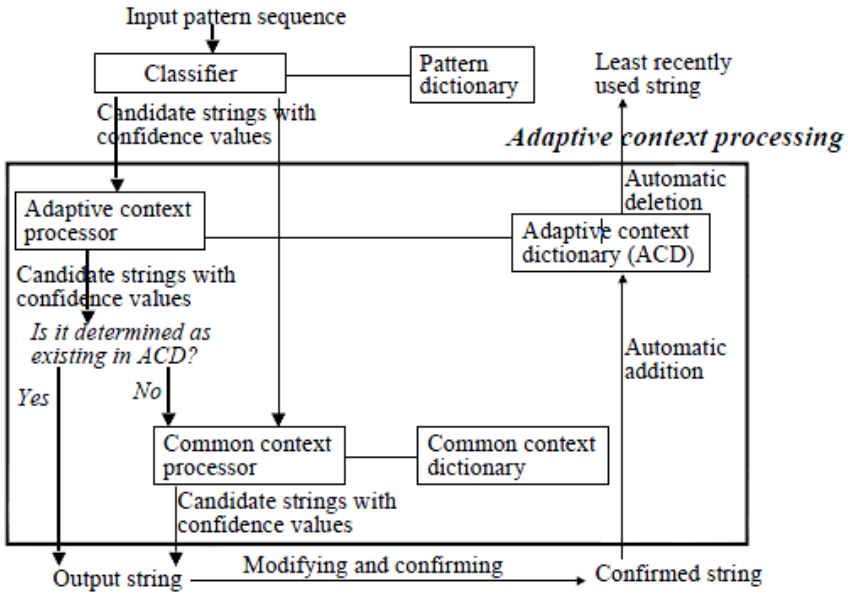



**Abbildung 20: Adaptive Context Processor ACP nach [26].**

Der Classifier der Autoren liefert eine First Hit Rate von 88%, also der Wahrscheinlichkeit, dass der vom Classifier als bester Kandidat gelieferte String auch der Eingabe entspricht. Die Wahrscheinlichkeit, dass die Eingabe unter den besten 20 Kandidaten enthalten ist liegt jedoch bei 99%. Getestet wurde die Integration des ACP auf 400 Kalendereinträge von jeweils 14 Benutzern innerhalb von drei Monaten. Drei Testreihen wurden gestartet. Bei der ersten war das ACD leer, bei der zweiten war das ACD bereits etwas gefüllt und bei der dritten Reihe wurden nur Eingaben getestet, die sicher im ACD vorhanden waren. Hierbei erreichte der Classifier knapp 81% bei der First Hit Rate und etwa 97% bei der Nominatig Hit Rate jeweils ohne ACP in allen drei Testreihen. Mit ACP lag die First Hit Rate in der ersten Testreihe, also nur auf dem allgemeinen CD, bereits bei 86%. In der zweiten und dritten Testreihe schließlich sogar bei 89% bzw. 95% und wurde somit um 15% im Vergleich zur First Hit Rate ohne ACP verbessert.

Htwe et al., die bereits viele Arbeiten für die Erkennung der Kurzschrift Pitman geliefert haben, zeigen ein System für das Transkribieren von Pitman unter Verwendung von Worthäufigkeiten und kontextuellem Wissen [27]. Dabei stellen sie fest, dass die Transkription aufgrund von umfangreichem Einsatz von kontextuellem Wissen und Erinnerungen des ursprünglichen Schreibers eine schwierige Aufgabe darstellt. Ihr Ansatz, das kontextuelle Wissen zu verwenden, basiert hierbei darauf, dass ein Mensch Texte trotzdem lesen kann, auch wenn diese falsch geschrieben sind. Dabei haben Forscher herausgefunden, dass es vor allem auf den ersten und letzten Buchstaben eines Satzes ankommt. Den Satz „Wornlgy seplled Egnlish words are sitll leiglbe as lnog as the frist and lsat ltteers are crroect“, kann normalerweise ohne Schwierigkeiten gelesen werden. Weiter fanden sie in Tests heraus, dass vor allem Wortanfang und -ende von Pitman Schreibern oft sehr deutlich geschrieben werden. Eine ihrer Methoden arbeitet daher vor allem mit den vom Recognizer gelieferten Buchstaben am Anfang und am Ende und versucht damit Wörter in einem Wörterbuch zu finden, die am besten mit Wortlänge und den restlichen Buchstaben korrespondieren. Als weitere Möglichkeit nennen sie den Einsatz von prägnanten Segmenten innerhalb eines Wortes. Dies sind Segmente, die vom Recognizer mit hoher Wahrscheinlichkeit erkannt wurden. Dieser Gedanke ähnelt den bisher vorgestellten Systemen, die die mit hohen Wahrscheinlichkeiten erkannten Buchstaben verwenden um solche mit niedriger Wahrscheinlichkeit zu verifizieren oder zu korrigieren. Weiter empfehlen sie den Einsatz eines Wörterbuchs mit den 5.000 häufigsten Wörtern gepaart mit deren Häufigkeit in der englischen Sprache. Für sehr themenbezogene Anwendungsgebiete (wie z.B. in der Medizin) könnte das

Wörterbuch entsprechend auf die Anwendung ausgelegt werden. Die Entscheidung für ein Wort wird dann anhand der vom Recognizer gelieferten Wahrscheinlichkeit und anhand der Häufigkeit getroffen. Der Score für ein Wort müsse dabei so ausgelegt sein, dass Wörter, die mit hoher Wahrscheinlichkeit vom Recognizer geliefert werden, nicht durch die Worthäufigkeit ähnlicher Wörter korrigiert werden, sondern diese nur bei Buchstaben mit geringer Wahrscheinlichkeit greifen. Auf ein Wörterbuch mit den 5.000 häufigsten Wörtern erreichte ihr Ansatz eine Genauigkeit von 94%, wobei die Eingabe für den Test nicht von einem Recognizer geliefert wurde, sondern anhand eines phonetischen Lexikons simuliert wurde. Zur Art der Simulation machen die Autoren keine Aussage, wodurch die Aussagekraft der 94% begrenzt ist, da nicht hervorgeht, mit welcher Wahrscheinlichkeit die einzelnen Buchstaben der Wörter simuliert wurden.

Alle gefundenen und vorgestellten Ansätze basieren immer nur auf dem kontextuellen Wissen einzelner Wörter. Also entweder durch die Suche in einem Wörterbuch oder durch die Wahrscheinlichkeiten der Buchstaben eines Wortes zueinander. Im Rahmen der Arbeit wurde leider kein Ansatz für die Kontexterkennung innerhalb einer Phrase oder eines Satzes gefunden. Jedoch ist es erfreulich zu sehen, dass es effiziente Ansätze gibt, die eventuelle Fehler, die bei der Segmentierung oder bei der Erkennung gemacht wurden, korrigieren können um die gesamte Erkennungsrate zu verbessern. Weiter hat das Studium der Arbeiten gezeigt, dass die Kontexterkennung i.d.R. als eigenständiger Postprozess abläuft und daher bei der Entwicklung der Methode nicht auf die Kontexterkennung Rücksicht genommen werden muss. Ein Recognizer, der eine Liste von Buchstaben gepaart mit deren Wahrscheinlichkeit ausgibt, ist ausreichend.

## 3.4. Text2DEK

Wie bereits erwähnt, konnte keine Arbeit gefunden werden, die sich mit der Handschriftenerkennung der DEK (weder offline noch online) beschäftigt. Jedoch beschreibt die Arbeit von Herrn Sarman den umgekehrten Weg [28]. Er analysiert darin das Schriftbild der DEK, um aus LATEX heraus mittels METAFONT Verkehrsschrift-Stenogramme zu erzeugen.

Dazu wurden zunächst die elementaren Zeichen für Konsonanten(folgen), Präfixe und Suffixe aus der Wiener Urkunde [2] digitalisiert und gerade gestellt. Anschließend wurden je Zeichen „die Koordinaten und Tangenten in Anfangs- und Endpunkten, sowie in Punkten der lokalen Krümmungsextrema erfasst und in METAFONT-Pfaden (paths) verzeichnet“. Genau genommen wurden die Zeichen mittels Hermetischer Interpolation als krümmungsstetige Bezier-Spline-Kurven modelliert, d.h. aus Anfangs- und Endpunkten sowie der Tangentensteigung in diesen Punkten wurden die Stützpunkte für die Bezier-Kurven berechnet. Da sich einzelne Zeichen nicht durch eine einzige Bezier-Kurve darstellen lassen, wurden die Zeichen in mehreren Teilen modelliert, den so genannten Splines.

Anhand der möglichen Strichrichtung der Tangente (aufwärts/abwärts), sowie der positiven, negativen und verschwindenden Krümmung an Anfangs- und Endpunkten wurde eine Klassifizierung der Zeichen vorgenommen (siehe Abbildung 21). Diese stellt eine Verfeinerung der üblichen Zeichenunterteilung von Kurzschriften dar, anhand deren Enden in Geradeausläufe, Rechtsschräge, Linksausläufe, Rechtsrunde und Fußschleifen (was den Spalten in Abbildung 21 entspricht) unterteilt werden. Leere Kästchen der Abbildung stehen für Kombinationen, welche in der DEK nicht vorkommen.


**Abbildung 21: Zeichentypen laut [28].**

Die Auslautvokalisation eines vorhergehenden Vokals oder Diphthongs wird in der DEK in bestimmten Fällen („ä“, „ö“, …) verstärkt geschrieben und erreicht ihr Maximum in Punkten mit lokal minimaler Krümmung (siehe auch Abbildung 22). Um dies nachzubilden, verwendet der Autor anstelle eins normalen Zeichenbefehls einen Befehl zum Füllen einer Fläche an eben diesen Stellen. Dadurch muss nicht jedes Zeichen einmal normal und einmal verstärkt hinterlegt werden.

Um nun eine Eingabe in DEK-Steneme (d.h. Kurzschriftglyphen) umzuwandeln, muss zuerst die Metaform des eingegebenen Textes generiert werden. Dies geschieht entweder mithilfe eines Wörterbuchs (z.B. bei Zahlen, Satzzeichen und Verkehrsschriftkürzeln) oder durch eine morphologische Wortzerlegung[6] anhand bestimmter Regeln. Aufgrund der alternierend auftretenden Konsonanten und Vokale in einem Wort und dem zu beachtenden Auslautvokalisationsprinzip verwendet der Autor folgende Zerlegung:

$$(\,,K\,)\,\{\,(\,V\,,K\,)\,\}^{*}$$

Bsp. $(,t)$(ü,$r)$ für das Wort „Tür“. Dabei bezeichnen K Konsonantenzeichen und V Vokalzeichen. Diese Metaform wird in METAFONT mit Init und Join Anweisungen in die Wortsteneme umgesetzt. Für das Wort „Ritter“ ergibt sich also die Metaform $(,r)(i,t)(e,r)$, man beachte, dass es in der DEK keine doppelten Mitlautfolgen und keine Großschreibung gibt, welches dann in die METAFONT Anweisungen $I(,r);J(i,t);J(e,r);$ umgesetzt wird. Es wird also mit dem Zeichen für das „r“ begonnen, dann das „t“ angefügt und schließlich das „r“. Dabei bestimmt der erste Parameter eines Joins die relative Positionierung und evtl. Verstärkung des im zweiten Parameter angegebenen Konsonantenzeichens gemäß dem Auslautvokalisationsschema (siehe Abbildung 22).

[6] Die Morphologie befasst sich mit dem Aufbau von Wörtern aus Morphemen, den kleinsten semantisch interpretierbaren Zeichen einer Sprache.

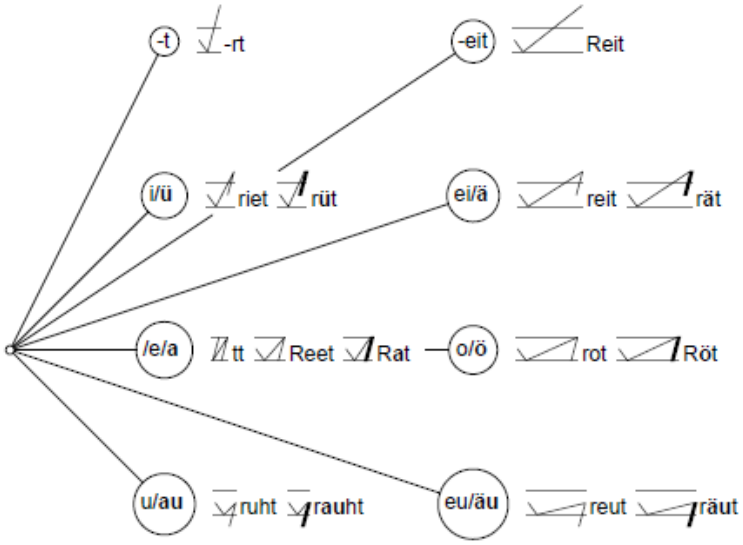



**Abbildung 22: Schema der Auslautvokalisation [28].**

Die Verbindungsroutine des Join Befehls unterscheidet dabei die in Abbildung 21 dargestellten Typen um Ende und Anfang zweier Konsonantenzeichen so glatt als möglich verbinden zu können. Hierbei können manche Verbindungen nur rein Punkt zu Punkt erzeugt werden, bei anderen muss die Tangente der Verbindungsgeraden der Tangente im Anfangs- bzw. Endpunkt eines Zeichens angepasst werden.

Text2DEK kann leider nicht erworben oder heruntergeladen werden, um es in eigenen Anwendungen oder Latex Texten zu verwenden. Die TU Clausthal bietet jedoch eine Webanwendung[7], mit deren Hilfe Formulareingaben mittels Text2DEK in ein GIF Bild umgewandelt werden.

[7] http://www3.rz.tu-clausthal.de/~rzsjs/steno/DEK.php

# 4. Methode

In diesem Kapitel erläutere ich die verwendeten Ansätze und außerdem, warum ich mich für diese entschieden habe.

## 4.1. Entscheidung gegen einen globalen Ansatz

Die Zielsetzung dieser Arbeit war es, sowohl globale als auch segmentierende Ansätze für die Erkennung der DEK genauer zu betrachten. Beim Studium der in Kapitel 3.1 aufgeführten Methoden war jedoch schnell klar, dass diese für das Erkennen der DEK nur wenig geeignet sind.

Hauptgrund hierfür ist, das für diesen Ansatz notwendige Wörterbuch, mithilfe dessen eingegebene Wörter als Ganzes erkannt werden sollen. Dies bedeutet, dass für jedes Wort, das erkannt werden soll, müsste ein Eintrag im Wörterbuch vorhanden sein. Dies gestaltet sich vor allem durch die Anwendungsgebiete der DEK aber als schwierig. Zum einen werden Kurzschriften vor allem in spezialisierten Umgebungen, wie z.B. in der Medizin, der Kriminalistik oder bei Gerichtsverhandlungen und in Plenumssitzungen verwendet. In den ersten zwei Fällen müsste das Wörterbuch vor allem sehr themenbezogen ausgeführt werden, da z.B. in der Medizin viele lateinische Ausdrücke für Krankheiten oder Körperteile verwendet werden. Es wäre zwar möglich, solche besonderen Ausdrücke in spezialisierten Wörterbüchern für die einzelnen Anwendungsgebiete vorzusehen, doch deren Größe würde dann wieder Probleme bei der Geschwindigkeit der Suche bereiten. Bei den letzteren Gebieten ergeben sich vor allem Probleme durch Eigennamen, die in einem Wörterbuch in der Regel nicht vorkommen. Könnte man sich noch vorstellen, dass Namen wie „Schmidt“ oder „Müller“ in einem angepassten Wörterbuch vorkommen, so wäre es vor allem bei ausländischen Namen unmöglich. Ein weiteres Problem bei der Wörterbuchvariante in Zusammenhang mit der DEK ergibt sich durch die Möglichkeit, Wörter so zu schreiben, wie sie gesprochen werden. So ist es in der DEK z.B. erlaubt, „Ingenieur“ sowohl in korrekter Rechtsschreibung zu erfassen, aber auch „Ingeniör“ oder „Inscheniör“ sind ebenfalls zulässig (siehe Abbildung 23). Spätestens hier sollte deutlich werden, dass der Aufbau eines adäquaten Wörterbuchs für die DEK mit allen Besonderheiten gänzlich unmöglich ist, doch ohne ausreichendes Wörterbuch ist keine annehmbare Erkennungsrate möglich.

**Abbildung 23: "Ingenieur" vs. "Ingeniör" vs. "Inscheniör" in der DEK.**

Einige der vorgestellten Methoden beschreiben eine Erkennung durch Zusammensetzen einzelner Buchstaben, also dass ein Wort durch ein Sliding Window gescannt wird und Buchstabe für Buchstabe erkannt wird. Die gezeigten Ansätze verwenden dafür aber ebenfalls ein Wörterbuch, um die Erkennung zu beschleunigen. Aber selbst wenn die Suchraumproblematik ohne Einsatz eines Wörterbuchs in den Griff zu bekommen wäre, so bereitet hier eine andere Eigenschaft der DEK Probleme: die Hoch- und Tiefstellung von Konsonanten, um bestimmte Vokale anzuzeigen. Während die gezeigten Ansätze mit einer konstanten Texthöhe und der Gewissheit von maximal drei vertikalen Bereichen (obere, mittlere und untere Zone) arbeiten können, sind diese bei der DEK nicht gegeben. Innerhalb des gescannten Bereichs müsste dann zusätzlich zum Zeichen selber ebenfalls die Lage relativ zum vorhergehenden Zeichen berücksichtigt werden, was bei den genannten Methoden nicht vorgesehen ist. Abbildung 24 zeigt die Problematik anhand des Wortes „identifizieren“, bei dem jedes Konsonantenzeichen nach einem „i“ um eine halbe Stufe höher

gestellt wird. Außerdem zeigt dieses Beispiel auch, dass es bei der DEK ebenso notwendig ist, die Verbindungen zwischen zwei Konsonantenzeichen zu klassifizieren.

**Abbildung 24: Hochstellung der Konsonantenzeichen durch Vokal "i" hier im Wort "identifizieren".**

Erschwerend kommen Überlappungen von Konsonantenzeichen selbst bei idealer Schreibweise schriftbildbedingt sehr häufig vor. Abbildung 25 zeigt das Wort „Pflock“ in der DEK. An diesem Beispiel zeigt sich die genannte Problematik sehr gut, denn das „l“ rutscht hier unter die Schleife des „Pf“ und somit in dessen Bereich hinein. Mit globalen Methoden ist es äußerst schwierig, diesen Sachverhalt nachzubilden. Diese Eigenschaft behindert auch die Möglichkeit einer Query-by-Example Datenbank. Mithilfe von Text2DEK (siehe Kapitel 3.4) ließe sich eine Datenbank einzelner Konsonantenzeichen und Kürzeln aufbauen, die anhand eines eingegebenen Zeichens das Ähnlichste liefert. Doch hierzu müsste eine automatische Segmentierung durch die angewendete Methode gegeben sein, analog zu der Sliding Window Methode für normale Langschriftzeichen, was durch die Überlappungen aber schwer erreicht werden kann.

**Abbildung 25: Überlappungen von Konsonantenzeichen, hier das „Pf“ und das „l“ im Wort "Pflock".**

Durch das nahezu unmögliche Aufbauen eines adäquaten Wörterbuchs, wird der Einsatz globaler Methoden sehr schwer bis unmöglich. Aus diesen Gründen wurden Adaptierungen der vorgestellten Ansätze sowie die Ausarbeitung einer eigenen Methode für die globale Erkennung nicht näher in Betracht gezogen.

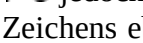
## 4.2. Segmentierung mittels Histogramm

Die Methode von Rehman et al. [21] zur impliziten und expliziten Segmentierung eines Wortes oder von Shaikh et al. [19] mit Height Profile Vectors wirkten zunächst sehr vielversprechend für die Segmentierung der DEK. Wie in Kapitel 3.2 beschrieben, werden bei beiden Methoden die PSPs dabei auf Bereiche mit Werten von 0 oder 1 im vertikalen Histogramm (Rehman) bzw. im HPV (Shaikh) gesetzt. PSPs im Bereich von Schleifen und anderen markanten Punkten werden anschließend verworfen. Ebenso PSPs die in einem bestimmten Abstand zueinander stehen.

