Implementation und Test eines Sprechererkennungsverfahrens für die biometrische Authentikation

Projektbericht im Rahmen des Projektes „Biometrik“
am Arbeitsbereich AGN des
FB Informatik der Universität Hamburg

Verfasser:
Karin Colsman
Matr.-Nr. 478 14 53
Sperlingsweg 59
22453 Hamburg

Betreuer:
Dipl.-Inform. B.Sc. Arslan Brömme
FB Informatik, Universität Hamburg
Arbeitsbereich AGN

Prof. Dr.-Ing. Wolfgang Menzel
FB Informatik, Universität Hamburg
Arbeitsbereich NATS

26. August 2001

Inhaltsverzeichnis

1 Einleitung

In diesem Projektbericht soll die Implementation und der Test eines Sprechererkennungsverfahrens für die biometrische Authentikation beschrieben und unter den Gesichtspunkten des Datenschutzes und des Nutzens von Realtests betrachtet werden.

Dazu sollen zuerst einige Grundlagen vermittelt werden: In Abschnitt 2.1 zum Thema Sicherheitsmodelle und Zugriffskontrolle, in Abschnitt 2.2 zum Thema biometrische Authentikation, und in Abschnitt 2.3 zum Thema Sprechererkennung. In einem nächsten Schritt wird in Abschnitt 3 die Implementation eines Dynamic Time Warping-Verfahrens beschrieben. Die Testbedingungen und -ergebnisse werden in Abschnitt 4.2 erläutert, um zum Abschluß in Abschnitt 5 den Nutzen von Realtests und die Auswirkung von biometrischen Authentikationsverfahren auf den Datenschutz zu diskutieren.

2 IT-Sicherheit und Biometrik

2.1 Sicherheitsmodelle und Zugriffskontrolle

Im Zusammenhang mit der Sicherheit eines IT-Systems sind verschiedene Aspekte zu beachten: Zum einen gilt es, das System selber vor unautorisiertem Zugriff zu schützen, so daß die Verfügbarkeit der Daten (im Sinne von Zugreifbarkeit) und ihre Integrität gewährleistet sind. Zum anderen muß darauf geachtet werden, daß vertrauliche Daten über Personen und Organisationen nicht mißbraucht oder manipuliert werden können.[1]

Um ein IT-System vor unautorisiertem Zugriff zu schützen müssen mehrere Ebenen betrachtet werden (vgl. [Fischer-Hübner 2001]): Zuerst muß eine *Sicherheitspolitik* beschrieben werden, die u.a. genau definiert, wer Zugriff auf welche Prozesse und Objekte haben bzw. nicht haben soll. Solch eine Sicherkeitspolitik läßt sich durch ein *Sicherheitsmodell* formalisieren, um dann durch die passenden *Sicherheitsfunktionen* und *-mechanismen* implementiert zu werden. Nicht technische Aspekte einer Sicherheitspolitik wie z.B. gewünschtes Benutzerverhalten können jedoch nicht implementiert werden.

2.1.1 Role-Based Access Control

Die flexibelsten Sicherheitsmodelle, die bis heute entwickelt wurden sind die Rollenbasierten Zugriffsmodelle (RBAC), durch die sich die technischen

[1] StGB § 202a (Ausspähen von Daten), §263a (Computerbetrug), §303a (Datenveränderung), §303b (Computersabotage)

Aspekte verschiedenster Sicherheitspolitiken formalisieren lassen. Dabei definiert ein Sicherheitsadministrator die Rollen durch ihre Zugriffsrechte auf die Systemobjekte. Einem Benutzer des Systems kann dann durch den Administrator Zugang zu einer oder mehreren Rollen gewährt werden.