Bei beiden Ansätzen ist eine Preprocessing Phase für die Korrektur einer evtl. Schrägstellung notwendig. Ähnlich wie von Petr Slavik und Venu Gofindaraju in [29] vorgeschlagen, werden dabei die Winkel zwischen allen Abstrichen eines Wortes und der x-Achse berechnet. Es werden hier deshalb nur die Abstriche berücksichtigt, da Konsonantenzeichen i.d.R. durch Abstriche repräsentiert werden (siehe Analyse von Herrn Sarman in Kapitel 3.4). Aufstriche stellen die Verbindung zwischen zwei Konsonantenzeichen dar und sind für die Geradestellung der Konsonantenzeichen deshalb irrelevant. Der durchschnittliche Wert dieser Berechnungen ergibt schließlich die Korrekturgröße für eine Scherung des gesamten Wortes.

An Stellen, an denen das vertikale Histogramm den Wert 0 oder 1 aufweist wird anschließend ein PSP gesetzt. Zusätzlich wird am Anfang und Ende des Wortes ein PSP fest vorgeschrieben. Danach

werden die Abstände zwischen den PSPs überprüft. Zwischen zwei PSPs muss ein Abstand von mindestens acht Pixeln liegen, dieser Wert wurde dabei empirisch festgelegt. Außerdem ist ein PSP nur dann tatsächlich gültig, wenn in seinem näheren Umfeld ebenfalls potentielle PSPs liegen. Damit werden Ausreißer verhindert, die durch das Rauschen bei der Erfassung der Stiftdaten erzeugt wurden. Liegen viele PSPs in geringem Abstand zueinander, wird nur der erste und letzte aller solcher PSPs verwendet. Dadurch soll sichergestellt werden, dass auf einer langen Gerade nur am Anfang und Ende der Geraden segmentiert wird und diese nicht auch noch in mehrere Teile geteilt wird.

Der Ansatz funktioniert für manche Zeichen, wie z.B. das  sehr gut, für andere, wie z.B.  oder  jedoch überhaupt nicht. Der Grund hierfür ist, dass diese Zeichen logischerweise inmitten eines Zeichens ebenfalls öfters den Wert 1 im Histogramm besitzen. Des Weiteren ergibt sich bei dieser Methode ebenfalls das Problem, dass sich überlappende Konsonantenzeichen nicht segmentiert werden würden (siehe Kapitel 4.1).

## 4.3. Segmentierung an Aufstrichen

Durch die Probleme mit der Histogramm Methode entstand schließlich ein anderer vielversprechender Ansatz. Die Zeichen, die fälschlicherweise in der Mitte segmentiert wurden, waren immer Abstrichzeichen. Es sollen also Verbindungen zwischen zwei Zeichen segmentiert werden, die bei idealer Schreibweise immer Aufstrichen entsprechen. Abstriche, die nur innerhalb eines Zeichens vorkommen, sollten jedoch nicht segmentiert werden (siehe hierzu auch Kapitel 2.1 und 3.4). Eine genauere Analyse aller Konsonantenzeichen und Kürzel zeigt, dass wenn Aufstriche innerhalb eines Zeichens vorkommen, dann normalerweise nur innerhalb von Schleifen, wie in den Beispielen in Abbildung 26.

**Abbildung 26: Beispiele für DEK Zeichen mit Aufstrichen innerhalb von Schleifen (hier „pf“, „z“, „sch“, „d“).**

Außerdem sind nur Aufstriche bei der Segmentierung zu betrachten, die in positive x- und y-Richtung verlaufen. Da eine solche Erkennung nicht mehr von einer vertikalen Linie durch das gesamte Wort abhängt, wie es bei der Histogrammvariante notwendig war, können auch problemlos sich überlappende Konsonantenzeichen (wie z.B. „Pf“ und „l“ in Abbildung 25) sauber getrennt werden. Abbildung 27 zeigt die Idee anhand des Wortes „Frankreich“.

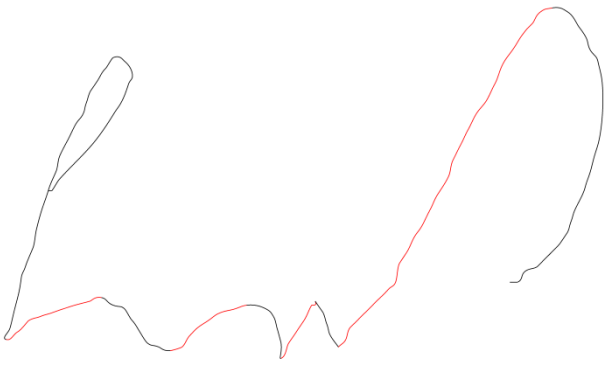


**Abbildung 27: Wort "Frankreich" mit Konsonantenzeichen (schwarz) und Verbindungen (rot).**

### 4.3.1. Schleifenerkennung

Wie bereits erwähnt, kommen Aufstriche innerhalb eines Konsonantenzeichens oder eines Kürzels normalerweise nur innerhalb von Schleifen vor. Um zu verhindern, dass an solchen Aufstrichen segmentiert wird muss also erkannt werden, ob es sich um einen Aufstrich innerhalb einer Schleife handelt oder nicht. Dabei gibt es drei Arten von Schleifen:

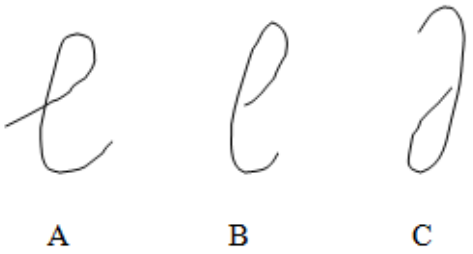



**Abbildung 28: Drei mögliche Schleifen - Closed Loop (A), Open Loop am Wortanfang (B), Open Loop am Wortende (C).**

Typ A Schleifen sind dabei relativ einfach zu definieren: schneidet sich eine Eingabe innerhalb eines einzigen Pen-Down Pen-Up Zyklus mit sich selbst, dann sind alle Punkte, die zwischen den Schnittpunkten liegen, innerhalb einer Schleife, alle anderen nicht. Schwieriger wird es, wenn ein Zeichen mehrere, verschachtelte Schleifen besitzt (siehe Abbildung 29). Dafür müssen die Ein- und Austrittspunkte jeder einzelnen Schleife berechnet werden. Im Anschluss werden alle Punkte in der Reihenfolge durchlaufen, in der sie eingegeben wurden. Jeder Punkt, bis zu dem mehr Schleifeneintrittspunkte als Austrittspunkte passiert wurden, ist Teil einer Schleife.

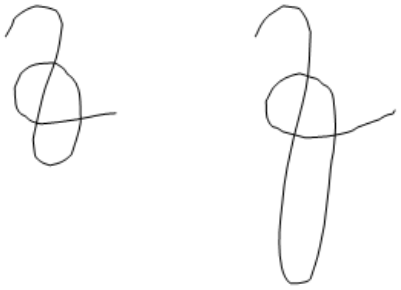


**Abbildung 29: Zeichen mit verschachtelten Schleifen, hier "mp" und "mpf".**

Am Anfang und am Ende eines Wortes kann es außerdem zu so genannten Open-Loops kommen. Das sind Schleifen, die eigentlich geschlossen sein sollen und für den Benutzer vielleicht sogar als geschlossen erscheinen, aber in den Daten, die vom Tablet geliefert werden, keinen Schnittpunkt aufweisen (siehe Typ B und C in Abbildung 28). Analog zu Ma und Leedham [16], die bei der Erkennung von Requn ebenfalls diese drei Schleifentypen unterscheiden, muss die Eingabe dabei nur entsprechend erweitert werden, um aus Typ B und C Schleifen des Typs A werden zu lassen. Dabei werden Typ B Schleifen, die nur am Wortanfang vorkommen können, in entgegengesetzter Strichrichtung erweitert. Diese Erweiterung muss allerdings nur geschehen, wenn der Anfangspunkt der Eingabe nicht am linken Rand liegt. Ist dies der Fall, kann es keinen Abstrich noch weiter links geben, mit dem sich die Eingabe schneiden könnte. Typ C Schleifen werden in Strichrichtung erweitert, aber auch nur dann, wenn der letzte Punkt der Eingabe nicht am rechten Rand liegt. Hier gäbe es ebenfalls keinen Abstrich weiter rechts, mit dem ein Schnittpunkt gebildet werden könnte. Abbildung 30 zeigt die Erweiterung bei beiden Typen durch gestrichelte Linien. Anschließend können Typ B und C Schleifen mit derselben Methode wie Typ A Schleifen erkannt werden.

B C

**Abbildung 30: Erweiterung bei Typ B und C Schleifen.**

### 4.3.2. Schematischer Ablauf der Segmentierung

Die Eingabe wird erfasst und alle Ein- und Austrittspunkte aus Schleifen gesucht. Anschließend werden die erfassten Punkte sequentiell durchlaufen und ein Segmentierungspunkt gesetzt, wenn

i. $l(P_i)$ wahr, $l(P_{i-1})$ falsch und $a(P_i)$ wahr sind

oder

ii. $l(P_i)$ falsch, $l(P_{i-1})$ wahr und $a(P_i)$ wahr sind

oder

iii. $l(P_i)$, $l(P_{i-1})$, $a(P_{i-1})$ falsch und $a(P_i)$ wahr sind

oder

iv. $l(P_i)$, $l(P_{i-1})$, $a(P_i)$ falsch und $a(P_{i-1})$ wahr sind

wobei $a(P_i)$ und $l(P_i)$ wie folgt definiert sind:

$a(P_i)$ ist wahr, genau dann wenn $x(P_i) < x(P_{i+1})$ und $y(P_i) < y(P_{i+1})$ wahr sind[8],
mit $x(P_i)$ für die x-Koordinate von Punkt $i$
und $y(P_i)$ für die y-Koordinate von Punkt $i$.

$l(P_i)$ ist wahr, genau dann wenn $entry(P_i) > exit(P_i)$ wahr ist,
mit $entry(P_i)$ für die Anzahl der Schleifeneintritte bis zum Punkt $i$
und $exit(P_i)$ für die Anzahl der Schleifenaustritte bis zum Punkt $i$.

Formel i (a) sorgt dafür, dass ein PSP gesetzt wird, wenn sich der Punkt auf einem Aufstrich befindet und eine Schleife eines Konsonantenzeichens betreten wird. Formel ii (b) setzt einen PSP, wenn sich der Punkt auf einem Aufstrich befindet und die letzte Schleife eines Konsonantenzeichens verlassen wurde. Die Formeln iii (d) und iv (c) setzen PSPs an den Übergängen zwischen Aufstrichen und Abstrichen außerhalb von Schleifen. Die Kleinbuchstaben in Klammern korrespondieren dabei mit den zugehörigen Punkt in Abbildung 31 um die einzelnen Regeln nochmals zu veranschaulichen.

[8] Bei Verwendung eines kartesischen Koordinatensystems.

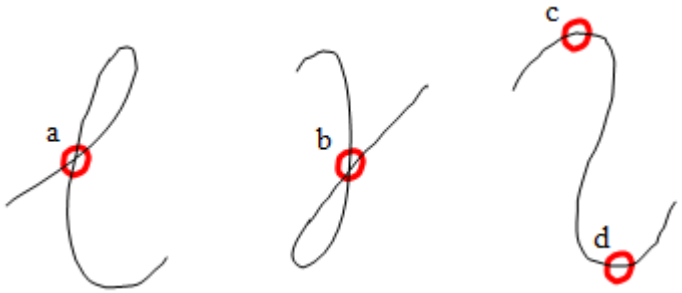



**Abbildung 31: Segmentierungspunkte an Schleifen (a, b) und bei Übergängen (c, d).**

Zeichen, bei denen dieser Algorithmus nicht korrekt funktioniert, sind in Abbildung 32 aufgeführt. Das Problem beim „c“ und „cr“ ist ein kurzer Aufstrich innerhalb des Zeichens. Diesem Problem kann damit begegnet werden, dass eine Überprüfung auf eine Mindestlänge eines Aufstrichs eingeführt wird. $a(P_i)$ darf also z.B. nur wahr sein, wenn es für eine definierte Distanz der nächsten x Pixel wahr ist.

Eine andere Möglichkeit, mit der auch der Fall von „u(h)r“ gelöst werden könnte, ist die Zeichen zu segmentieren und bei der späteren Erkennung diese entsprechend zu berücksichtigen. Wie auch in Kapitel 3.2 gezeigt, ist es besser eine Übersegmentierung zu verwenden und die Spezialfälle dann in der nächsten Phase zu korrigieren, als evtl. Segmente und damit Informationen zu verlieren.

**Abbildung 32: Konstantenzeichen und Kürzel die nicht korrekt segmentiert werden („c“, „cr“ und „u(h)r“).**

### 4.3.3. Umsetzung

Da die Microsoft INK API Schnittpunkte eines Stroke Objekts (siehe Kapitel 2.3) mit sich selbst bereits berechnet, müssen diese nur noch verifiziert werden. Denn es werden teilweise auch Schnittpunkte berechnet, die zufällig und nicht als Teil einer gewollten Eingabe entstanden sind (siehe hierzu auch Kapitel 5.1, wo dieses Problem noch genauer erläutert wird). Daher werden im ersten Schritt Schnittpunkte verworfen, die in der Sequenz der Punkte dicht beieinander liegen. Im zweiten Schritt wird für alle übrigen Schnittpunkte das jeweilige Gegenstück berechnet. Außerdem werden in diesem Schritt Schnittpunkte verworfen, die kein Gegenstück aufweisen. Dieses Phänomen tritt zum einen bei dicht beieinander liegenden Schnittpunkten auf, die teilweise im ersten Schritt verworfen wurden. Zum anderen werden manchmal von der API falsche Schnittpunkte geliefert, die in Wirklichkeit keine Schnittpunkte sind. Dadurch entsteht eine Liste mit Eintrittspunkt in eine Schleife und dem jeweils zugehörigen Austrittspunkt. An dieser Stelle muss jedoch angemerkt werden, dass die Schnittpunkte den vom Tablet gelieferten Eingabepunkten entsprechen (diese Tatsache wird im weiteren Verlauf nochmals aufgegriffen).

Offene Schleifen am Anfang und Ende werden von der API nicht erkannt, daher müssen diese berechnet werden. Hierzu wird die Eingabe am Anfang und Ende, wie in Kapitel 4.3.1 gezeigt, erweitert. Wird dabei ein Schnittpunkt festgestellt, wird der Index des Schnittpunkts für Anfang und Ende separat für den nächsten Schritt der Segmentierung gespeichert. Diese Berechnung erfolgt bereits anhand der von der API gelieferten Bezier-Darstellung der Eingabe, welche einer geglätteten Version der Eingabe entspricht und weitestgehend frei von Rauschen ist.

Anschließend werden in einer Loop[9] alle Stützpunkte der Bezierkurve durchlaufen. Es wird deshalb die Bezierkurve verwendet, da diese für die Berechnung der Aufstriche kein Rauschen enthält und daher kein Preprocessing erfordert. Dabei beginnt die Loop entweder beim ersten Punkt oder bei einem Schnittpunkt, der in der vorherigen Phase erkannt wurde und läuft bis zum letzten Punkt oder ebenfalls bis zu einem vorher erkannten Schnittpunkt. Dadurch werden automatisch offene Schleifen am Anfang und Ende eines Wortes von der Segmentierung ausgeschlossen.

Innerhalb der Loop wird zunächst die Überprüfung für einen Aufstrich durchgeführt. Dabei werden nur Aufstriche mit einer bestimmten Länge berücksichtigt, um dem am Ende des vorigen Kapitels angesprochenen Problem vorzubeugen. Es wird also keine Übersegmentierung durchgeführt, sondern genau an den Zeichengrenzen versucht zu segmentieren. Wird dabei ein Übergang von einem Abstrich in einen Aufstrich oder umgekehrt festgestellt, wird dies entsprechend für weitere Durchgänge gespeichert. Zusätzlich wird dabei der Winkel berechnet, den der aktuelle Punkt mit dem vorhergehenden und dem nachfolgenden Punkt erzeugt. Dieser Wert wird bei einer eventuellen Segmentierung gleich beim jeweiligen Segment gespeichert und stellt den Ein- und Austrittswinkel in bzw. aus dem Segment dar, also ob der Übergang zwischen Verbindung und Zeichen am Anfang und Ende des Zeichens eher spitz oder stumpf verläuft.

Im Anschluss wird der Zähler für geschlossene Schleifen ausgesteuert. Dies geschieht durch eine Abstandsberechnung des aktuellen Punktes der Bezierkurve zu den Ein- und Austrittspunkten, die am Anfang der Segmentierung berechnet wurden. Ist die Liste der Ein- und Austrittspunkte gleich groß, ist noch keine Schleife betreten worden und es muss nur der Eintritt überprüft werden. Ist der Abstand des aktuellen Punktes zum ersten Schnittpunkt kleiner als eine bestimmte Schwelle, so wird ein Eintritt festgestellt und der Zähler erhöht. Die Schwelle ist dabei kleiner als der halbe Abstand, der in der ersten Phase für die minimale Distanz zwischen zwei Schnittpunkten festgelegt wurde. Da die Schnittpunkte nicht zwangsläufig genau auf der Bezierkurve liegen müssen, muss hier der Abstand in Kombination mit einer Schwelle verwendet werden. Erst wenn in weiteren Durchgängen der Loop sich der dann aktuelle Punkt wieder vom Schnittpunkt entfernt hat, wird der Eintrittspunkt aus der Liste der Eintrittspunkte entfernt. In weiteren Durchgängen der Loop werden somit auch die Austrittspunkte überprüft. Dieses Vorgehen ist notwendig, da sonst nach dem Eintritt in eine Schleife auch sofort der Austritt erkannt werden würde. Denn zu dieser Zeit ist der Abstand des aktuellen Punktes zum Ein- und Austrittspunkt gleich groß, da diese im Koordinatensystem i.d.R. auf denselben Punkt fallen. Mit Austrittspunkten wird dann genauso verfahren, nur dass der Schleifenzähler beim Verlassen einer Schleife herabgesetzt wird. Somit gibt der Zählerstand zu jeder Zeit an, in wie vielen Schleifen sich der aktuelle Punkt gerade befindet.

Bei der Abstandsberechnung zwischen dem aktuellen Bezier-Spline Teilstück und den Schnittpunkten muss jedoch die Eigenschaft von Bezier-Kurven beachtet werden, auch lange Kurven mit wenigen Stützstellen darstellen zu können. Zunächst wurden nur die Anfangs- und Endpunkte der einzelnen Bezier Teilstücke verwendet. Dies führte jedoch zu Problemen, wenn die Eingabe z.B. eine lange Gerade enthält. So konnte es vorkommen, dass die Punkte des Teilstücks, das eigentlich den Schnittpunkt enthielt, nicht in der Nähe des Schnittpunktes waren (siehe Abbildung 33). Deshalb müssen auch Punkte innerhalb eines Teilstückes berechnet werden. Für Kubische Bezierkurven erfolgt die Berechnung einzelner Punkte auf folgende Weise:

$$C(t) = (-P_0 + 3P_1 - 3P_2 + P_3)t^3 + (3P_0 - 6P_1 + 3P_2)t^2 + (-3P_0 + 3P_1)t + P_0$$

mit $t \in [0,1]$. Dadurch kann die Genauigkeit beliebig durch die Schrittweite von $t$ gewählt werden.

[9] for-Schleife. Hier die Bezeichnung Loop verwendet um dem Leser die Unterscheidung zwischen einer Schleife in der Implementierung und einer Schleife innerhalb eines DEK-Zeichens leichter zu machen.

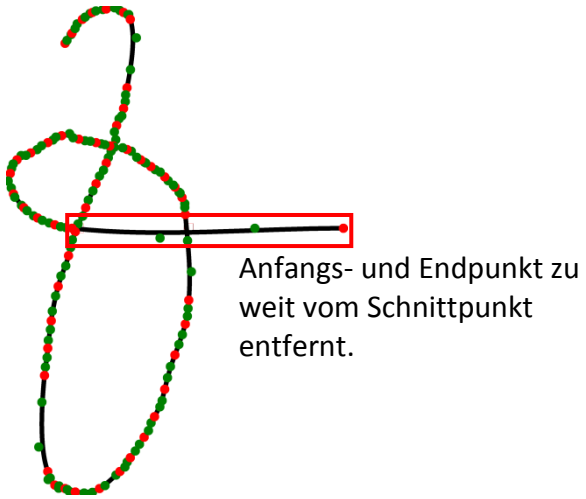



**Abbildung 33: Fehlerhafte Segmentierung durch langen Bezer-Spline (Rotes Rechteck). Rote Punkte sind jeweils Anfangs- und Endpunkte, grüne Punkte sind Stützstellen.**

Am Ende der Loop wird dann über die Segmentierung entschieden. Wenn der Schleifen Zähler null ist und ein Aufstrich erkannt wurde, wird segmentiert. Außerdem wird segmentiert, wenn sich dieser Zustand wieder ändert, also wenn entweder der Schleifenzähler ungleich null wird oder es sich nicht mehr um einen Aufstrich handelt. Dabei wird zusätzlich zu den Bezier-Punkten und des Übergangswinkel auch die Anzahl und Position der Schleifen hinterlegt, genauer gesagt ob es sich um einen Konsonant oder Vokal handelt, die Höhe einer Stufe und des Schreibraums und ob mit verstärktem Druck geschrieben wurde oder nicht.

Ist die Loop am Ende der Bezier-Kurve angelangt oder an dem Index, der bei der Erkennung für offene Schleifen als Schnittpunkt berechnet wurde, endet die Segmentierung.

## 4.4. Klassifizierung der Konstantenzeichen

### 4.4.1. Verwendung der Anfänge und Enden als Merkmale

Herr Sarman klassifiziert bei der Umsetzung von Text2DEK [28] die Konsonantenzeichen anhand deren Anfang und Ende (siehe Abbildung 21), um eine möglichst glatte Verbindung zweier Zeichen zu erhalten. Bei näherer Betrachtung aller DEK Zeichen in [2] zeigt sich, dass dieser Ansatz bereits ein sehr gutes Merkmal für die Zeichenklassifikation darstellt. Weitere Merkmale, wie Größe eines Zeichens sowie Position und Größe von Schleifen könnten anschließend verwendet werden, um innerhalb der 25 möglichen Kombinationen das Zeichen letztlich genau zu klassifizieren.

Bei der korrekten Klassifikation von Anfang und Enden ist zu beachten, dass der gezeigte Segmentierungsalgorithmus Zeichen der vierten Reihe und Spalte in Abbildung 21 nicht in korrekte DEK Zeichen segmentiert, sondern die Segmentierung an den Extremwerten stattfindet. Für die Klassifizierung stellt dies aber kein Problem dar, wie im weiteren Verlauf gezeigt wird. Bei genauerer Betrachtung der 5 x 5 Typen lassen sich folgende Beobachtungen feststellen:

- Zeichen, die mit Typ 1 – 3 anfangen (entspricht Zeile 1 – 3) bzw. mit enden (entspricht Spalte 1 – 3), haben alle scharfe Übergänge von der Vokalverbindung zum Konsonantenzeichen und unterscheiden sich nur durch die Krümmung zu Beginn bzw. Ende des Zeichens.
- Anfänge bzw. Enden mit Typ 4 haben einen flachen Übergang.

- Zeichen, die mit Typ 5 beginnen oder enden haben ebenfalls einen flachen Übergang. Die Besonderheit: Sie verlaufen in positive X und Y Richtung. Dies kann nur bei Zeichen mit einer Schleife am Anfang bzw. Ende vorkommen.

Typ 4 und 5 stellen bei der Klassifizierung der Anfang und Enden eines Zeichens kein Problem dar, da sie über eindeutige Merkmale verfügen. Bei den drei anderen Typen muss die Krümmung der Kurve zu Beginn bzw. Ende des Zeichens untersucht werden. Hierzu werden die Winkel gemessen, die ein Punkt auf der Kurve mit seinen zwei Nachbarpunkten bildet. Als Punkte wurden hier mehrere in gleichem Abstand zueinander stehende Punkte auf den ersten $n$ Pixel des Zeichens verwendet und anschließend der Durchschnitt aller Winkel gebildet. Hierbei wird $n$ abhängig von der Höhe eines Zeichens gesetzt, da sich der zu messende Bereich stark zwischen Zeichen mit einer Höhe von einer halben Stufe zu Zeichen mit drei Stufen unterscheidet. Anhand des Durchschnittswerts kann anschließend die durchschnittliche Krümmung in diesem Bereich berechnet werden. Würden hier die Tangentensteigungen verwendet werden, wären die Werte abhängig von der Schrägstellung der Zeichen. Anhand der Krümmung, ob rechtsgerichtet, linksgerichtet oder relativ gerade, kann anschließend eine Unterscheidung der Typen 1 bis 3 gefällt werden.

Bei Tests zeigte sich jedoch, dass dieser Algorithmus nur bei sehr gut quantifizierten Eingabepunkten vom Tablet adäquat funktioniert und daher stark vom verwendeten Tablet und der Genauigkeit der Eingabe abhängt. Eine nähre Analyse der Fehlerfälle zeigte, dass die Berechnung der Krümmung Probleme bereitete. Das Hauptproblem dabei stellt der Bereich für die Messung der Krümmung dar. Wenn dieser die richtige Größe hat, konnten auch die fehlerhaft identifizierten Typen richtig klassifiziert werden. Die richtige Größe ließ sich dabei jedoch nicht für alle Zeichen einer bestimmten Höhe korrekt bestimmen, da in manchen Zeichen bereits früh eine Krümmung innerhalb des Zeichens beginnt, die dann nicht mehr berücksichtigt werden dürfte. Dadurch musste $n$ relativ klein gewählt werden. Doch gerade beim Übergang von Vokalverbindung zu Konsonantenzeichen entsteht das meiste Rauschen durch Verringerung der Schreibgeschwindigkeit. Vor allem da es sich bei den Zeichen, für die die Krümmung betrachtet werden muss, um welche mit spitzem Übergang handelt. Dadurch verläuft der Strich nach dem Übergang oftmals fast in die entgegengesetzte Richtung als vor dem Übergang.

Abbildung 34 zeigt die Problematik anhand des Konsonantenzeichens für „t“, was nur einem einstufigen Abstrich entspricht (Vokalverbindungen wurden in der Abbildung zum besseren Verständnis angedeutet). Idealerweise weist das Zeichen einen scharfen Übergang und eine gerade Linienführung nach dem Übergang auf. Bei der Quantisierung des Tablets kann es aber zu leichten Krümmungen kommen.

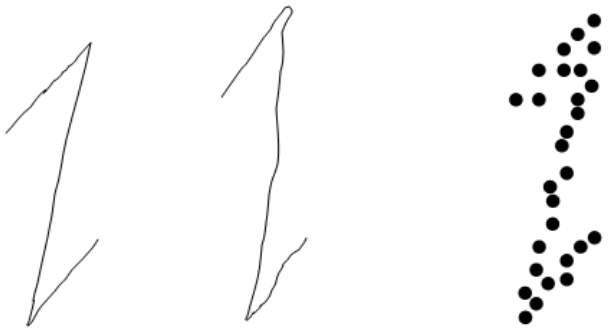


**Abbildung 34: Eingabe eines Zeichens (Ideal, Bezier, Punkte).**

In Abbildung 35 wird der Übergangsbereich vergrößert dargestellt. Wird für die Berechnung der Krümmung nur der Bereich bis zum Strich bei a durchgeführt, ist die Krümmung im Durchschnitt

nicht gerade. Wird die Messung bis zum Strich bei b fortgesetzt, hebt sich die Krümmung im Schnitt wieder auf und wäre nahezu gerade.

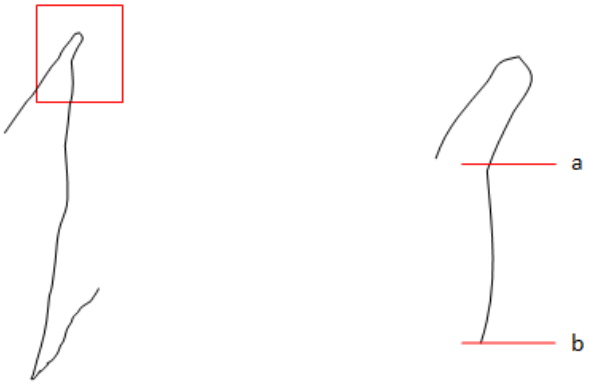



**Abbildung 35: Detailierte Darstellung eines Übergangs (rotes Rechteck). Länge des Messbereichs entscheidend für richtige Erkennung.**

Eine variable Wahl von $n$ anhand bestimmter Merkmale hätte das Problem dabei verringern können, erschien aber letztlich zu aufwändig. Denn die Probleme an diesem Beispiel haben gezeigt, dass eine harte Entscheidung anhand eines fest vorgegebenen Wertes nicht zu guten Ergebnissen führt. Gerade dann nicht, wenn die zu klassifizierenden Werte von einigen Faktoren abhängen und ein gewisses Rauschen in den Daten enthalten ist.

### 4.4.2. Verwendung eines Direction-Chain-Code

Wie im vorherigen Kapitel gezeigt, sind vor allem die Übergänge von Zeichen zu Verbindungen sowie die Form von Anfang und Ende als Merkmale für die Klassifikation sehr interessant. Da diese alleine aber immer noch keine eindeutige Klassifizierung erlauben, wird für weitere Merkmale die Methode von Zobrak und Sze verwendet [30]. Dabei wird eine Eingabe im gleichen Abstand in Teilstriche zerlegt, die Richtung jedes Teilstriches berechnet und in acht Stufen quantisiert (siehe Abbildung 36). Man spricht hierbei von einem 8-Direction-Chain-Code. Dadurch werden sowohl die Anfänge und Enden berücksichtigt, als auch der Teil dazwischen.

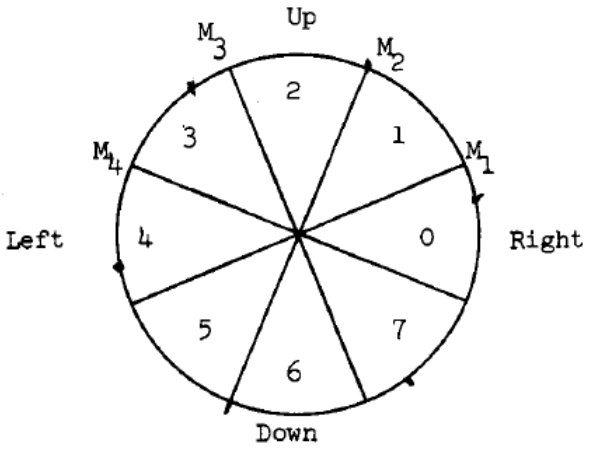



**Abbildung 36: Quantisierung der Strichrichtungen [30].**

Zur genaueren Illustration soll Abbildung 37 dienen. Bei (a) werden eingegebene Punkte gezeigt. Werden diese in der Reihenfolge der Zahlen erfasst, entsteht bei der Quantisierung der Richtungen

eine Sequenz von „6, 6, 0, 0, 2, 2“ oder weiter zusammengefasst „6, 0, 2“ wie in Bild (b). In Bild (c) verläuft die Eingabe von Punkt 1 zunächst zu 3, dann zu 7, 5 und wieder zu 1. Dadurch entsteht eine Sequenz von „6, 1, 6, 3“.

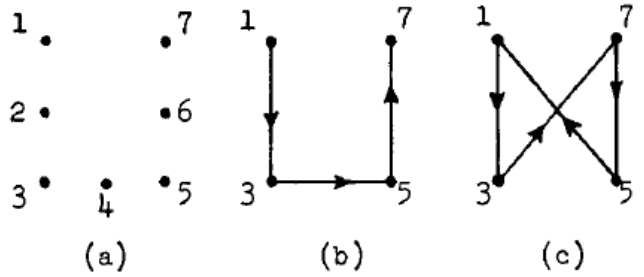



**Abbildung 37: Illustration einer quantisierten Richtungssequenz einer Eingabe [30].**

Durch Variierungen des Abstands für die Zerlegung kann bestimmt werden, wie genau die Auflösung des eingegebenen Musters sein soll. Dabei gilt jedoch zu beachten, dass eine zu hohe Auflösung nicht mehr gut generalisiert und eine höhere Rechenzeit des Classifiers zur Folge hat. Eine zu niedrige Auflösung führt jedoch zu einem Verlust von Informationen. Da die Größe des Merkmalsvektors fest ist, muss der Abstand auch darauf ausgelegt sein, genau so viele Teilstücke zu produzieren, wie der Vektor aufnehmen kann. Die Berechnung des Abstands erfolgt dann wie folgt:

$$Abstand\ zwischen\ Punkten = \frac{Strichlänge\ der\ Eingabe}{Größe\ n\ des\ Merkmalsvektors}$$

Mit diesem Abstand werden genau $n + 1$ Punkte aus der Eingabe extrahiert, was zu $n$ Richtungen führt. Die quantisierten Richtungen werden anschließend normalisiert und jeweils als separates Merkmal dem Merkmalsvektor zugeführt.

Um von der Schreibrichtung unabhängig zu sein, wurde der Algorithmus zunächst leicht verändert. Anstatt die Richtungen zu verwenden, wurden für jeden Punkt der Winkel berechnet, den die beiden Nachbarpunkte in diesem Punkt aufspannen. Somit ist die Lage und Richtung eines Zeichens irrelevant und lediglich die Form wichtig. Da sich manche Zeichen hier aber ähneln, wurde als zusätzliches Merkmal die durchschnittliche Richtung aller Abstriche berechnet. Somit sollte eine eindeutige Entscheidung zwischen zwei von der Form her gleichen Zeichen erreicht werden können.

Anschließende Tests mit neuronalen Netzen konnten die Leistungsfähigkeit der Methode mit den Winkeln anstatt den Richtungen jedoch nicht bestätigen. Vor allem Zeichen mit ähnlicher Form, aber unterschiedlichen Durchschnittsrichtungen wurden dabei oft falsch klassifiziert, obwohl die Trainings- und Testdaten sich im Merkmal der Schreibrichtung zwischen den Klassen deutlich unterschieden. Offenbar generalisierte das neuronale Netz hier zu stark und setzte das Gewicht dieses einen Merkmals nicht hoch genug, als dass eine Unterscheidung möglich gewesen wäre. Auch eine Änderung der Struktur der neuronalen Netze durch Variierung der Anzahl der Neuronen je Schicht und der Anzahl der Schichten selber brachten keine merkliche Verbesserung bei den Ergebnissen.

Hingegen waren die Tests mit den extrahierten Richtungen, wie von von Zobrak und Sze [30] beschrieben, sehr vielversprechend (näheres hierzu in Kapitel 5.2). Dabei zeigte sich auch, dass die Quantisierung der Richtungen anhand von acht Bereichen sogar eine veränderte Schreibweise in Bezug auf die Schrägstellung eines Zeichens zulässt und weiterhin korrekt klassifiziert. Hierbei hilft auch die Normalisierung der Merkmale, durch die ähnliche Bereiche auch ähnliche Werte aufweisen und somit über Abweichungen in der Schreibrichtung gut generalisiert werden kann.

Daher wurden die Tests mit den Winkeln eingestellt und diese Methode für eine genauere Betrachtung ausgewählt.

### 4.4.3. Weitere Merkmale

Zusätzlich zum Direction-Chain-Code werden bereits während der Segmentierung weitere Merkmale extrahiert. Naheliegende ist hierbei die Größe eines Zeichens. Die Höhe eines Zeichens kann in vier Stufen aufgeteilt werden: Zeichen mit einer Höhe von einer halben Stufe, einer ganzen Stufe, zwei Stufen und drei Stufen. Die Breite wird in Pixel gemessen und anhand der Stufenhöhe (Anzahl an Pixel zwischen zwei Stufen) normalisiert. Wie bereits im vorigen Kapitel erwähnt, sind auch die Übergänge eines Konsonantenzeichens wichtig für deren Klassifizierung. Daher wird der Ein- und Austrittswinkel in vier Stufen quantisiert und normalisiert. Vier Stufen sind dabei völlig ausreichend um zwischen spitzen und stumpfen Winkeln unterscheiden zu können. Weitere Merkmale sind die Anzahl und Position der Schleifen. Es können maximal drei Schleifen in einem Zeichen auftreten. Um die Position abbilden zu können, gibt es je ein Merkmal für eine Schleife zu Beginn, in der Mitte und am Ende eines Zeichens, mit dem Wert 0 für keine oder 1 für eine Schleife an der jeweiligen Position. Ein Merkmalsvektor für ein Zeichen hat demnach folgenden Aufbau:

1. Höhe (Anzahl der Stufen)
2. Breite (durch Stufenhöhe normalisiert)
3. Eintrittswinkel (in vier Stufen quantisiert)
4. Austrittswinkel (in vier Stufen quantisiert)
5. Schleife zu Beginn (0 oder 1)
6. Schleife in der Mitte (0 oder 1)
7. Schleife am Ende (0 oder 1)
8. Direction-Chain-Code (Länge wird durch Höhe des Zeichens bestimmt)

### 4.4.4. Klassifizierung mittels Lernverfahren

Lernverfahren sind mittlerweile die State-of-the-art Methode für Klassifizierungsaufgaben. Mit neuronalen Netzen, Hidden Markov Modellen, Bayes Netzen, Support Vektor Maschinen oder selbstlernenden Entscheidungsbäumen, jeweils in diversen Ausführungen, gibt es viele verschiedene Algorithmen, die bereits in zahlreichen Anwendungsgebieten ihre Praxistauglichkeit bewiesen haben. Auch für die Handschriftenerkennung werden mittlerweile fast ausschließlich Lernverfahren eingesetzt (siehe auch Kapitel 3). Je nach gewähltem Algorithmus, können diese sogar als Black-Box betrachtet werden, die ihre Arbeit verrichten, ohne dass der Programmierer sich genauer mit ihrer Funktionsweise auseinandersetzen muss. Wichtiger und meist viel aufwändiger ist die Wahl und Extraktion von relevanten Merkmalen (Features) aus den zu klassifizierenden Mustern. In der Regel werden bei einer Klassifizierungsaufgabe nach und nach weitere Features hinzugezogen, bis der Classifier eine ausreichende Trefferquote erreicht. Dabei wird jedes neue Merkmale darauf geprüft, ob es tatsächlich zu einer Verbesserung der Erkennung beiträgt, also ob sich der Informationsgehalt des Merkmals lohnt. Zu viele Features die nur wenig zur Erkennung beitragen, können zu einer schlechten Generalisierung des Classifiers führen. Dies zeigt sich darin, dass der Classifier zwar gelernte Muster sehr gut, neue und unbekannte Muster hingegen eher schlecht klassifizieren kann. Die Leisungsfähigkeit eines Classifiers wird daher anhand einer Datenbank von Mustern überprüft. Diese Datenbank wird meistens zu 2/3 Trainings- und 1/3 Testdaten aufgeteilt. Mit den Trainingsdaten wird der Classifier trainiert, bis dieser eine bestimmte Erkennungsrate auf den Trainingsdaten erreicht. Mit den Testdaten wird anschließend überprüft, wie gut der Classifier mit nicht gelernten Mustern umgeht und die Erkennungsrate sowie

die Fehlerrate berechnet. Diese bestimmt dann die Qualität des eingesetzten Lernverfahrens mit den gewählten Features.

**Auswahl**

In dieser Arbeit werden für die Klassifizierung des Direction-Chain-Code und der weiteren beschriebenen Merkmale sowohl neuronale Netze als auch lernende Entscheidungsbäume näher betrachtet und damit die Leistungsfähigkeit der gewählten Merkmale verifiziert. Diese Entscheidung basiert hauptsächlich auf dem begrenzten zeitlichen Rahmen, der eine genauere Betrachtung einzelner Lernverfahren nicht zuließ. Die gewählten Methoden bieten den Vorteil, dass sie ohne großen Konfigurationsaufwand zu sehr guten Ergebnissen führen. Da die Hauptaufgabe bei Klassifizierungsproblemen in der Wahl der geeigneten Merkmale besteht, liegt der Fokus der Arbeit nicht auf den Lernverfahren selber. Dennoch folgt nun ein kurzer Überblick der gewählten Verfahren. Für eine nähere Betrachtung empfiehlt sich das Standardwerk für Klassifizierungsaufgaben „Pattern Classification“ von Duda und Hart [31] als Einführung kann auch das deutschsprachige Buch „Grundkurs Künstliche Intelligenz“ von Wolfang Ertel [32] dienen.