Die Elemente eines RBAC-Modells sind folgendermaßen definiert: Ein *User* ist eine Person, ein *Subjekt* ein aktiver User-Prozess, eine *Rolle* eine Sammlung von Privilegien, und eine *Operation* der Zugriffsmodus auf ein geschütztes *Objekt*. Diese Elemente werden durch folgende *Funktionen* untereinander in Beziehung gesetzt:

- *subject-user*(s:Subjekt): Der dem Subjekt s zugeordnete User.
- *authorized-roles*(s:Subjekt): Die Menge der Rollen, für die das Subjekt s autorisiert ist.
- *role-members*(r:Rolle): Die Menge der User, die für die Rolle r autorisiert sind. Dabei muß die Konsistenz mit den beiden vorangehenden Funktionen gewahrt werden.
- *user-authorized-role*(u:User): Die Menge der Rollen, für die der User u autorisiert ist.
- *role-operations*(r:Rolle): Die Menge der Operationen, auf die die Rolle r Zugriff hat.
- *operation-objects*(op:Operation): Die Menge der autorisierten Objekte, auf die die Operation *op* angewendet werden darf.

Für ein RBAC-Modell können folgende *Sicherheitseigenschaften* definiert werden:

1. *Rollenhierarchie*: Eine Rolle kann in einer anderen Rolle enthalten sein. Wenn ein Subjekt für die Oberrolle autorisiert ist, hat es auch Zugriff auf die Unterrolle.
2. *Statische Pflichtentrennung*: Wenn sich zwei Rollen r_1, r_2 gegenseitig ausschließen, kann ein User nur Mitglied der Rolle r_1 werden, wenn er nicht bereits Mitglied der Rolle r_2 ist.
3. *Rollenkapazität*: Die maximale Anzahl der Mitglieder wird für bestimmte Rollen festgelegt und darf dort nicht überschritten werden.
4. *Rollenautorisierung*: Die Menge der Rollen, die ein Subjekt gerade benutzt, wird als *aktive Rollen* bezeichnet und muß zu jeder Zeit eine Teilmenge der für das Subjekt autorisierten Rollen sein.

5. *Rollenausführung*: Ein Subjekt kann eine Operation nur innerhalb einer aktiven Rolle ausführen.

6. *Dynamische Pflichtentrennung*: Wenn sich zwei Rollen r_1, r_2 gegenseitig ausschließen kann ein Subjekt die Rolle r_1 nur für sich aktivieren, wenn er nicht bereits die Rolle r_2 für sich aktiviert hat.

7. *Autorisierung der Operation*: Ein Subjekt kann eine Operation nur ausführen, wenn diese Operation für eine seiner aktiven Rollen zulässig ist.

8. *Operationale Pflichtentrennung*: Die für einen bestimmten Geschäftsvorgang benötigten Operationen dürfen nicht exklusiv von den für einen einzelnen User autorisierten Rollen ausgeführt werden.

9. *Autorisierung des Objektzugriffs*: Ein Subjekt darf nur auf ein Objekt zugreifen, wenn eine seiner aktiven Rollen eine für diese Rolle zulässige Operation ausführen darf und diese Operation wiederum für den Zugriff auf dieses Objekt autorisiert ist.

Die technischen Aspekte der verschiedenen Sicherheitspolitiken lassen sich nun durch die Spezifikation der RBAC-Funktionen und die Wahl und Spezifikation der Sicherheitseigenschaften formalisieren.

2.1.2 Mandatory Access Control

RBAC kann durch Mandatory Access Control (MAC) implementiert werden. Dabei werden die Sicherheitsattribute der Subjekte und Objekte entweder durch einen Administrator oder automatisch durch das System festgelegt. Ein Subjekt kann nur auf ein Objekt zugreifen, wenn es über die für dieses Objekt notwendigen Sicherheitsattribute verfügt.