Neuronale Netze haben sich in der Vergangenheit für Klassifizierungsaufgaben bewährt, bei denen die zu klassifizierenden Daten nicht eindeutig linear separierbar sind. Dabei orientieren sie sich von der Funktionsweise her am menschlichen Gehirn: Vernetzte Neuronen, die aktiviert werden, wenn die gewichteten Eingänge eine bestimmte Schwelle überschreiten und somit wieder Eingabe für weitere Neuronen eine Schicht höher bilden. Gerade bei der Erkennung von Handschriften wird neben Hidden Markov Modellen und Bayes Netzen gerne ein neuronales Netz verwendet, weil sie gute Ergebnisse liefern, ohne vorher aufwändig für die Klassifikationsaufgabe parametriert werden zu müssen. Der Nachteil von neuronalen Netzen liegt jedoch in der Ausgabe eines Netzes. Diese entspricht generell nicht den tatsächlichen Wahrscheinlichkeiten, welches ein eingegebenes Muster einer bestimmten Klasse entspricht. Für einen anschließenden Post Processing Schritt, wie der Kontexterkennung, sind die Wahrscheinlichkeiten jedoch nötig. Allerdings konnten Hung et al. an diversen Aufgabenstellungen zeigen, dass die Ausgabe nahezu den Wahrscheinlichkeiten entspricht, wenn die Netze nur wenige verdeckte Schichten aufweisen [33]. Ein weiterer Nachteil ist in der Struktur von neuronalen Netzen begründet. Aufgrund des komplexen Zusammenspiels von gewichteten Verbindungen der Neuronen und deren Aktivierungsfunktion ist es nahezu unmöglich das Verhalten eines Netzes als Mensch nachzuvollziehen. Arbeitet ein Netz nicht wie erwartet, ist die Diagnose daher meist zu aufwändig.

Bei Entscheidungsbäumen hingegen ist eine Diagnose sehr einfach, da die Klassifikation hierbei ähnlich einer menschlichen Entscheidung gefällt wird. Die Merkmale werden anhand einer Reihe von Fragen einer bestimmten Klasse zugeordnet wobei die jeweils nächste Frage von den bereits gestellten abhängig ist. Dadurch lässt sich dieser Entscheidungsprozess anschaulich als Entscheidungsbaum repräsentieren. Die Fragen befinden sich in den einzelnen Knoten des Baums und die Antwort bestimmt, welcher Verbindung zum nächsten Knoten und damit zur nächsten Frage gefolgt wird. Am Ende eines Pfads sind die Blattknoten, in denen die jeweilige Klasse definiert ist. Das Lernen eines Entscheidungsbaums ist dabei ein rekursiver Prozess, bei dem die Trainingsdaten in immer kleinere Teilmengen zerlegt werden. Ziel bei jeder Zerlegung ist es, dass sich in jeder Teilmenge so wenig wie möglich unterschiedliche Klassen befinden. Vorteil von Entscheidungsbäumen ist die leichte Diagnose, da Entscheidungsbäume für Menschen lesbar sind. Darüber hinaus werden automatisch nur die Merkmale verwendet, die tatsächlich für eine Entscheidung notwendig sind. Außerdem liefern sie tatsächliche Wahrscheinlichkeiten für ein eingegebenes Muster. Zudem sind das Lernen und die Klassifizierung extrem schnell im Vergleich zu anderen, rechenintensiven Algorithmen wie z.B. den neuronalen Netzen (Lernen dauert extrem lange, Klassifizierung ausreichend schnell). Andere Vorteile, wie z.B. dass Entscheidungsbäume auch mit fehlenden Merkmalen umgehen können, sind hier nicht von Bedeutung. Ein Nachteil von

Entscheidungsbäumen ist, dass wenn sich eines der betrachteten Features nur leicht ändert, zum Beispiel durch eine andere Schrägstellung eines Zeichens durch einen anderen Schreiber, die Klassifikation fehlschlagen kann. Daher ist es wichtig, für das Training eines Entscheidungsbaums eine möglichst große Datenmenge zur Verfügung zu haben. Bei Anwendung in der Handschriftenerkennung sollte der Verwender des Systems dieses auch selbst trainieren.

**Umsetzung**

Die Anzahl der einzelnen Richtungen des Direction-Chain-Codes bestimmen, wie bereits erwähnt, die Auflösung eines eingegebenen Musters. Diese sollte nicht zu hoch sein, da sonst auch Ausreißer betrachtet werden, aber auch nicht zu gering, da sonst wichtige Informationen, die zur korrekten Klassifizierung eines Zeichens notwendig sind, verloren gehen. Daher hängt die Anzahl auch maßgeblich von der Höhe eines Zeichens ab. Zeichen mit einer Höhe von bspw. einer halben Stufe dürfen nicht in gleich viele Teilstriche zerlegt werden wie die Zeichen mit einer Höhe von drei Stufen. Daher werden die Zeichen anhand ihrer Höhe in vier Gruppen aufgeteilt. Bei einer Höhe von einer halben Stufe wird die Eingabe in 10, bei einer Höhe von einer Stufe in 20, bei zwei Stufen in 40 und bei vier Stufen in 60 Teilstriche zerlegt, deren Richtungen anschließend den Direction-Chain-Code bilden. Zusammen mit den weiteren Merkmalen ergeben sich somit Feature Vektoren mit einer Größe von 16, 26, 46 oder 66. Dementsprechend muss auch für jede der vier Höhen ein separater Classifier verwendet werden.

Wird als Classifier ein Entscheidungsbaum verwendet, reicht es aus, je Höhe einen eigenen Entscheidungsbaum zu trainieren und für die Klassifizierung zu verwenden. Die Ausgabe ergibt hierbei eine Liste aller Klassen mit den zugehörigen Wahrscheinlichkeiten. Als Algorithmus zum Entscheidungsbaumlernen wird C4.5 (J48) aus der WEKA Bibliothek[10] verwendet. Da diese Bibliothek in Java implementiert ist, lässt sie sich nicht direkt in .NET Anwendungen verwenden. WEKA.NET[11] stellt eine Portierung der Version 3.4 dieser Bibliothek dar, welche für den Zweck der Arbeit ausreichend ist.

Bei Verwendung von neuronalen Netzen ist es nicht praktikabel, je Höhe nur ein einziges Netz zu verwenden. Um nicht zu viele Muster mit einem einzigen Netz klassifizieren zu müssen (ohne Kürzel bei einer Höhe von zwei Stufen bereits 29 Klassen), gibt es innerhalb einer Höhe für jede Klasse ein Netz, das genau diese Klasse identifizieren kann und alle anderen Klassen verwirft. Ein Zeichen wird bei der Erkennung auf alle Netze einer Höhe gegeben. Die Ausgabe jedes Netzes wird dann zusammen mit der Klasse des Netzes in einer Liste zurückgeliefert. Für die Umsetzung wird das Encog[12] Framework von Heaton Research verwendet. Dieses beinhaltet diverse Algorithmen für neuronale Netze. Konkret wird Resilient Propagation (RPROP) als Lernalgorithmus und die Sigmoid Funktion als Aktivierung verwendet.

Es sei hier nochmals darauf hingewiesen, dass die Entscheidung für die genannten Algorithmen lediglich aufgrund ihrer hohen Popularität und Einfachheit bei der Verwendung fiel und keine ausführliche Untersuchung anderer Algorithmen möglich war.

Die Liste der Ausgabe kann, egal welcher Classifier verwendet wird, einem Post Processing Schritt übergeben werden. Hier wird aufgrund des fehlenden Post Processings einfach die Klasse mit der höchsten Wahrscheinlichkeit bzw. dem höchsten Wert der Ausgabe verwendet.

## 4.5. Klassifizierung der Vokalverbindungen

Die Verbindungen zwischen zwei Konsonantenzeichen oder Kürzeln ergeben bei der DEK die Vokale. Dabei hängt die Klasse eines Vokals nicht von der Form selbst ab, denn die Verbindungen

[10] http://www.cs.waikato.ac.nz/~ml/weka/index.html
[11] http://sourceforge.net/projects/wekadotnet/
[12] http://www.heatonresearch.com/encog

sind i.d.R. gerade Linien die das Ende und den Anfang eines Segments direkt verbinden. Wichtiger ist hierbei vielmehr die Stellung der benachbarten Konsonantenzeichen. Die Unterscheidung findet nur auf Basis der Länge der Verbindung bzw. des Raums zwischen zwei Zeichen, einer möglichen Hoch- oder Tiefstellung und einer eventuellen Verstärkung des folgenden Zeichens statt. Letzteres wird anhand einer Schwelle für den vom Tablet gelieferten Drucks bereits während der Segmentierung festgestellt.

Der Raum zwischen zwei Zeichen ist dabei nicht einfach zu berechnen. Für die Berechnung muss bekannt sein, an welcher Stelle der Übergang von Zeichen zu Zeichenverbindung liegt. Denn manche Zeichen beginnen und enden nahe an der Segmentgrenze, andere nicht. Für die korrekte Klassifizierung müssen daher die Nachbarzeichen bereits klassifiziert worden sein.

Ebenso ist es bei der Hoch- und Tiefstellung notwendig, vor allem das folgende Zeichen bereits zu kennen. Die meisten Zeichen beginnen zwar bei der vollen Höhe des Zeichens, manche aber auch bei ein Drittel, halber oder zwei Drittel der Höhe (normalerweise wenn sie mit einer Schleife beginnen). Um nun feststellen zu können, ob das folgende Zeichen hoch- oder tiefgestellt geschrieben wurde, muss der Eintrittspunkt relativ zur Höhe eines Zeichens in Relation zur Grundlinie berechnet werden (siehe Abbildung 38). Im Falle eins hoch- oder tiefgestellten Zeichens muss außerdem die Grundline für den nächsten Vokal neu berechnet werden.

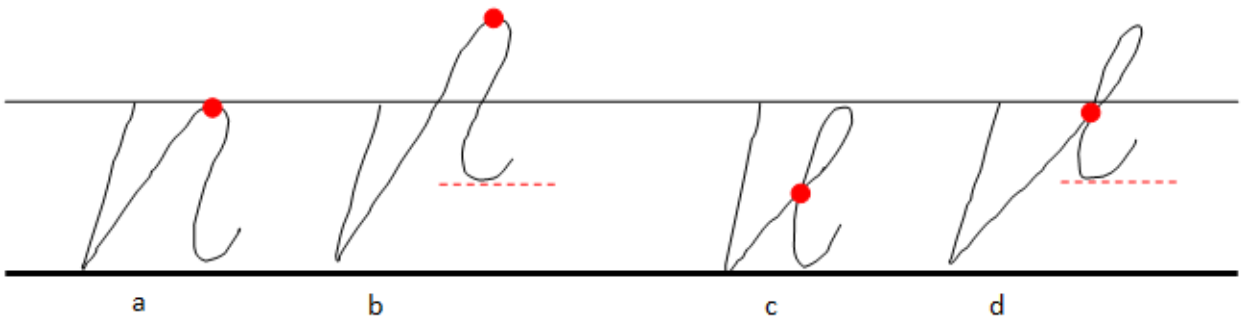



**Abbildung 38: (a) Normal- und (b) hochgestelltes "m" und (c) normal- und (d) hochgestelltes "d", jeweils nach dem Buchstaben "t". Rote Punkte entsprechen PSPs, gestrichelte Rote Linie neuer Grundlinie nach Hochstellung.**

Zusätzlich muss beachtet werden, dass manche Zeichen, wie z.B. das „s“, die Grundlinie auch ohne Hoch- oder Tiefstellung in bestimmten Fällen verschieben. Abbildung 39 zeigt die zwei Wörter „se(h)r“ und „sehen“, wobei im ersten Wort das Dehnungs-h nicht geschrieben wird. Das „r“ wird halbstufig geschrieben, daher verschiebt sich im ersten Wort die Grundlinie um eine halbe Stufe nach oben (siehe rote gestrichelte Linie). Beim zweiten Wort ist das Zeichen nach dem „s“ einstufig, daher gibt es keine Verschiebung. Soll eine Hoch- oder Tiefstellung nach einem „s“ erreicht werden, muss die neue Grundlinie ausgehend von der bereits verschobenen Grundlinie betrachtet werden. Im ersten Wort würde das Zeichen bei einer Hochstellung über die erste Stufe (durchgezogene dünne Linie), bei einer Tiefstellung hingegen auf die Standardgrundlinie (durchgezogene dicke Line) rutschen. Bei einstufigen oder höheren Zeichen nach dem „s“ erfolgt die Hoch- und Tiefstellung wie gehabt.

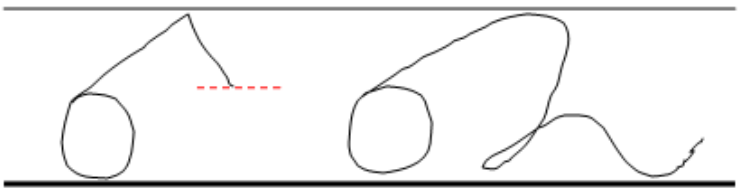


**Abbildung 39: Neue Grundlinie (rote gestrichelte Linie) nach "s" bei halbstufigen Zeichen ("se(h)r") (Beachte: Dehnungs-h wird nicht geschrieben), nicht bei einstufigen Zeichen ("sehen").**

Aus diesen Gründen kann eine Klassifizierung eines Vokals nur erfolgen, wenn die Grundlinie stimmt und beide Nachbarzeichen bekannt sind. Es empfiehlt sich daher, die Vokale von Beginn ab des Wortes in einem iterativen Prozess zu klassifizieren um bei jedem Vokal die zuvor neu berechnete Grundline verwenden zu können. Eine Klassifikation eines Vokals innerhalb eines Wortes ist also nur möglich, wenn die Grundlinie bekannt ist und hierzu müssen alle vorherigen Konsonantenzeichen und Kürzel richtig klassifiziert und die Vokalverbindungen zumindest auf Hoch- oder Tiefstellung untersucht werden.

Leider reichte die Zeit nicht aus, um auch noch die Vokalverbindungen durch einen lernfähigen Algorithmus abzudecken. Daher wurde hierfür ein einfacher Entscheidungsprozess implementiert. Hierbei werden nur die Eingangshöhen der nachfolgenden Konsonantenzeichen berücksichtigt und nicht die der vorhergehenden. Zeichen, bei denen in bestimmten Fällen eine Hoch- und Tiefstellung automatisch notwendig wird, wie z.B. beim oben angesprochenen Fall des Konsonantenzeichens für das „s“, werden demnach nicht berücksichtigt. Spezialfälle am Wortanfang und –ende hingegen werden schon beachtet. Zeichenverbindungen am Wortanfang, die nicht in der Nähe der Grundlinie beginnen, sind z.B. nur Teile des ersten Konsonantenzeichens, welche beim Segmentieren entstehen und daher nicht als Vokal klassifiziert werden dürfen (siehe roter Rahmen bei a in Abbildung 40). Am Wortende werden Vokalverbindungen maßgeblich anhand deren Länge und Höhe klassifiziert (siehe b-d in Abbildung 40).

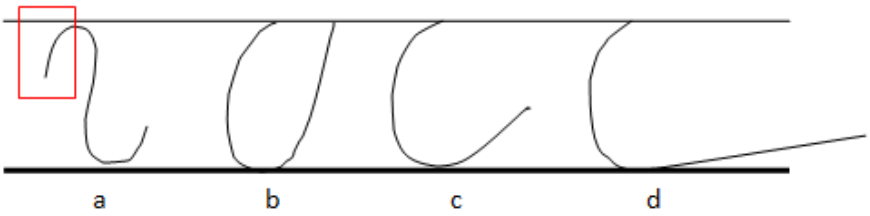



**Abbildung 40: Spezialfäll bei Verbindungen am Anfang (a) „m“ und am Ende (b) „wie“, (c) „we“, (d) „wo“.**

Die Länge der Verbindungen zwischen zwei Zeichen erfolgt durch die Grenzen der Zeichen und nicht anhand der Länge der Verbindung selber. Dies ist eine erhebliche Vereinfachung, die aber in einigen Fällen zu einer falschen Klassifikation führt. Abbildung 41 zeigt das Wort „Täter“ (Beachte: „ä“ darf als „e“ geschrieben werden). Bei relativ aufrechter Schrift (a) ist genug Platz zwischen den zwei zeichenumgebenden Flächen (rote Rechtecke). In diesen Fällen funktioniert die Vereinfachung sehr gut. Bei schrägem Schriftbild hingegen überlappen sich die Flächen und der Abstand ist somit sogar negativ. Würde für dieses Beispiel die Länge der Verbindung selber verwendet werden, entspräche Beispiel (b) evtl. bereits einer langen Verbindung und würde somit als „o“ klassifiziert werden.

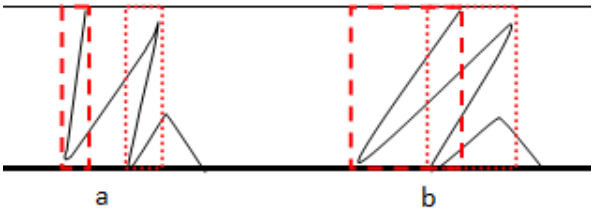



**Abbildung 41: Klassifikation von Vokalen. Probleme bei Schrägstellung der Konsonantenzeichen. Hier am Wort "Täter".**

Dieses Beispiel zeigt auch, dass bereits die vermeintlich einfache Klassifikation der Verbindungen durch viele Faktoren beeinflusst wird. Dass harte Grenzen durch Entscheidungsprozesse für solche Szenarien eher ungeeignet sind, zeigt vor allem das letzte Beispiel sehr deutlich. Ähnlich dem Klassifikationsproblem der Konsonantenzeichen wäre auch hier der Einsatz von Lernalgorithmen deutlich von Vorteil.

# 5. Ergebnisse

In diesem Kapitel werden die ausgewählten Methoden genauer untersucht um zu verifizieren, dass deren Leistungsfähigkeit für eine Handschriftenerkennung der DEK ausreichend ist. Hierzu wurden diese in Form eines Softwareprototypen umgesetzt, welcher dieser Arbeit auf einem Datenträger beiliegt. Es muss an dieser Stelle angemerkt werden, dass dieser Prototyp zum Testen der Segmentierung und der ausgewählten Merkmale dienen soll. Nicht alle Tests lassen sich über die Oberfläche steuern und einstellen, manche wurden nur im Quellcode für die Dauer der Tests eingebaut und hinterher deaktiviert. Das letztendlich erzielte Ergebnis lässt sich jedoch mit der Software ohne Eingriffe im Quellcode verwenden. Da es bei der Arbeit primär um die Entwicklung und Verifizierung der Methode geht und nicht um die Entwicklung einer Software, wird diese in Anhang A nur kurz beschrieben.

## 5.1. Segmentierungsalgorithmus

Die Idee, die hinter dem Segmentierungsalgorithmus steht, nur an den Aufstrichen zu segmentieren, die nicht in einer Schleife liegen, ist meiner Meinung nach für die DEK der beste Ansatz. Jedoch hat die Umsetzung an einigen Stellen Probleme gezeigt. Da, wie in Kapitel 4.3.2 gezeigt, für das Erkennen der Aufstriche die Koordinaten verwendet werden, kann es z.B. bei nicht idealer Handschrift (was eine Handschrift in den seltensten Fällen ist) dazu kommen, dass tatsächliche Segmentierungspunkte nicht erkannt werden. So zum Beispiel, wenn die Verbindung zwischen zwei Kosonantenzeichen leicht in die negative y-Richtung verläuft, was vor allem bei Tiefstellung von Konsonanten und Kürzeln häufiger vorkommt. Es wäre daher sinnvoller, anstatt der Richtung die Extremwerte zu berechnen. Segmentiert wird dann an allen Extremwerten (Minima und Maxima), die außerhalb einer Schleife liegen. Dabei muss allerdings darauf geachtet werden, dass an Schleifen i.d.R. am Ausgang, Eingang oder an beiden auch segmentiert werden muss (siehe Abbildung 31 in Kapitel 4.3.2). Eine Möglichkeit dies zu erreichen, könnte das Berechnen der Tangente im Ein- bzw. Austrittspunkt sein. Wenn diese eine bestimmte Steigung gegenüber der x-Achse aufweist, muss hier ebenfalls segmentiert werden. Abbildung 42 zeigt das Ablaufdiagramm des verbesserten Segmentierungsalgorithmus.

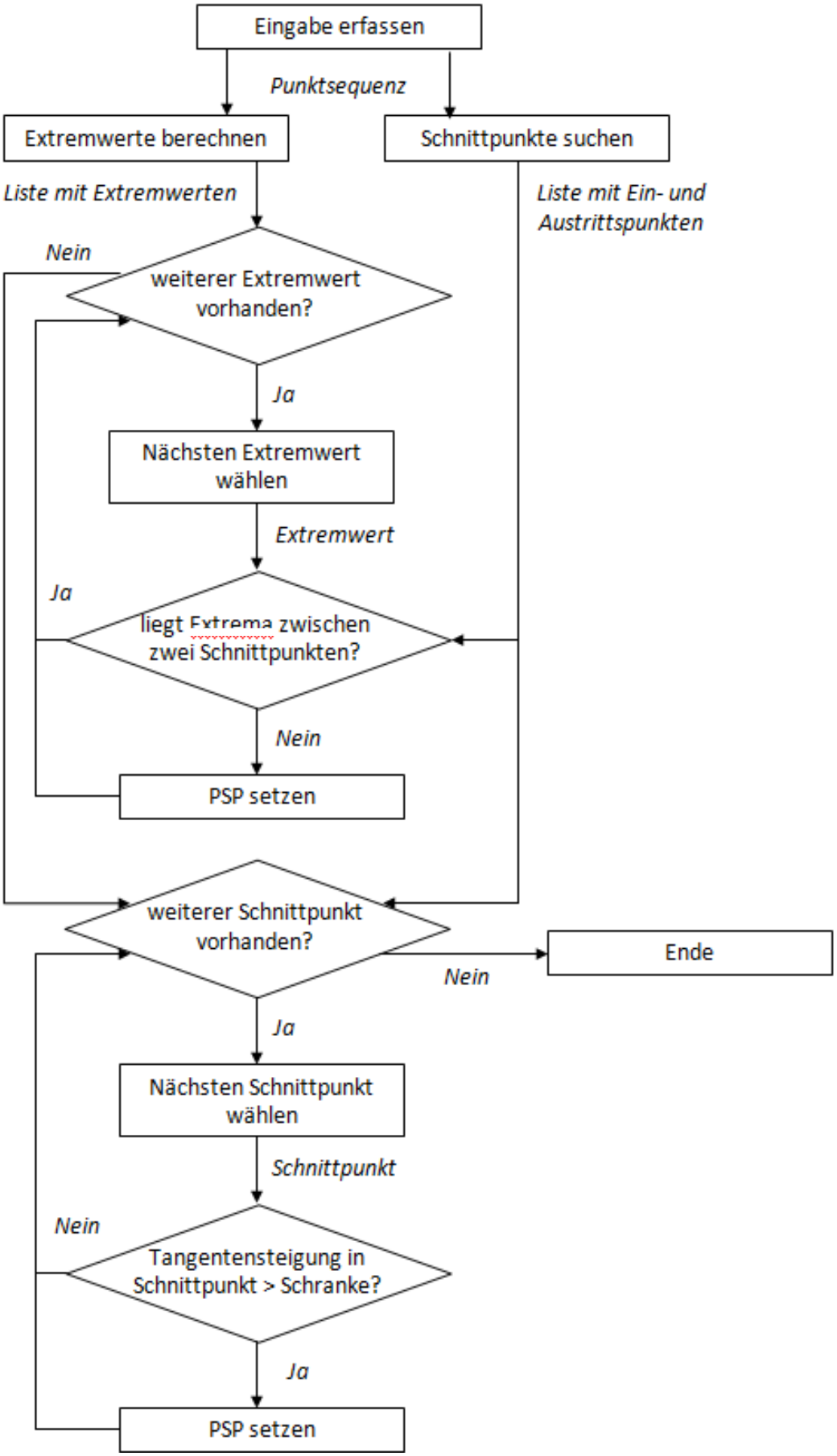



**Abbildung 42: Verbesserter Segmentierungsalgorithmus.**

Bei der Implementierung sollte eine Methode gewählt werden, die nicht auf die Microsoft INK API angewiesen ist. Das Verwenden sowohl der erkannten Punkte als auch der Bezier Darstellung, wie es bei der Umsetzung gemacht wurde, ist zwar naheliegend und einfacher, birgt aber auch Schwachstellen. So ist es nicht immer zufriedenstellend möglich, zu einem erkannten Punkt den passenden Punkt auf der Bezierkurve zu finden. In seltenen Fällen kann es vorkommen, dass eng aneinander liegende Linien vertauscht werden. Dadurch ist die Zuordnung eines Schnittpunktes, der mittels INK API erkannt wurde, nicht immer korrekt. Um sowohl solche Probleme in den Griff zu bekommen als auch den Algorithmus auf andere Plattformen portierbar zu machen, empfiehlt es sich nur rein mit den erkannten Punkten zu arbeiten. Da diese aber auch bei OLCR nicht frei von Rauschen sind, müsste zunächst ein Preprocessing durchgeführt werden. Dabei sollten die Punkte geglättet werden, wobei unwichtige Punkte gelöscht und evtl. wichtige fehlende Punkte ergänzt werden könnten. Es muss aber sichergestellt werden, dass die gewählte Methode scharfe Übergänge von einer Verbindung zu einem Konsonantenzeichen nicht glättet, da diese Übergänge entscheidend für die spätere Erkennung sind. Auch andere Autoren beschreiben ein solches Vorgehen, siehe z.B. [18] oder [34].

Für die Schnittpunktberechnung, die ohne die INK API manuell durchgeführt werden muss, kann z.B. der Ansatz von Ma und Leedham verwendet werden, den sie für die Schleifenerkennung in Renqun [16] und Pitman [18] verwenden. Dabei wird die Eingabe zuerst in einen Freeman chain code mit 16 Richtungen umgesetzt. Anschließend werden Regionen immer dort gebildet, wo sich der Richtungscode von aufeinanderfolgenden Punkten nicht ändert. Für sich überlappende Regionen wird schließlich der Schnittpunkt berechnet (siehe Abbildung 43).

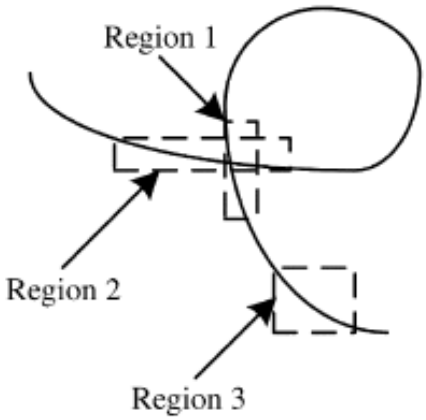



**Abbildung 43: Berechnung von Schnittpunkten mithilfe von überlappenden Regionen [18].**

Eine weitere Verbesserung, sowohl des verwendeten Ansatzes als auch bei einer Überarbeitung der Segmentierung, wäre die Erkennung des Zeichens für „l“ bereits in der Preprocessing Phase. Dieses ist in der DEK ein kleiner Kreis bzw. eine Schleife. Probleme hierbei treten vor allem bei der korrekten Erkennung von Schleifen auf. Da die Distanz zwischen Ein- und Austrittspunkt bei diesem Zeichen sehr gering sind, darf die Schwelle zwischen falschen und echten Schleifen (siehe Kapitel 4.3.3) nicht sehr hoch ausfallen. Daher kann es passieren, dass auch Schleifen als solche erkannt werden, die nur versehentlich durch eng aneinander liegende Linien verursacht wurden (siehe Abbildung 44). Würde die Erkennung für das Zeichen „l“ bereits vorab erfolgen, könnte die Schwelle weiter angehoben und so die Anzahl der falsch erkannten Schleifen reduziert werden.

**Abbildung 44: Fälschlicherweise erkannte Schleife am Beispiel des Konsonantenzeichens für „w“.**

Außerdem definiert die Wiener Urkunde [2] in §4 Absatz 4, dass das die Schleife für das „l“ unter bestimmten Umständen vorgezogen und somit als Teil des vorherigen Konsonantenzeichens geschrieben werden darf, wie in Abbildung 45 illustriert.

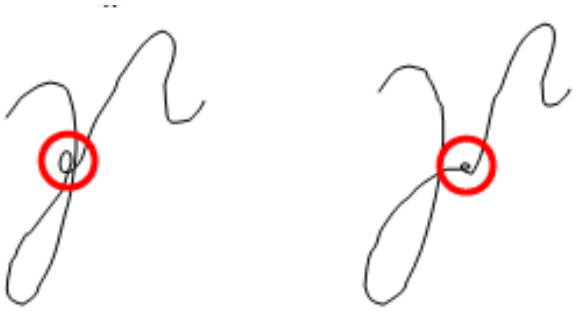


**Abbildung 45: Wort „schlimm" mit und ohne vorgezogenem „l".**

Damit würde mit dem vorgestellten Ansatz keine Segmentierung der beiden Zeichen erfolgen. Um solche Fälle dennoch erkennen zu können, müssten Muster als separate Klasse zum Training der Erkennung bereitgestellt werden, was die Komplexität erhöhen würde. Auch dieses Problem würde sich durch eine Voraberkennung beheben lassen. Da das „l“ eine besondere Form aufweist, wäre eine Möglichkeit für die Erkennung in der Preprocessing Phase die Suche von kleinen Punkthaufen, wo alle Punkte in ein Quadrat mit bestimmter Größe passen und sich die Eingabe mit sich selbst schneidet. Leider war die Implementierung im Rahmen der Arbeit aus Zeitgründen nicht mehr möglich. Es gilt jedoch auch zu beachten, dass die Erkennung von vorgezogenen Konsonantenzeichen nicht zur Zielsetzung der Arbeit gehört (siehe Kapitel 1.2).

Rückblickend war es nicht sinnvoll, sich für die Verwendung der Bezier-Splines zu entscheiden. Anders als vermutet, wurde die Implementierung dadurch nicht vereinfacht, sondern an vielen Stellen eher komplizierter, was auch zu Lasten der Robustheit geht. Trotz allem kann auch die in dieser Arbeit verwendete Umsetzung der Segmentierung als gut bezeichnet werden. Die angesprochenen Fälle, in denen sie fehlschlägt, kommen nicht sehr häufig vor.

Dennoch muss erwähnt werden, dass der Algorithmus in seiner Gesamtheit nur von mir getestet wurde und dies auch nur während der Bedienung. Es wurden keine gesonderten Testfälle erstellt, die die Korrektheit der Segmentierung verifizieren könnten. Daher ist auch keine klare Aussage zu treffen, wie stark die beschriebenen Probleme im praktischen Einsatz auftreten. Das Fehlen von Testfällen liegt einerseits daran, dass mir keine Personen mit Kenntnissen der DEK bekannt sind und wie bereits an anderer Stelle erwähnt wurde, jegliche Kontaktaufnahme zu Stenographen Vereinen im Sande verlief. Andererseits ist das Erstellen einer Datenbank zur Verifikation der Segmentierung sehr aufwändig. Für andere Sprachen gibt es solche Datenbanken bereits, für die DEK müsste diese jedoch selbst aufgebaut werden, was im zeitlichen Rahmen dieser Arbeit nicht möglich war.

Erschwerend kommt beim Testen der Anwendung die Abhängigkeit zur Microsoft INK API hinzu. Da der Segmentierungsalgorithmus auf das *Stroke* Objekt der API angewiesen ist, kann dieser in seiner Gesamtheit nur mit einem solchen getestet werden. Das manuelle Erstellen (also nicht durch Eingabe an einem Tablet) solcher Objekte ist aber nur eingeschränkt möglich. Die Druckstärke sowie andere notwendige Eigenschaften lassen sich hierbei nicht vorgeben.

Um aber Teile des Algorithmus zu testen, wurden z.B. für die Berechnung von Winkeln Testfälle erstellt um deren Korrektheit zu verifizieren. Hierbei wurden fast ausschließlich Grenzwerttests eingesetzt, da es schlicht zu aufwändig wäre, bei einer Winkelberechnung jede Kombination zu testen. Daher wird bei Grenzwerttests angenommen, dass die Algorithmen sich zwischen zwei Grenzen ebenfalls korrekt verhalten und es ausreicht, nur die Grenzwerte und problematische Konstellationen zu testen.

## 5.2. Klassifizierung der Konsonantenzeichen

Anders als in Kapitel 4.4.4 beschrieben, konnte in dieser Arbeit aus zeitlichen Gründen nicht jedes neue Feature auf seinen Informationsgehalt geprüft werden, da das Training eines Classifiers je nach gewählten Algorithmus sehr aufwändig sein kann und mir nur begrenzte Rechenkapazitäten zur Verfügung standen (siehe Training des neuronalen Netzes im weiteren Verlauf des Kapitels). Daher wurden neue Features nicht auf den gesamten Daten, sondern nur in kurzen Einzeltests verifiziert.

Zum Testen der Klassifizierung wurde eine Datenbank angelegt, in der für jede Klasse manuell etwa 100-mal das entsprechende Zeichen eingegeben wurde. Insgesamt enthält die Datenbank 6.218 Einträge für 63 Klassen. 55 Klassen entsprechen den Konsonantenzeichen der Wiener Urkunde [2]. Weitere acht Klassen waren notwendig, um Zeichen zu modellieren, die je nach folgendem Zeichen mit oder ohne Schleife auslaufen können (siehe erste zwei Zeichen von Abbildung 46). Abbildung 46 zeigt einige Einträge der Datenbank. Um die Ein- und Austrittswinkel bei der Segmentierung korrekt berechnen zu können, wurden diese mit Pseudoverbindungen versehen, die nicht zum Zeichen selber gehören.

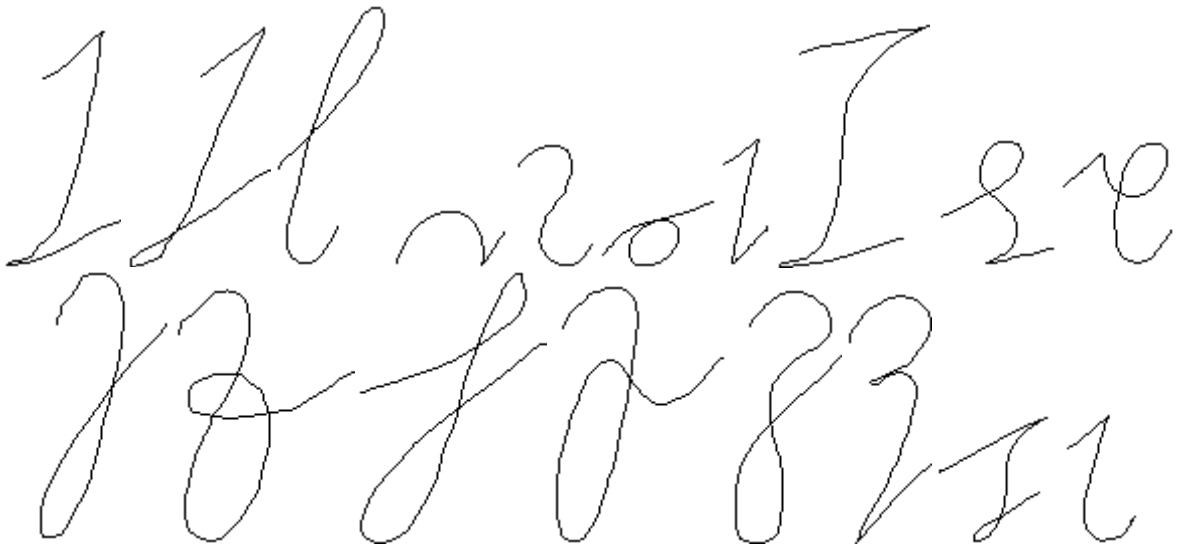


**Abbildung 46: Einige Muster der Datenbank.**

Für die Eingabe wurde ein Wacom PL-550 (Cintiq 15X) verwendet, das eine Auflösung von 508 dpi (200 Linien/cm) und eine Sample Rate von 206 Punkten in der Sekunde ermöglicht. Dies ist nicht mehr ganz zeitgemäß, denn bereits moderne Consumertablets erreichen Auflösungen von

2540 dpi. Dies ist auch der Grund, weshalb beim Anlegen der Datenbank die Zeichen sehr genau eingegeben werden mussten. Beim ersten Versuch wurde darauf nicht sonderlich geachtet. Dabei traten vor allem Probleme bei der Berechnung der Ein- und Austrittswinkel auf, welche für die Zeichenklassifikation sehr wichtig sind. Es zeigte sich, dass bei schneller Eingabe und geringer Stufengröße es schwierig ist, Rundungen bei Übergängen von Verbindungen in Konsonantenzeichen so zu schreiben, dass diese vom Tablet nicht als rechte Winkel erkannt wurden. Gerade bei halbstufigen Zeichen wie dem „r“ und dem „n“ ist somit eine Unterscheidung im weiteren Verlauf kaum möglich. Abbildung 47 soll dieses Problem verdeutlichen.

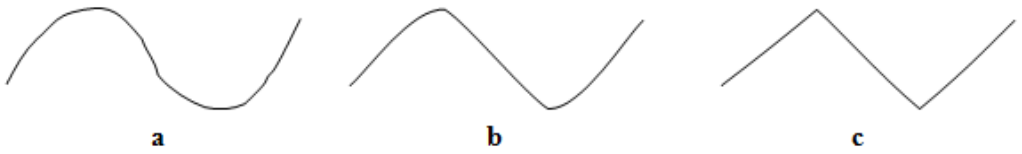



**Abbildung 47: halbstufiges "n" (a), halbstufiges "r" (c) und vom Tablet geliefertes Muster (b).**

Es wird angenommen, dass diese Probleme mit moderner Hardware weitaus weniger ins Gewicht fallen, dies konnte jedoch aufgrund fehlender Hardware nicht verifiziert werden. Es gilt hier jedoch zu berücksichtigen, dass diese Probleme letztlich nur durch die verwendete Hardware verursacht werden und nicht auf die Leistungsfähigkeit des gefunden Algorithmus bezogen werden dürfen.

Um die optimale Struktur der jeweiligen neuronalen Netze zu finden wurde eine Kreuzvalidierung implementiert. Dabei wurde jedes Netz einer Klasse in zehn Durchgängen zufällig mit etwa 90% der Daten trainiert und mit den verbleibenden 10% getestet. Die Auswahl der Trainings- und Testdaten erfolgten hierbei bei jedem neuen Durchgang erneut. Auf diese Weise wurde automatisch jede Kombination von ein oder zwei verdeckten Schichten mit einer Anzahl von 1/5 der Eingabeneuronen bis hin zur Anzahl der Eingabeneuronen je verdeckter Schicht getestet. Allein für die Zeichen mit einer Höhe von einer halben Stufe entstehen dadurch $8 * 10 * 2 * (17 - 3) = 2240$ zu trainierende Netze[13]. Die Netze wurden dabei so lange trainiert, bis der Fehler auf den Trainingsdaten weniger als 0,1% betrug. Da ein Netz mit zufälligen Gewichten initialisiert wird, kann es vorkommen, dass ein Netz gegen einen Fehler größer als 0,1% konvergiert. Daher wird die Berechnung nach spätestens 1000 Iterationen abgebrochen (Tests haben ergeben, dass eine Konvergenz meist noch vor 500 Iterationen erreicht wird). In solchen Fällen wird das Netz mit anderen zufälligen Gewichten initialisiert und erneut trainiert. Da es auch aufgrund der Struktur möglich ist, dass kein Fehler kleiner als 0,1% auf den Trainingsdaten erreicht werden kann, wird dieser Vorgang maximal zehnmal wiederholt. Wurde bis dahin kein kleinerer Fehler erreicht, wird das letzte gelernte Netz verwendet.

Anschließend wurde der Fehler auf den Testdaten gemessen, wobei die falsch negativen Fehler (bei denen ein zugehöriges Muster als nicht zugehörig klassifiziert wird) hierbei größere Bedeutung haben als die falsch positiven (wo nicht zugehörige als zugehörige klassifiziert werden). Denn selbst wenn ein Netz einer falschen Klasse ein Muster mit einer Ausgabe von 0,98 als erkannt feststellt, so kann das Netz, zu dem die Klasse tatsächlich gehört, immer noch eine Ausgabe von 0,99 oder gar 1,0 aufweisen. Damit würde das Muster im anschließenden Entscheidungsprozess trotz falsch positiver Klassifikation einzelner Netze dennoch der richtigen Klasse zugeordnet. Die Gewichtung der Fehler wird automatisch erreicht, indem sie auf die jeweilige Anzahl der Testdaten bezogen werden. D.h. falsch negative Fehler werden auf die Anzahl der zugehörigen Muster (etwa 10% von 100 der jeweiligen Klasse) und falsch positive auf die Anzahl der nicht zugehörigen Muster (ebenfalls etwa 10% von 100 allerdings von allen anderen Klassen) bezogen.