2.1.3 Discretionary Access Control

Discretionary Access Control (DAC) ist das Komplement zu MAC. Wenn ein Subjekt ein Zugriffsrecht auf ein Objekt besitzt, kann es dieses Recht an andere Subjekte weitergeben. Der User, der ein Objekt erzeugt hat (*Owner*) entscheidet darüber, wer auf diesem Objekt welche Operationen ausführen darf. Man kann DAC in zwei Unterformen einteilen: Listen von Privilegien (*Capability Lists*) und Zugriffskontrollisten (*Access Control Lists*). Im ersten Fall wird jedem Subjekt s eine Liste von Privilegien zugewiesen, die für jedes Objekt die für s autorisierten Zugriffsmodi festlegt, und das Subjekt s kann seine Privilegien in Bezug auf ein Objekt an andere Subjekte weitergeben

oder ihnen diese entziehen. Im zweiten Fall verfügt jedes Objekt o über eine Liste der Benutzer und/oder Benutzergruppen, die auf o zugreifen dürfen. Wenn sowohl MAC als auch DAC auf ein Objekt angewendet werden, muß MAC immer den Vorrang erhalten (vgl. [Pfleeger 1997]).

2.2 Biometrische Authentikation

Damit ein Benutzer Zugang zu einem geschützten System erhält, muß er sich gegenüber dem System identifizieren, und das System muß ihn authentifizieren. Für die Authentikation kann folgendes durch das System überprüft werden (vgl. [Fischer-Hübner 2001], [Brömme 2001]):

- *Wissen*, z.B. ein Passwort,
- *Besitz*, z.B. eine Chipkarte,
- *Aufenthaltsort*, z.B. ein bestimmtes Terminal oder die geographischen Koordinaten des Benutzers,
- *Physische Eigenschaften*
  - *statisch*, z.B. ein Fingerabdruck,
  - *dynamisch*, z.B. die Pupillenzusammenziehung und -erweiterung bei Änderung der Lichtverhältnisse,
- *Verhalten*, z.B. der Vorgang des Unterschreibens.

Die letzten beiden Punkte sind Teil der *biometrischen Authentikation*. Dabei wird eine biologische Eigenschaft des Benutzers durch ein Sensorsystem erfaßt und aus den so entstandenen biometrischen Rohdaten eine *biometrische Signatur* berechnet. Bei der Registrierung des Benutzers wird dieser Vorgang mitunter mehrfach wiederholt und dann entweder die Menge der Signaturen oder ein aus ihnen berechnetes Modell zusammen mit einer Nutzerkennung in einer Signaturdatenbank abgespeichert.

Für die Authentisierung eines Benutzers wird die Signatur des eingehenden Signals mit der zugehörigen Signatur in der Datenbank verglichen (vgl. Abbildung 1). Wenn die Distanz zwischen der eingehenden Signatur und der Signatur in der Datenbank unterhalb eines vorher definierten Schwellwertes (*Threshold*) liegt, akzeptiert das System den Benutzer.

Da die Signatur eines Benutzers nicht eindeutig ist, kann es bei biometrischen Verfahren zu Fehlentscheidungen kommen. Diese werden durch die