[13] 8 Klassen mit je 10 Durchgänge bei 2 verschiedene Konfigurationen der verdeckten Schichten, 17 Eingabeneuronen, 3 entspricht 1/5 der Eingabeneuronen, also 14 Konfigurationen für die Anzahl der Neuronen je verdeckter Schicht

Das Training und Testen der einzelnen Netzstrukturen inkl. der Kreuzvalidierung dauerte insgesamt etwa 22 Tage auf einem MacBook Pro mit einem 2,53GHz Intel Core 2 Duo P8700 Prozessor unter Windows 7. Als Grafikkarte ist eine NVIDIA GeForce 9400M verbaut und OpenCL[14] ist installiert. Da die Encog Bibliothek OpenCL unterstützt[15], konnte auch die GPU der Grafikkarte verwendet werden. Abbildung 48 bis Abbildung 51 zeigen die Ergebnisse durch die Angabe der Fehler auf den Testdaten jeweils für eine der vier verschiedenen Zeichenhöhen. Die optimale Struktur eines Netzes ergibt sich durch den geringsten Fehler.

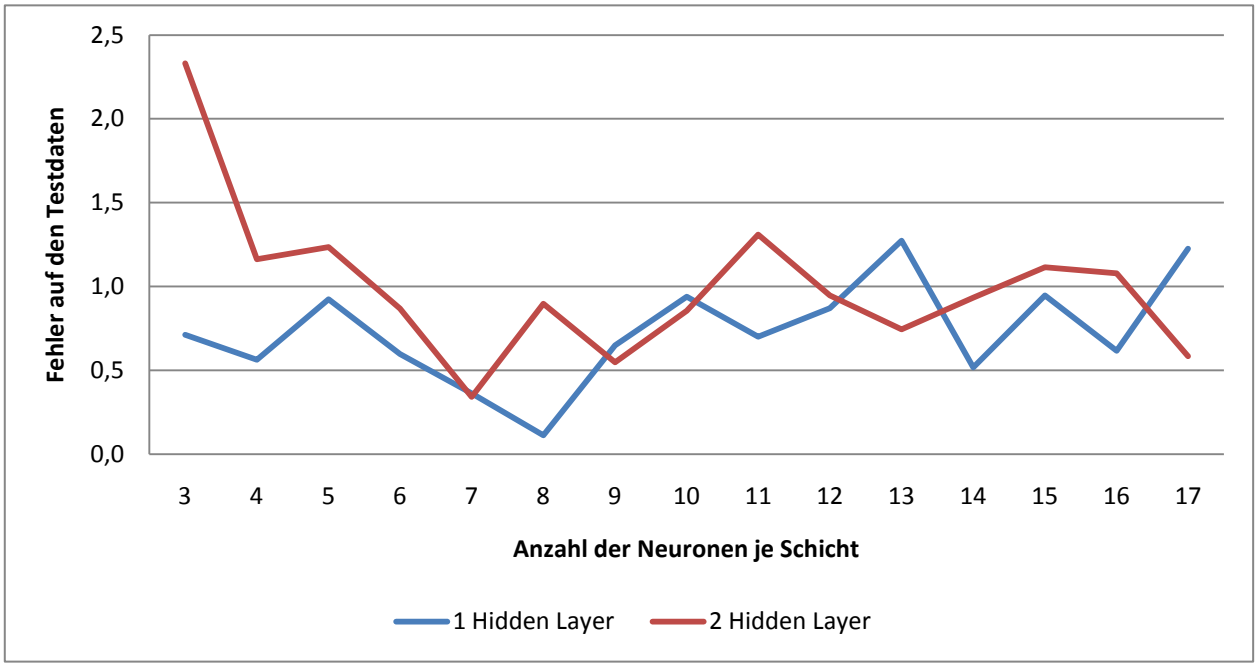



**Abbildung 48: Fehler auf den Testdaten bei halbstufigen Zeichen.**

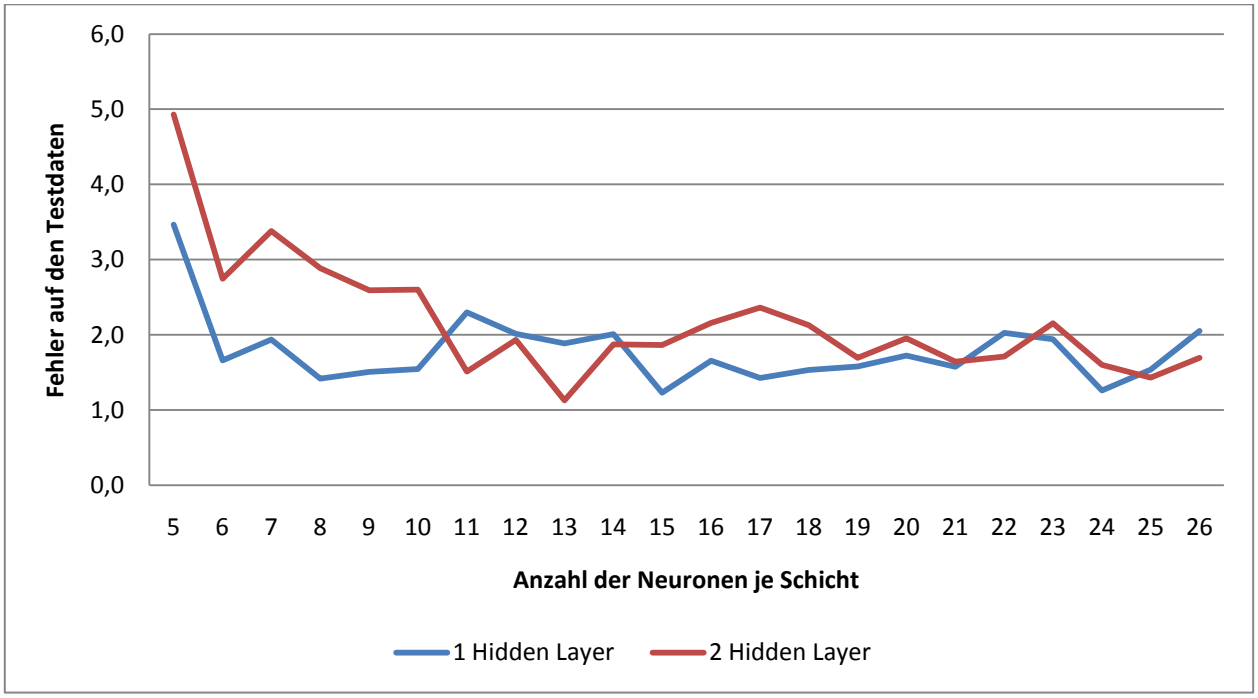



**Abbildung 49: Fehler auf den Testdaten bei einstufigen Zeichen.**

[14] http://de.wikipedia.org/wiki/OpenCL
[15] http://www.heatonresearch.com/encog/opencl

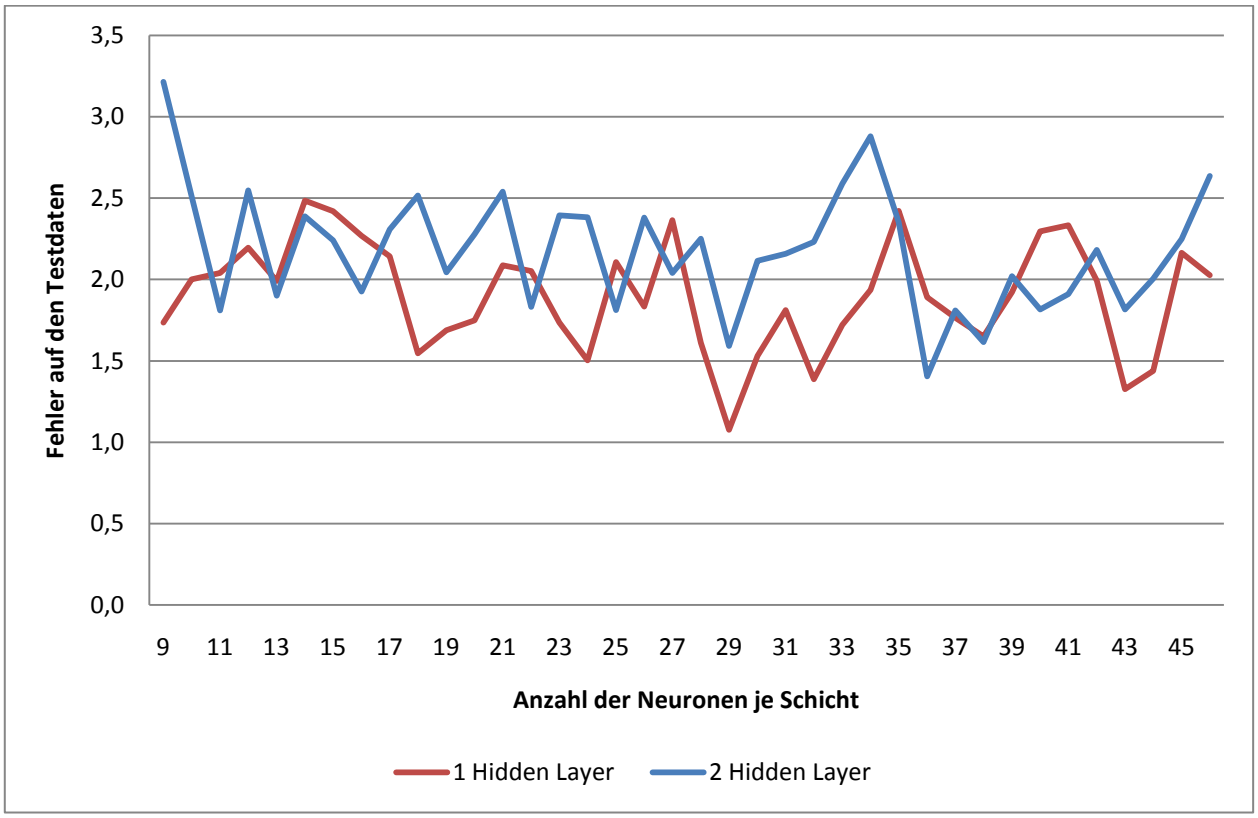



**Abbildung 50: Fehler auf den Testdaten bei zweistufigen Zeichen.**

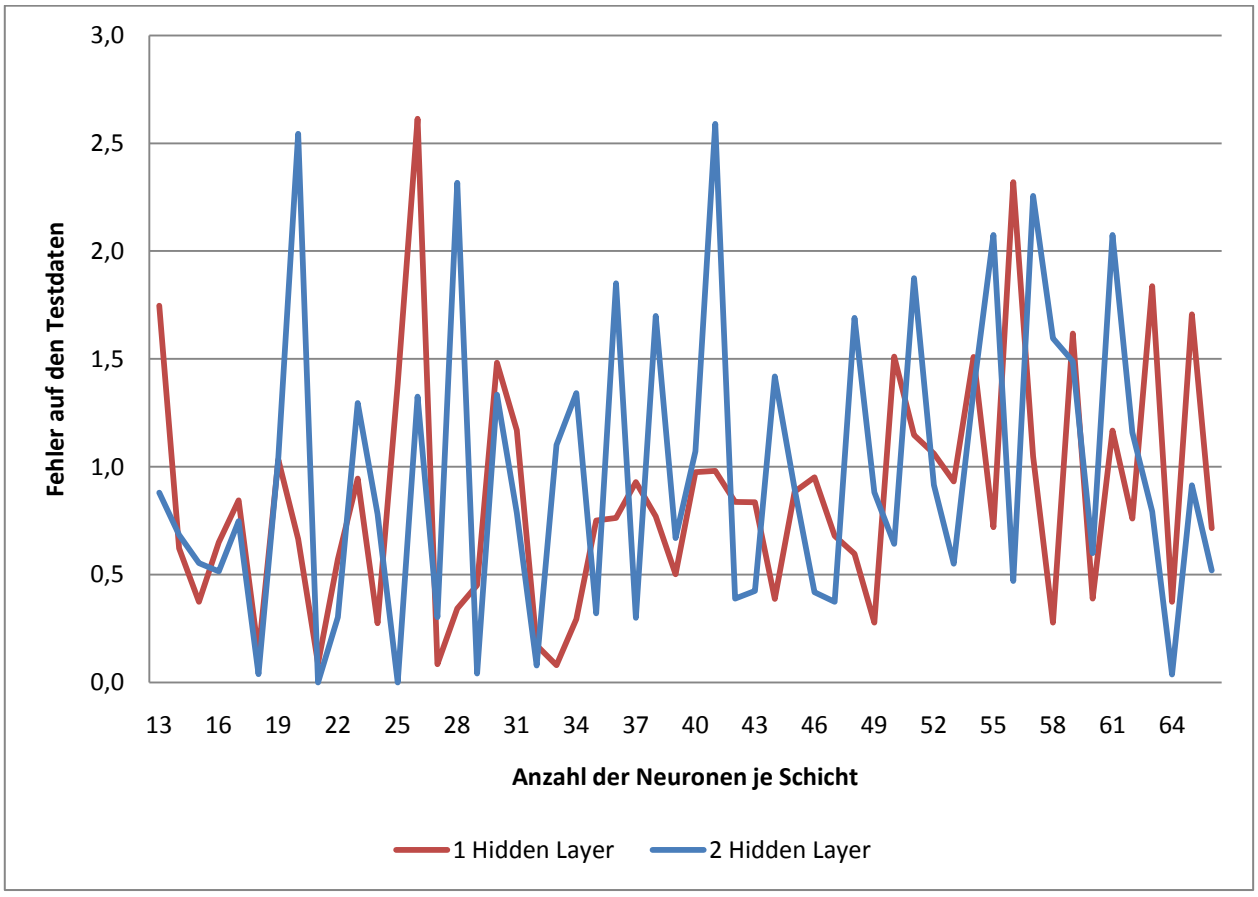



**Abbildung 51: Fehler auf den Testdaten bei dreistufigen Zeichen.**

Bei halbstufigen Zeichen ist demnach eine verdeckte Schicht mit acht Neuronen die beste Struktur. Damit ergibt sich ein Fehler von 0,11%. Für Zeichen mit einer Höhe von einer Stufe liegt der geringste Fehler von 1,12% bei zwei verdeckten Schichten mit je 13 Neuronen. Ein Fehler von 1,08% können bei zweistufigen Zeichen mit einer verdeckten Schicht mit 29 Neuronen erreicht werden. Dreistufige Zeichen können sogar mit einem Fehler von 0,0% bei zwei Schichten mit je 21 Neuronen erkannt werden. Im Umkehrschluss bedeuten diese Fehler eine korrekte Erkennungsrate zwischen 98,88% und 100% bei nicht gelernten Mustern.

Ein Test mit der kompletten Datenbank liefert je nach Training unterschiedliche Ausgaben, denn im Gegensatz zum Entscheidungsbaum schwanken die Netze, je nach dem mit welchen zufälligen Werten die Gewichte eines Netzes vor dem Training initialisiert wurden. Ein Trainingsdurchlauf mit der kompletten Datenbank und der ermittelten, idealen Struktur der Netze ergab, dass von den 6.218 Mustern 6.201 korrekt und lediglich 17 Einträge falsch klassifiziert werden. Dies entspricht einer Erkennungsrate von 99,72%. Dabei gilt ein Muster als falsch, wenn die zugehörige Klasse in der vom Classifier gelieferten Liste nicht die höchste Wahrscheinlichkeit aufweist (TOP1). Werden auch die Plätze der 2. wahrscheinlichsten Klassen für das Muster berücksichtigt (TOP2), ergeben sich nur noch 11 Fehler usw. (siehe Tabelle 1). Abbildung 52 zeigt die falsch klassifizierten Zeichen in der zur Tabelle zugehörigen Reihenfolge, sprich die ersten sechs Zeichen befanden sich unter den TOP2 Kandidaten, die nächsten drei unter den TOP3 usw.

**Tabelle 1: Fehler bei Betrachtung weiterer Kandidaten.**

| **TOP** | **1** | **2** | **3** | **4** | **6** | **8** | **9** | **22** | **23** | **29** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Fehler** | 17 | 11 | 8 | 7 | 6 | 4 | 3 | 2 | 1 | 0 |

Es ist nicht ungewöhnlich auch die Kandidaten zu betrachten, deren Wahrscheinlichkeit zwar nicht die höchste, aber eine der höchsten ist. Diese Angabe ist dann wichtig, wenn ein intelligenter Post Processing Schritt verwendet wird, der z.B. aufgrund der Nachbarzeichen eine höhere Wahrscheinlichkeit für TOP2 Zeichen berechnet als für TOP1 Kandidaten.

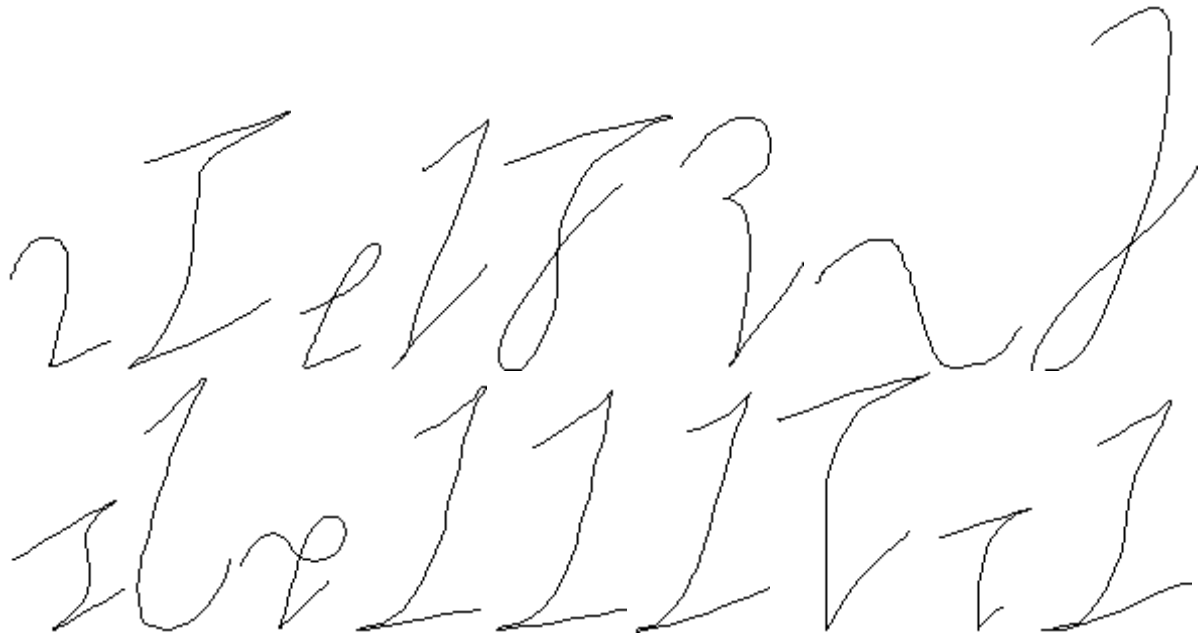


**Abbildung 52: Vom neuronalen Netz falsch klassifizierte Muster.**

Bei Verwendung eines Entscheidungsbaums als Classifier ist das Training durch die Verwendung der WEKA Bibliothek denkbar einfach. Diese bietet eine automatische Kreuzvalidierung und wählt dabei den Baum mit dem geringsten Fehler auf den Testdaten selber aus. Das Training aller Klassen dauert hierbei nur 13 Sekunden auf derselben Hardware, die auch für die neuronalen Netze verwendet wurde (siehe oben). Die Ausgabe des Algorithmus ist in Anhang B zu finden und zeigt den maximalen Fehler von 1,8544%, bei Zeichen mit einer Höhe von zwei Stufen. Es werden also Zeichen aller Klassen mit einer Rate von über 98% erkannt.

Bei genauerer Analyse der Ausgabe zeigt sich für halbstufige Zeichen, dass diese maßgeblich anhand der Richtungen am Anfang und Ende eines Zeichens klassifiziert werden. Ein- und Austrittswinkel (Merkmal a1 und a2 im Baum in Anhang B) sowie die Breite eines Zeichens (a0) spielen bei der Detailklassifikation eine wichtige Rolle. Da es nur Zeichen mit einer Schleife gibt, findet nur die Schleife in der Mitte (a4) Beachtung, Schleifen am Anfang (a3) und Ende (a5) eines Zeichens hingegen nicht. Interessanterweise werden "l" nur mittels Richtungen und Eintrittswinkel klassifiziert, die Schleife scheint hierbei keine Rolle zu spielen. Dieser Fakt ist insofern eine nützliche Information, da bei der Segmentierung das „l“ eine große Rolle bei der Schleifenerkennung spielt (siehe Kapitel 5.1), was demnach nicht notwendig wäre, da es für die Erkennung scheinbar unwichtig ist, ob eine Schleife erkannt wurde.

Für die Klassifikation von einstufigen Zeichen werden vor allem die Richtungen am Anfang und Ende und die Position von Schleifen berücksichtigt. Die Richtungen inmitten eines Zeichens spielen eher eine untergeordnete Rolle, werden aber dennoch benötigt. Ein- und Austrittswinkel sind nur für wenige Detailentscheidungen notwendig und die Breite eines Zeichens ist irrelevant. Insgesamt weist der Entscheidungsbaum aber keine Überraschungen auf.

Ebenso ist der Baum für zweistufige Zeichen wie erwartet. Vor allem die Schleifen spielen eine große Rolle. Von den Richtungen werden 23 von 40 im Baum verwendet. Dabei sind Anfänge und Enden aber ebenso wichtig wie die Richtungen innerhalb eines Zeichens. Durch die gleichmäßige Verwendung kann hier kaum eine Richtung als Merkmal vernachlässigt werden. Ebenso werden auch hier erwartungsgemäß die Ein- und Austrittswinkel benötigt. Die Breite wird in drei Fällen für die Unterscheidung zwischen zwei Klassen verwendet.

Für Zeichen mit einer Höhe von drei Stufen werden außer der Breite für eine Unterscheidung zwischen zwei Klassen lediglich die Richtungen verwendet und auch hier nur vier von 60. Anstatt die Schleifen zu Beginn eines Zeichens zu verwenden, was bereits eines der drei Zeichen eindeutig klassifizieren würde, findet die Klassifikation anhand der Strichrichtung zu Beginn des Zeichens statt. Bei genauerer Betrachtung scheint dies ebenso gut wie die Schleifen, denn innerhalb einer Schleife können Richtungsvektoren auftreten, die es nur in Schleifen geben kann. Daher leitet der Baum aus der Richtungsinformation automatisch das Vorhandensein einer Schleife ab und kann danach entscheiden. Interessanterweise wird zu drei Richtungen am Anfang der Zeichen auch ein einziges Richtungsmerkmal am Ende der Zeichen verwendet. Das ist insofern komisch, da die Zeichen alle auf dieselbe Weise auslaufen und das Ende somit keine Rolle spielen sollte.

Die Analyse der gelernten Entscheidungsbäume zeigt, dass zumindest für diesen Classifier alle extrahierten Merkmale von Relevanz sind, wenn auch bei manchen Klassen mehr bei manchen weniger. Es gibt aber kein Merkmal, das für alle Klassen bedenkenlos gestrichen werden kann. Lediglich die Auflösung für den Direction-Chain-Code könnte eventuell noch feinjustiert werden.

Tests der gesamten Datenbank auf dem gelernten Entscheidungsbaum führen zu 25 falsch und 6.193 korrekt klassifizierten Mustern, was einer Erkennungsrate von 99,5979% entspricht. Da immer der optimale Baum gelernt wird, unterscheiden sich diese Ergebnisse auch nicht von Lauf zu Lauf. Betrachtet man hier nicht nur die wahrscheinlichsten Kandidaten, sondern auch weitere, ergibt sich bereits bei den TOP2 Kandidaten nur noch ein Fehler und bei den TOP3 keine Fehler und somit eine Erkennungsrate von 100%, denn 24 falsch klassifizierten Muster befinden sich auf

Platz 2 der vom Classifier gelieferten Liste und nur ein Fehler auf Platz 3. Abbildung 53 zeigt die falsch klassifizierten Muster.

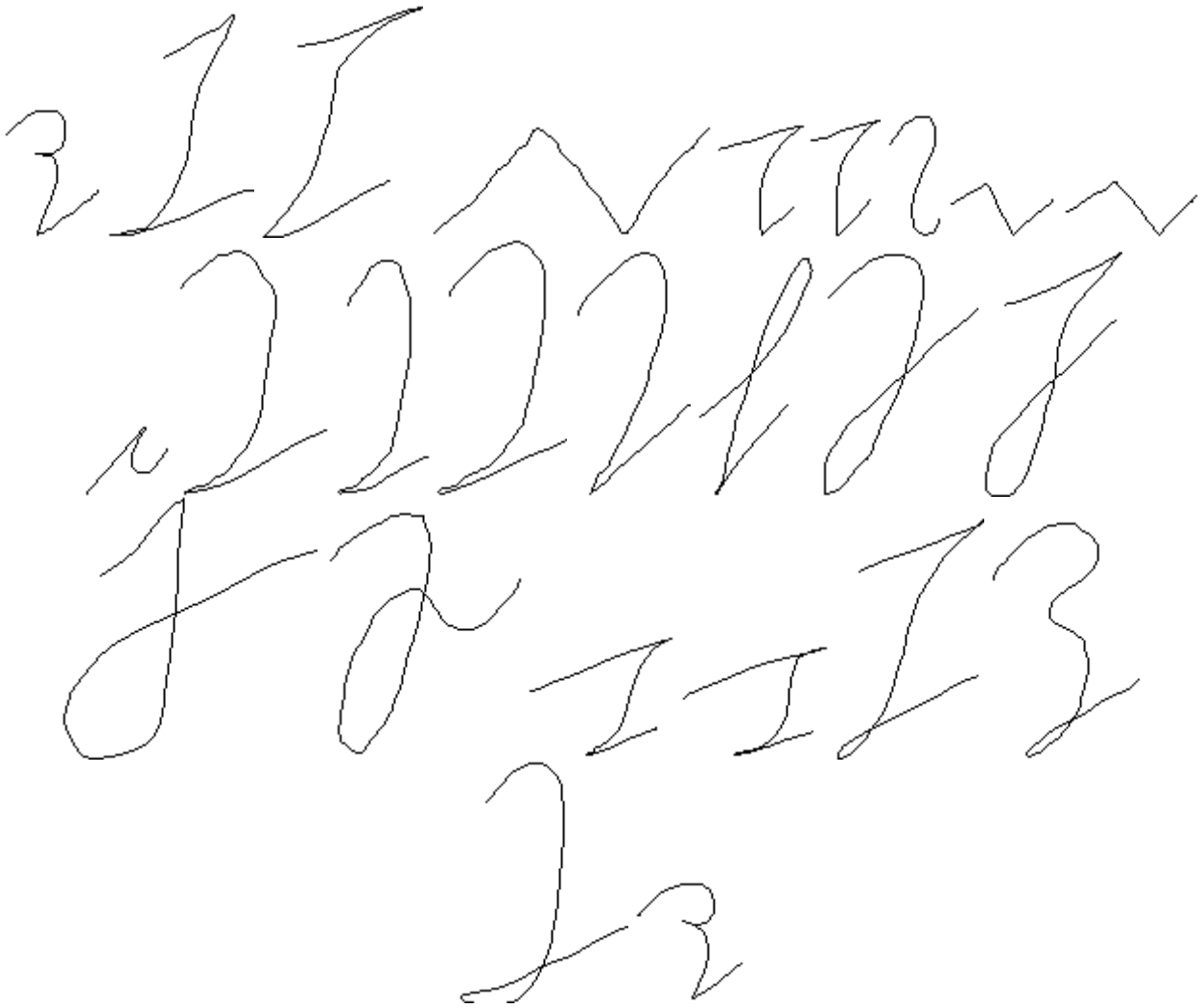


**Abbildung 53: Vom Entscheidungsbaum falsch klassifizierte Muster.**

Wie bei der Segmentierung muss auch bei den Ergebnissen der beiden Classifier darauf hingewiesen werden, dass das System nur von einem Schreiber trainiert und getestet wurde. Es ist zu erwarten, dass die Erkennungsrate bei steigender Schreiberanzahl sinkt. Da eine Eingabe von längeren Texten für einen Stenographen jedoch kein Problem darstellt, ist es durchaus denkbar, dass ein Schreiber ein System nur für sich in kurzer Zeit trainiert und dann ähnliche Erkennungsraten erreicht werden.

Ungeachtet der Vor- und Nachteile der verwendeten Classifiern belegen die Tests jedoch, dass die ausgewählte Methode eine sehr hohe Erkennungsrate bei beiden Classifiern erreicht. Die erzielten Ergebnisse können daher als erfolgreich gewertet werden und bestätigen die Leistungsfähigkeit der extrahierten Merkmale.

## 5.3. Klassifizierung der Vokalverbindungen

Einige Probleme der gewählten Umsetzung wurden bereits in Kapitel 4.5 angesprochen. Es sollte klar geworden sein, dass auf die Korrektheit der Umsetzung keinerlei Anspruch erhoben wird und diese ganz klar als Notlösung betitelt werden kann. Die Probleme entstehen hier hauptsächlich durch viele zu beachtende Faktoren und Klassen, die nur schwer linear zu trennen sind. Sprich, ein Entscheidungsprozess anhand manuell fest vorgegebener Werte wird keine adäquate Ergebnisse liefern. Die Parameter für den Prozess wurden zwar so angepasst, dass nach einer kurzen Eingewöhnungsphase für den Schreiber etwas mehr als die Hälfte aller Eingaben korrekt klassifiziert werden. Es kann jedoch nicht das Ziel sein, dass der Schreiber sich auf das System einstellen muss, sondern umgekehrt. Daher wird auch hier ein lernfähiger Algorithmus empfohlen, der anhand der in Kapitel 4.5 angedeuteten Merkmale und im Kontext der umgebenden Konsonantenzeichen eine Entscheidung trifft.

# 6. Zusammenfassung und Ausblick

Die Worterkennung wurde untersucht und weitere Anstrengungen in diese Richtung eingestellt. Der gefundene Ansatz zur Segmentierung der DEK kann jedoch als Erfolg betrachtet werden. Die umzusetzenden Konsonantenzeichen laut Zielsetzung in Kapitel 1.2 wurden alle beachtet und können problemlos mit der gewählten Methode segmentiert werden. Auch die Erweiterbarkeit um Kürzel der DEK wurde berücksichtigt. Diese können ebenfalls problemlos mit dem gezeigten Ansatz segmentiert werden. Die Methode ist im Vergleich zu gezeigten Ansätzen anderer Sprachen sehr einfach und funktioniert dennoch sehr gut. Die Umsetzung hingegen bietet noch Potential für Verbesserungen, welche in Kapitel 5.1 ausführlich erläutert wurden. Außerdem wären Tests der Segmentierung anhand einer manuell angelegten Datenbank sinnvoll, um die Leistungsfähigkeit der Methode besser bewerten zu können.

Für die Klassifikation der Konsonantenzeichen wurden Merkmale extrahiert, deren Leistungsfähigkeit anhand von zwei unabhängigen Algorithmen getestet und bestätigt wurde. Die Extraktion erfolgt dabei bereits während der Segmentierung bzw. im Anschluss daran durch das sehr einfache Verfahren des Direction-Chain-Codes, der für das Schriftbild der DEK sehr gut geeignet ist. Eine Erkennungsrate von über 98% ist ein sehr guter Wert für ein Handschriftenerkennungssystem.

Aufgrund des begrenzten zeitlichen Rahmens war es nicht möglich, auch noch die Klassifikation der Vokalverbindungen umzusetzen. Dennoch wurde auch diese Problemstellung analysiert und bereits erste Lösungswege skizziert.

In diesem Zusammenhang empfiehlt es sich, das System so zu erweitern, dass zum Lernen sowohl der Konsonantenzeichen als auch der Vokalverbindungen nicht einzelne Kombinationen gelernt werden, sondern dass ein statistisch sinnvoller Text durch den tatsächlichen Verwender des Systems in der DEK abgeschrieben wird. Hier kann der Vorteil der DEK genutzt werden, umfangreiche Texte in hoher Geschwindigkeit zu übertragen. Bei einer Schreibgeschwindigkeit von 120 Wörtern pro Minute können in nur wenigen Minuten genug Muster zum Trainieren der Classifier produziert werden.

Des Weiteren wäre ein adaptiver Prozess denkbar, bei dem zusätzlich zu den Initial gelernten Mustern während der Verwendung des Systems vom Benutzer korrigierte Muster ebenfalls in das Trainingsdaten Set aufgenommen werden. Automatisch oder durch manuelles Anstoßen könnten diese von Zeit zu Zeit ebenfalls vom Classifier gelernt werden. Das System würde sich somit dem Verwender über die Dauer besser anpassen und auch seltene Zeichenkombinationen würden irgendwann gelernt.

Weiter ist die Integration einer Kontexterkennung aufgrund der Doppeldeutigkeit mancher DEK Zeichen zwingend erforderlich. Dabei sind Systeme auf Satzbasis notwendig. Die in der Arbeit gezeigten Systeme zur Kontexterkennung arbeiten lediglich innerhalb eines Wortes. Dennoch könnte auch der Einsatz der gezeigten Ansätze die Erkennungsrate weiter erhöhen, gerade wenn ein Schreiber ein System verwendet, das er nicht selbst trainiert hat.

# 7. Literaturverzeichnis

1. **Behm, Wolfgang.** Forschungs- und Ausbildungsstätte für Kurzschrift und Textverarbeitung in Bayreuth e.V. [Online] 23. September 2001. [Zitat vom: 15. November 2010.] http://www.forschungsstaette.de/PDF/behm.pdf.
2. *Systemurkunde der Deutschen Einheitskurzschrift, Wiener Urkunde.* **Treschwig, Hans, [Hrsg.].** Wolfenbüttel : Heckners Verlag, 1968.
3. **Drews, Ilse.** *STENO heute.* Troisdorf : Bildungsverlag EINS GmbH, 1999. ISBN 978-3-8242-6100-0.
4. **Microsoft Corporation.** Microsoft.Ink. [Online] [Zitat vom: 16. November 2010.] http://msdn.microsoft.com/en-us/library/ms826516.aspx.
5. **Lu, Yi und Shridhar, M.** Character Segmentation in Handwritten Words - An Overview. *Pattern Recognition.* 1, 1996, 29.
6. **Cho, Wongyu, Lee, Seong-Whan und Kim, Jin H.** Modelling and recognition of cursive words with hidden markov models. *Pattern Recognition.* 1995, Bd. 12, 28.
7. **Duneau, Laurent und Dorizzi, Bernadette.** On-line cursive script recognition: A user-adaptive system for word identification. *Pattern Recognition.* 1996, Bd. 12, 29.
8. **Powalka, R.K., Sherkat, N. und Whitrow, R.J.** Word shape analysis for a hybrid recognition system. *Pattern Recognition.* 1997, Bd. 3, 30.
9. **Dehghan, M., et al.** Handwritten Farsi (Arabic) word recognition: a holistic approach using discrete HMM. *Pattern Recognition.* 2001, 34.
10. **Günter, Simon und Bunke, Horst.** HMM-based handwritten word recognition: on the optimization of the number of states, training iterations and Gaussian components. *Pattern Recognition.* 2004, 37.
11. **Bozinovic, R.M. und Srihari, S.N.** Off-line cursive script word recognition. *IEEE Trans. PAMI.* 1989, 11, S. 68-83.
12. **Chen, M.Y., et al.** Off-line handwritten word recognition using Hidden Markov Model. *U.S. Postal Service 5th Adv. Technol. Conf.* 1992, S. 563-577.
13. **Forney Jr, G.D.** The Viterbi algorithm. *IEEE Proc.* 1973, 61, S. 268-278.
14. **Balestri, M. und Masera, L.** A system for isolating characters in cursive script. [Buchverf.] Lacoume et al. *Signal Processing IV: Theories and Applications.* North-Holland : Elsevier Science Publishers, 1988, S. 845-846.
15. **Bhowmik, T.K., Roy, A. und Roy, U.** Character segmentation for handwritten Bangla words using artificial neural network. *Neural Networks and Learning in Document Analysis and Recognition.* [Online] 29. August 2005. [Zitat vom: 21. November 2010.] http://www.dsi.unifi.it/NNLDAR/Papers/06-NNLDAR-roy.pdf.
16. **Ma, Yang und Leedham, Graham.** On-line recognition of handwritten Renqun shorthand for fast mobile Chinese text entry. *Pattern Recognition Letters.* 2007, 28.
17. **Teh, C.H. und Chin, R.T.** On the detection of dominant points on digital curves. *IEE Trans. Pattern Anal. Machine Intell.* 1989, Bd. 8, 11, S. 859-872.
18. **Ma, Yang, et al.** Segmentation and recognition of phonetic features in handwritten Pitman shorthand. *Pattern Recognition.* 2008, 41.
19. **Shaikh, Noor Ahmed, Mallah, Ghulam Ali und Shaikh, Zubair A.** Character segmentation of Sindhi, ana Arabic Style Scripting Language, using Height Profile Vector. *Australian Journal of Basic and Applied Sciences.* 2009, Bd. 4, 3, S. 4160-4169.
20. **Razak, Zaidi, et al.** Off-line handwritten Jawi character segmentation using histogram normalization and sliding window approach for hardware implementation. *Malaysian Journal of Computer Science.* 2009, Bd. 1, 22.
21. **Rehman, Amjad, Mohamed, Dzulkifli und Sulong, Ghazli.** Implicit vs explicit based script segmentation and recognition: A performance comparison on Benachmark Database. *Int. J. Open Problems Compt. Math.* 2009, Bd. 3, 2.

22. **Hanson, A.R., Riseman, E.M. und Fisher, E.** Context in word recognition. *Pattern Recognition.* 1976, 8, S. 35-45.
23. **Goshtasby, Ardeshir und Ehrich, Roger W.** Contextual word recognition using probabilistic relaxation labeling. *Pattern Recognition.* 1988, Bd. 5, 21.
24. **Rosenfeld, A., Hummel, R.A. und Zucker, S.W.** Scene labeling by relaxation operations. *IEEE Trans. Systems Man Cyber. SMC.* 1976, 6, S. 420-433.
25. **Wells, C.J., et al.** Fast dictionary look-up for contextual word recognition. *Pattern Recognition.* 1990, Bd. 5, 23, S. 501-508.
26. **Iwayama, Naomi und Ishigaki, Kazushi.** Adaptive context processing in on-line handwritten character recognition. [Buchverf.] L.R.B. Schomaker und L.G. Vuurpijl. *Proceedings of the Seventh International Workshop on Frontiers in Hadwriting Recognition.* Amsterdam : Nijmegen: International Unipen Foundation, 2000, S. 469-474.
27. **Myo Htwe, Swe, et al.** Knowledge based transcription of handwritten pitman's shorthand using word frequendy and context. *7th International Conference on Development and Application Systems.* 2004, S. 508-512.
28. **Sarman, Stanislav Jan.** DEK-Verkehrsschrift mit METAFONT und LATEX. *Die TEXnische Komödie.* 2009, 1, S. 8-20.
29. **Slavik, Petr und Govindaraju, Venu.** Equivalence of Different Methods for Sland and Skew Corrections in Word Recognition Applications. *IEEE Transactions On Pattern Analysis and Machine Intelligence.* 2001, Bd. 3, 23.
30. **Zobrak, M.J. und Sze, T.W.** A method of recognition of hand drawn line patterns. *Space Research Coordination Center Report.* 1967, 59.
31. **Duda, Richard O., Hart, Peter E. und Stork, David G.** *Pattern Classification.* New York : John Wiley & Sons, Inc., 2001. 978-0-471-05669-0.
32. **Ertel, Wolfgang.** *Grundkurs Künstliche Intelligenz.* Wiesbaden : Friedr. Vieweg & Sohn Verlag | GWV Fachverlage GmbH, 2008. 978-3-528-05924-8.
33. **Hung, M.S., et al.** Estimating Posterior Probabilities In Classification Problems With Neural Networks. *International Journal of Computational Intelligence and Organisations.* 1996, 1, S. 49-60.
34. **Leedham, C.G. und Downton, A.C.** Automatic recognition and transcription of Pitman's handwritten shorthand - an approach to shortforms. *Pattern Recognition.* 1987, Bd. 3, 20.