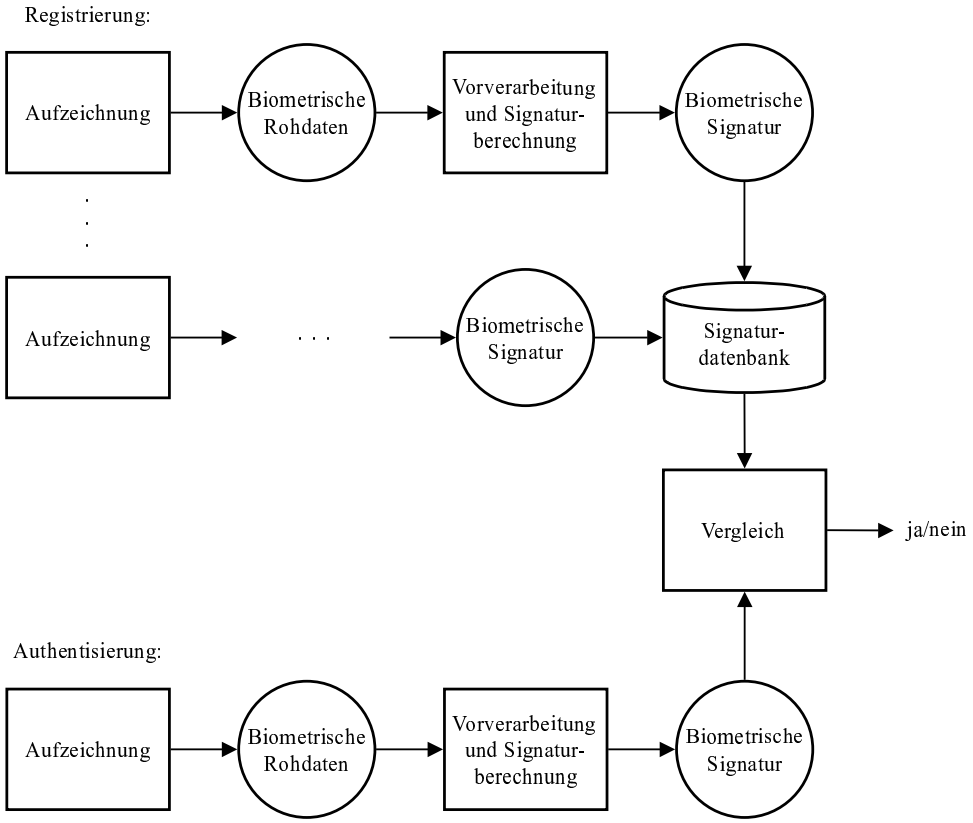

Abbildung 1: Biometrische Authentikation

False Acceptance Rate (FAR), wobei ein nicht autorisierter Benutzer (Impostor) Zugang zum System erhält und die *False Rejection Rate (FRR)*, wobei ein autorisierter Benutzer nicht ins System gelangt, erfaßt:

$$FRR = \frac{NFR}{NAA} * 100\% \tag{1}$$

$$FAR = \frac{NFA}{NIA} * 100\% \tag{2}$$

Dabei bezeichnet NFR die Anzahl der False Rejections, NAA die Anzahl der authorisierten Identifikationsversuche (Number of Authorized identification/verification Attempts), NFA die Anzahl der False Acceptances und NIA die Anzahl der nicht authorisierten Identifikationsversuche (Number of Impostor identification Attempts) (vgl. [Zhang 2000], S. 11). Sind die FAR und die FRR identisch, spricht man von der *Equal Error Rate (ERR)*.

Ein Ziel bei der Entwicklung von biometrischen Authentisierungsverfahren ist, die Anzahl der Fehlentscheidungen so weit wie möglich zu senken.

Dabei werden die FAR, die FRR und die ERR als Performancegrößen benutzt. Welcher Anteil der Fehlentscheidungen jeweils auf die FAR und die FRR fällt, läßt sich je nach Anforderungen an die Sicherheit des Systems durch die Wahl des Thresholds einstellen: Wird der Threshold gesenkt, sinkt auch die FAR; dafür steigt jedoch die FRR. Dieser Zusammenhang läßt sich durch eine sogenannte *Receiver Operating Curve (ROC)* veranschaulichen (vgl. Abbildung 2).

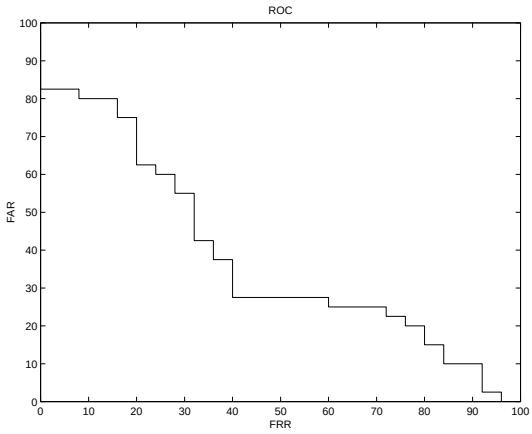

Abbildung 2: Receiver Operating Curve (ROC)

2.3 Sprechererkennung

Die für die Sprechererkennung genutzten Verfahren gleichen den Verfahren der Spracherkennung. Sie sind jedoch komplementär in der Zielsetzung: Während die Spracherkennungsverfahren zum Ziel haben, die sprecherspezifischen Charakteristika weitestmöglich auszublenden sind es genau diese Charakteristika, die die verschiedenen Sprechererkennungsverfahren aus dem Sprachsignal extrahieren müssen. In Abschnitt 2.3.1 werden die generellen Charakteristika von Sprachsignalen, und in Abschnitt 2.3.2 Grundlagen der Sprechererkennung erläutert.

2.3.1 Phonetik

Sprechermodell. In Abbildung 3 wird die Entstehung eines Sprachsignals modelliert: Der durch die Lungen erzeugte Luftstrom wird durch die *aktiven Artikulatoren* (Stimmbänder/Glottis, Gaumensegel/Uvula, Zunge und Lippen) verändert. Dabei kann von den Stimmbändern ein quasiperiodisches Signal (*Anregung*) erzeugt werden, wobei die anderen Artikulatoren

den Frequenzgang dieses Signals beeinflussen, indem die Form der Mundhöhle verändert und der Zugang zum Nasalraum geöffnet oder verschlossen wird. Die aktiven Artikulatoren können außerdem selbst als Signalquelle dienen, indem sie aperiodische Signale (Geräusche) erzeugen. Weiterhin kann das durch die Stimmbänder erzeugte Signal seine Grundfrequenz variieren (Intonation). Durch diese Veränderungen im Sprachsignal wird die Sprachinformation übertragen.