# Anhang A

Hier soll kurz die in der Arbeit entstandene Software „StenoPad“ vorgestellt werden. Mithilfe dieses Prototypen ist es möglich, neue Muster der Datenbank hinzuzufügen und den Classifier anhand der Datenbank zu trainieren. Außerdem kann der trainierte Classifier anschließend für die Erkennung einer Eingabe verwendet werden. Welcher Classifier verwendet werden soll (Neuronales Netz oder Entscheidungsbaum), kann in den Einstellungen konfiguriert werden.

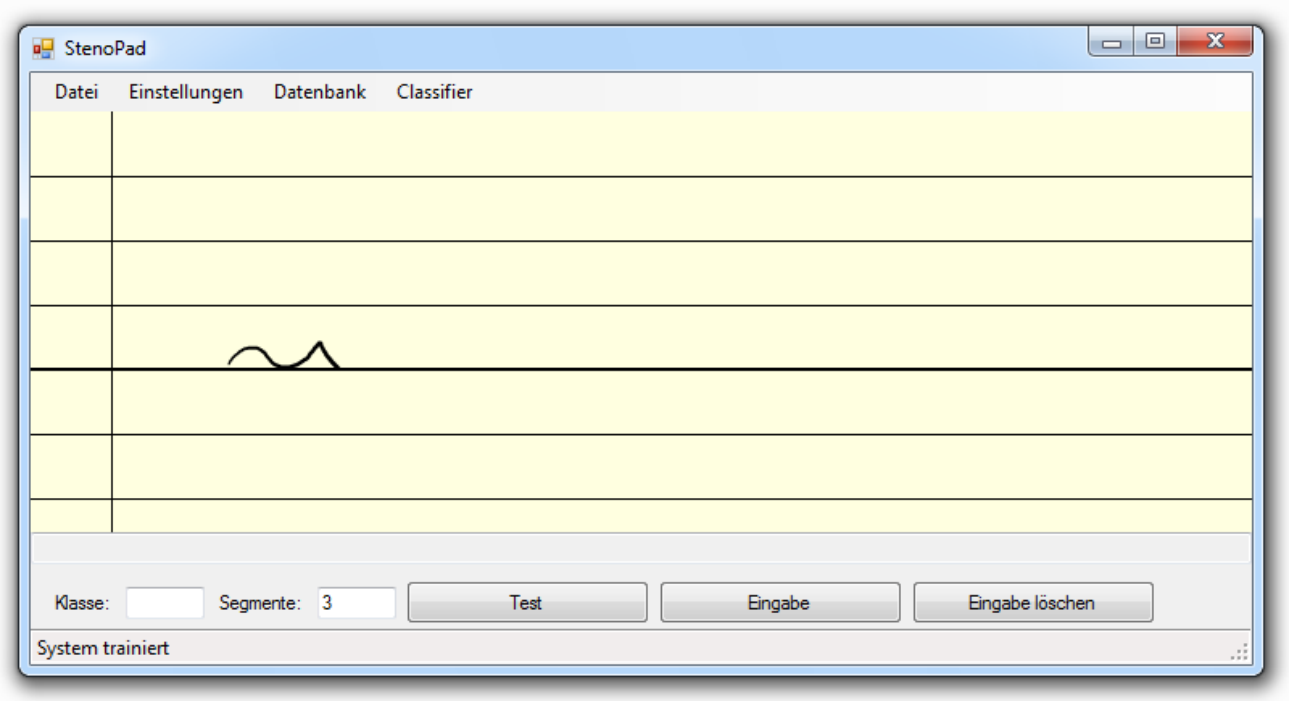



**Abbildung 54: Screenshot der Anwendung "StenoPad".**

## Training

Beim Training können neue Zeichen eingegeben werden. Hierzu muss zuerst der Datenbankpfad unter „Datenbank“ → „Pfad setzen…“ festgelegt werden. Anschließend wird das Zeichen mit hinführender und verlassender Vokalverbindung geschrieben. Unter „Klasse“ muss die Klasse des Zeichens hinterlegt werden und „Segmente“ gibt an, wie viele Segmente die Eingabe aufweisen wird. Anhand dieses Wertes kann vor der Speicherung in der Datenbank überprüft werden, ob die Eingabe evtl. Fehler enthält. Mittels „Test“ wird die Eingabe segmentiert und die Parameter der Merkmalsextraktion in der Statusleiste angezeigt. Mit „Eingabe“ wird ebenfalls segmentiert, anstatt die Parameter anzuzeigen werden diese in die Datenbank übernommen. „Eingabe löschen“ leert die Eingabe. Dies geschieht bei der Eingabe eines Zeichens automatisch, um schneller weitere Zeichen erfassen zu können.

Wurden alle Einträge in der Datenbank erfasst, kann der Classifier trainiert werden. Unter „Einstellungen“ kann der zu trainierende Algorithmus bestimmt werden. J48 entspricht hierbei dem Entscheidungsbaum und RPROP dem neuronalen Netz. Mittels „Classifier“ → „trainieren“ wird der ausgewählte Classifier mit der festgelegten Datenbank trainiert. Hierzu werden zunächst alle Einträge der Datenbank geladen, segmentiert und die Merkmale extrahiert. Diese Vorgehensweise bietet den Vorteil, dass bei Änderungen am Segmentierungsalgorithmus oder an der Merkmalsextraktion sich diese auch auf die bereits bestehende Datenbankeinträge auswirken. Das Training kann eine Weile in Anspruch nehmen und ist erst fertig, wenn in der Statuszeile „System trainiert“ angezeigt wird.

**Verwenden**

Wurde der Classifier trainiert, kann dieser anschließend verwendet werden. Hierzu muss unter „Einstellungen“ der „Trainingsmodus“ verlassen werden. Die Eingabefelder „Klasse“ und „Segmente“ sowie die Schaltfläche „Test“ werden daraufhin deaktiviert. Mittels „Eingabe“ wird nun jedes eingegebene Segment vom Classifier getestet und unterhalb der Eingabefläche die wahrscheinlichste Ausgabe für diese angezeigt.

**Systemvoraussetzungen**

Zum einfachen Ausführen der Anwendung sind aufgrund der Abhängigkeiten zu der WEKA und Encog Bibliothek folgende Versionen des .NET Frameworks notwendig:

- Microsoft .NET Framework Version 1.1 Redistributable Package[16]
- Microsoft .NET Framework 2.0 Redistributable (x86)[17]
- Microsoft .NET Framework 3.0 Redistributable Package[18]
- Microsoft .NET Framework 3.5[19]
- Microsoft .NET Framework 3.5 Service Pack 1[20]
- Microsoft .NET Framework 4.0[21]

Zusätzlich benötigt die WEKA Bibliothek:

- J# redistributable package[22]
- Microsoft Supplemental UI Library for Visual J# .NET v1.1[23]

Außerdem funktioniert die Anwendung nur auf PCs mit Tablet Betriebssystem. Dies kann entweder Windows XP Tablet PC oder Windows Vista oder später sein.

Die Entwicklung erfolgte mit Visual Studio 2010 Ultimate. Um das Projekt laden zu können, sind alle Voraussetzungen zu erfüllen, die auch für das Ausführen benötigt werden. Zusätzlich ist noch das Windows Tablet PC Plattform SDK zu installieren.

[16] http://www.microsoft.com/downloads/details.aspx?FamilyID=262d25e3-f589-4842-8157-034d1e7cf3a3&displayLang=de
[17] http://www.microsoft.com/downloads/details.aspx?familyid=0856EACB-4362-4B0D-8EDD-AAB15C5E04F5&displaylang=de
[18] http://www.microsoft.com/downloads/details.aspx?familyid=10CC340B-F857-4A14-83F5-25634C3BF043&displaylang=de
[19] http://www.microsoft.com/downloads/details.aspx?familyid=333325FD-AE52-4E35-B531-508D977D32A6&displaylang=de
[20] http://www.microsoft.com/downloads/details.aspx?familyid=AB99342F-5D1A-413D-8319-81DA479AB0D7&displaylang=de
[21] http://www.microsoft.com/downloads/details.aspx?familyid=0A391ABD-25C1-4FC0-919F-B21F31AB88B7&displaylang=de
[22] http://msdn2.microsoft.com/en-us/vjsharp/
[23] http://msdn2.microsoft.com/en-us/vjsharp/bb188695.aspx

# Anhang B

### Zeichen mit einer Höhe von einer halben Stufe

```
Correctly Classified Instances         788               98.995  %
Incorrectly Classified Instances         8                1.005  %
Kappa statistic                          0.9885
Mean absolute error                      0.0034
Root mean squared error                  0.0512
Relative absolute error                  1.5576 %
Root relative squared error             15.4782 %
Total Number of Instances              796

J48 pruned tree
------------------

a14 <= 0.3
|   a9 <= 0.1
|   |   a2 <= 0.333333: 11 (100.0/1.0)
|   |   a2 > 0.333333
|   |   |   a7 <= 0
|   |   |   |   a2 <= 0.666667: 11 (4.0/1.0)
|   |   |   |   a2 > 0.666667: 8 (100.0)
|   |   |   a7 > 0: 30 (98.0)
|   a9 > 0.1
|   |   a1 <= 0.333333: 9 (99.0)
|   |   a1 > 0.333333
|   |   |   a2 <= 0.333333: 33 (99.0)
|   |   |   a2 > 0.333333: 31 (101.0/1.0)
a14 > 0.3
|   a4 <= 0: 32 (95.0)
|   a4 > 0
|   |   a0 <= 0: 32 (5.0)
|   |   a0 > 0: 13 (95.0)

Number of Leaves  :     10
Size of the tree :      19
```

### Zeichen mit einer Höhe von einer Stufe

```
Correctly Classified Instances        2235               98.6755 %
Incorrectly Classified Instances        30                1.3245 %
Kappa statistic                          0.9862
Mean absolute error                      0.0014
Root mean squared error                  0.0326
Relative absolute error                  1.6369 %
Root relative squared error             15.9867 %
Total Number of Instances             2265

J48 pruned tree
------------------

a8 <= 0.1
|   a4 <= 0
|   |   a19 <= 0.1
|   |   |   a24 <= 0
|   |   |   |   a13 <= 0.1
|   |   |   |   |   a2 <= 0.333333: 25 (97.0)
|   |   |   |   |   a2 > 0.333333
|   |   |   |   |   |   a8 <= 0: 25 (2.0)
|   |   |   |   |   |   a8 > 0: 24 (13.0)
|   |   |   |   a13 > 0.1: 10 (4.0)
```

```
|   |   |   a24 > 0
|   |   |   |   a8 <= 0
|   |   |   |   |   a7 <= 0: 26 (101.0/2.0)
|   |   |   |   |   a7 > 0: 24 (3.0)
|   |   |   |   a8 > 0: 24 (76.0)
|   |   a19 > 0.1
|   |   |   a5 <= 0
|   |   |   |   a23 <= 0.2
|   |   |   |   |   a23 <= 0.1: 10 (85.0/1.0)
|   |   |   |   |   a23 > 0.1
|   |   |   |   |   |   a12 <= 0.2
|   |   |   |   |   |   |   a13 <= 0.2: 5 (99.0)
|   |   |   |   |   |   |   a13 > 0.2: 2 (5.0)
|   |   |   |   |   |   a12 > 0.2: 2 (83.0)
|   |   |   |   a23 > 0.2: 6 (100.0)
|   |   |   a5 > 0: 57 (99.0)
|   a4 > 0
|   |   a19 <= 0.3
|   |   |   a24 <= 0.1
|   |   |   |   a6 <= 0
|   |   |   |   |   a1 <= 0.333333: 22 (99.0)
|   |   |   |   |   a1 > 0.333333: 21 (3.0)
|   |   |   |   a6 > 0: 21 (95.0)
|   |   |   a24 > 0.1: 23 (101.0/1.0)
|   |   a19 > 0.3
|   |   |   a3 <= 0: 28 (95.0)
|   |   |   a3 > 0: 29 (15.0)
a8 > 0.1
|   a9 <= 0.3
|   |   a5 <= 0
|   |   |   a7 <= 0.2
|   |   |   |   a22 <= 0.1
|   |   |   |   |   a6 <= 0.2: 1 (97.0)
|   |   |   |   |   a6 > 0.2: 16 (2.0/1.0)
|   |   |   |   a22 > 0.1
|   |   |   |   |   a7 <= 0.1: 2 (9.0)
|   |   |   |   |   a7 > 0.1
|   |   |   |   |   |   a4 <= 0
|   |   |   |   |   |   |   a8 <= 0.2: 14 (98.0)
|   |   |   |   |   |   |   a8 > 0.2: 58 (3.0/1.0)
|   |   |   |   |   |   a4 > 0: 21 (2.0)
|   |   |   a7 > 0.2
|   |   |   |   a24 <= 0.2
|   |   |   |   |   a2 <= 0.666667: 16 (99.0)
|   |   |   |   |   a2 > 0.666667: 27 (99.0/1.0)
|   |   |   |   a24 > 0.2
|   |   |   |   |   a4 <= 0: 58 (94.0)
|   |   |   |   |   a4 > 0: 28 (5.0)
|   |   a5 > 0
|   |   |   a3 <= 0: 59 (101.0/1.0)
|   |   |   a3 > 0: 29 (84.0)
|   a9 > 0.3
|   |   a5 <= 0
|   |   |   a17 <= 0.1: 18 (99.0)
|   |   |   a17 > 0.1
|   |   |   |   a24 <= 0.1: 3 (100.0)
|   |   |   |   a24 > 0.1: 20 (100.0)
|   |   a5 > 0: 56 (98.0)

Number of Leaves  :     35
Size of the tree :      69
```

**Zeichen mit einer Höhe von zwei Stufen**

```
Correctly Classified Instances        2805               98.1456 %
Incorrectly Classified Instances        53                1.8544 %
Kappa statistic                          0.9808
Mean absolute error                      0.0015
Root mean squared error                  0.0347
Relative absolute error                  2.2652 %
Root relative squared error             19.022  %
Total Number of Instances             2858

J48 pruned tree
------------------

a5 <= 0
|   a16 <= 0.5
|   |   a8 <= 0.1
|   |   |   a4 <= 0
|   |   |   |   a2 <= 0.333333: 35 (101.0)
|   |   |   |   a2 > 0.333333
|   |   |   |   |   a30 <= 0.1: 17 (95.0)
|   |   |   |   |   a30 > 0.1
|   |   |   |   |   |   a41 <= 0.2
|   |   |   |   |   |   |   a40 <= 0.2
|   |   |   |   |   |   |   |   a12 <= 0.1
|   |   |   |   |   |   |   |   |   a20 <= 0.2
|   |   |   |   |   |   |   |   |   |   a22 <= 0.4: 39 (99.0)
|   |   |   |   |   |   |   |   |   |   a22 > 0.4: 51 (2.0)
|   |   |   |   |   |   |   |   |   a20 > 0.2: 51 (16.0)
|   |   |   |   |   |   |   |   a12 > 0.1: 51 (82.0)
|   |   |   |   |   |   |   a40 > 0.2: 17 (6.0/3.0)
|   |   |   |   |   |   a41 > 0.2
|   |   |   |   |   |   |   a11 <= 0.1: 36 (97.0/1.0)
|   |   |   |   |   |   |   a11 > 0.1: 17 (2.0)
|   |   |   a4 > 0: 49 (99.0)
|   |   a8 > 0.1
|   |   |   a1 <= 0.333333
|   |   |   |   a0 <= 0.266667: 52 (100.0)
|   |   |   |   a0 > 0.266667: 41 (2.0/1.0)
|   |   |   a1 > 0.333333
|   |   |   |   a2 <= 0.333333
|   |   |   |   |   a9 <= 0.2
|   |   |   |   |   |   a4 <= 0: 38 (100.0)
|   |   |   |   |   |   a4 > 0: 19 (2.0/1.0)
|   |   |   |   |   a9 > 0.2: 12 (100.0)
|   |   |   |   a2 > 0.333333
|   |   |   |   |   a7 <= 0.2
|   |   |   |   |   |   a41 <= 0.2
|   |   |   |   |   |   |   a8 <= 0.2: 37 (100.0)
|   |   |   |   |   |   |   a8 > 0.2: 41 (2.0)
|   |   |   |   |   |   a41 > 0.2: 7 (99.0)
|   |   |   |   |   a7 > 0.2
|   |   |   |   |   |   a42 <= 0.2
|   |   |   |   |   |   |   a43 <= 0.2: 41 (97.0/1.0)
|   |   |   |   |   |   |   a43 > 0.2: 15 (2.0)
|   |   |   |   |   |   a42 > 0.2: 15 (96.0)
|   a16 > 0.5
|   |   a2 <= 0.666667
|   |   |   a23 <= 0.2: 4 (101.0/1.0)
|   |   |   a23 > 0.2: 34 (100.0)
|   |   a2 > 0.666667
|   |   |   a25 <= 0.2: 40 (98.0)
```

```
|   |   |   a25 > 0.2: 42 (99.0)
a5 > 0
|   a3 <= 0
|   |   a8 <= 0.1
|   |   |   a28 <= 0.2
|   |   |   |   a36 <= 0.3
|   |   |   |   |   a27 <= 0.1
|   |   |   |   |   |   a37 <= 0.3: 62 (99.0)
|   |   |   |   |   |   a37 > 0.3: 50 (8.0)
|   |   |   |   |   a27 > 0.1: 63 (90.0/1.0)
|   |   |   |   a36 > 0.3
|   |   |   |   |   a24 <= 0.1: 50 (85.0)
|   |   |   |   |   a24 > 0.1
|   |   |   |   |   |   a13 <= 0.2
|   |   |   |   |   |   |   a19 <= 0.2
|   |   |   |   |   |   |   |   a36 <= 0.4: 63 (7.0)
|   |   |   |   |   |   |   |   a36 > 0.4: 43 (95.0/1.0)
|   |   |   |   |   |   |   a19 > 0.2: 50 (2.0)
|   |   |   |   |   |   a13 > 0.2: 50 (5.0)
|   |   |   a28 > 0.2
|   |   |   |   a33 <= 0.3: 63 (3.0)
|   |   |   |   a33 > 0.3: 48 (101.0/1.0)
|   |   a8 > 0.1
|   |   |   a7 <= 0.2
|   |   |   |   a28 <= 0.2
|   |   |   |   |   a36 <= 0.3: 60 (97.0/1.0)
|   |   |   |   |   a36 > 0.3
|   |   |   |   |   |   a40 <= 0.6: 19 (92.0/1.0)
|   |   |   |   |   |   a40 > 0.6
|   |   |   |   |   |   |   a34 <= 0.3: 60 (4.0)
|   |   |   |   |   |   |   a34 > 0.3: 19 (6.0)
|   |   |   |   a28 > 0.2
|   |   |   |   |   a27 <= 0.2
|   |   |   |   |   |   a0 <= 0.4: 19 (2.0)
|   |   |   |   |   |   a0 > 0.4: 47 (3.0)
|   |   |   |   a27 > 0.2: 47 (68.0)
|   |   |   a7 > 0.2
|   |   |   |   a36 <= 0.3: 61 (96.0)
|   |   |   |   a36 > 0.3
|   |   |   |   |   a0 <= 0.666667: 44 (98.0)
|   |   |   |   |   a0 > 0.666667: 61 (2.0)
|   a3 > 0
|   |   a4 <= 0: 46 (99.0/1.0)
|   |   a4 > 0: 45 (99.0)

Number of Leaves  :     47
Size of the tree :      93
```

**Zeichen mit einer Höhe von drei Stufen**

```
Correctly Classified Instances         295               98.6622 %
Incorrectly Classified Instances         4                1.3378 %
Kappa statistic                          0.9799
Mean absolute error                      0.009
Root mean squared error                  0.0867
Relative absolute error                  2.0195 %
Root relative squared error             18.3994 %
Total Number of Instances              299

J48 pruned tree
------------------

a14 <= 0.3
```

```
|   a7 <= 0.1: 55 (91.0)
|   a7 > 0.1
|   |   a64 <= 0
|   |   |   a8 <= 0.2: 55 (6.0)
|   |   |   a8 > 0.2: 54 (3.0)
|   |   a64 > 0
|   |   |   a7 <= 0.2
|   |   |   |   a0 <= 0.533333: 55 (2.0)
|   |   |   |   a0 > 0.533333: 54 (11.0)
|   |   |   a7 > 0.2: 54 (86.0)
a14 > 0.3: 53 (100.0)

Number of Leaves  :     7
Size of the tree :      13
```