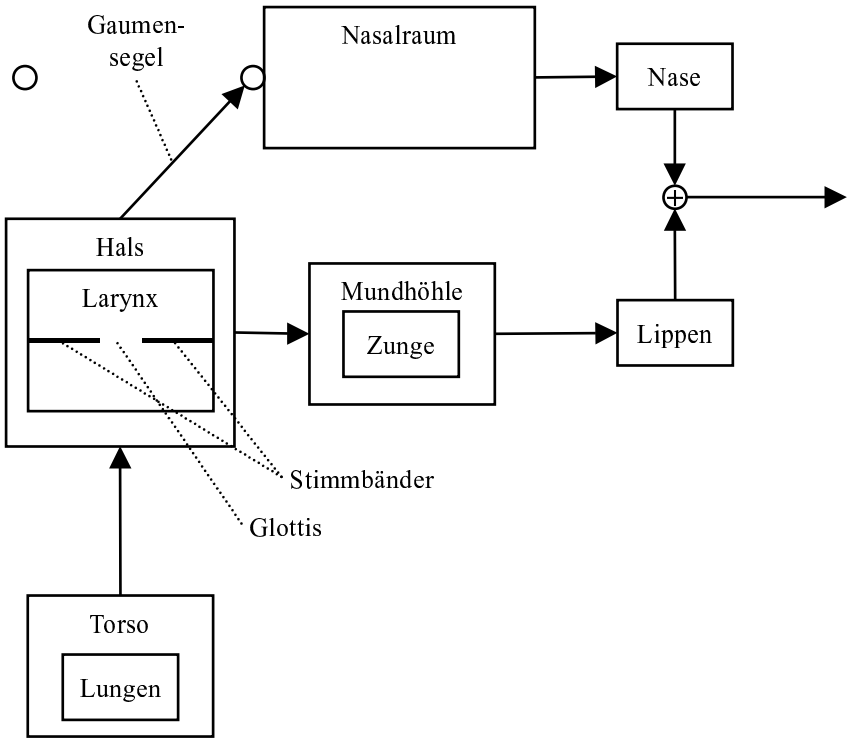

Abbildung 3: Sprechermodell

Es sind jedoch noch zusätzliche Informationen im Sprachsignal vorhanden: Wir können z.B. die Stimmung eines Sprechers erkennen oder ob er erkältet ist oder, wenn wir ihn gut genug kennen, seine Identität. Genau dies ist die Information, die wir hier aus dem Sprachsignal extrahieren wollen. Sie setzt sich aus den Charakteristika der *Resonanzräume*, die das Sprachsignal beeinflussen (Torso, Hals unter- und oberhalb der Larynx, Larynx, Mundhöhle und Nasalraum) zusammen. Die Frequenzen, die das Signal dominieren, nennt man *Formanten*. Dabei steckt die Sprachinformation in den ersten beiden und eingeschränkt auch im dritten Formanten. Ab dem dritten Formanten aufwärts finden sich die sprecherspezifischen Informationen. Weiterhin läßt sich der Beitrag der Anregung im Frequenzbereich oberhalb von

5 kHz vernachlässigen (vgl. [Hess 2000]). Die Anregung zieht spektrale Maxima bei den ganzzahligen Vielfachen der Grundfrequenz nach sich, während sich die Vokaltraktcharakteristik in der Form der globalen Einhüllenden des Spektrums wiederfindet (vgl. [Hess 1998]). Weitere Merkmale des Sprachsignals sind die sogenannten *prosodischen Merkmale*: Betonung, Intonation und Rhythmus. Sie können sowohl Sprachinformation übertragen als auch sprecherspezifische Verhaltensmuster wiederspiegeln.

Cepstralanalyse. Zur Analyse von Sprachsignalen wird häufig – sowohl bei der Sprechererkennung als auch bei der Spracherkennung – ein *Cepstrum* verwendet. Das komplexe Cepstrum einer stabilen Folge $x[n]$ definiert sich wie folgt:

$$\begin{aligned} \hat{x}[n] &= \tfrac{1}{2\pi} \textstyle\int_{-\pi}^{\pi} [\log X(e^{j\omega})] e^{j\omega n} d\omega \\ &= \tfrac{1}{2\pi} \textstyle\int_{-\pi}^{\pi} [\log |X(e^{j\omega})| + j\Phi(X(e^{j\omega}))] e^{j\omega n} d\omega. \end{aligned} \tag{3}$$

wobei $X(e^{j\omega})$ die Fouriertransformierte von $x[n]$ ist. $|X(e^{j\omega})|$ bezeichnet dabei das Amplituden- und $\Phi(X(e^{j\omega}))$ das Phasenspektrum. Damit das komplexe Cepstrum existiert, muß $\log[X(e^{j\omega})]$ zusätzlich noch bestimmte Konvergenzbedingungen einhalten. Wenn $x[n]$ reell ist, ist auch $\hat{x}[n]$ reell (vgl. [Oppenheim & Schafer 1999]).

Die Trennung von Betrag und Phase in zwei Summanden ergibt sich aus der Definition des komplexen Logarithmus':

$$\log[X(e^{j\omega})] = \log[|X(e^{j\omega})| e^{j\Phi X(e^{j\omega})}] = \log |X(e^{j\omega})| + j\Phi X(e^{j\omega}). \tag{4}$$

Vernachlässigt man die Phase, erhält man das reelle Cepstrum:

$$c_x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \log |X(e^{j\omega})| e^{j\omega n} d\omega. \tag{5}$$

Da hier die Phaseninformation verlorengeht, ist das reelle im Gegensatz zum komplexen Cepstrum nicht invertierbar.

Das Cepstrum wird verwendet, um die Vokaltraktcharakteristik von der Anregungscharakteristik zu trennen (vgl. [Hess 1998]): Das Spektrum eines Sprachsignals läßt sich als die Multiplikation der Anregung $P(n)$ mit der Einhüllenden $H(n)$ darstellen:

$$S(n) = H(n) \cdot P(n). \tag{6}$$

Die Logarithmierung des Spektrums bewirkt, daß die Multiplikation in eine Addition übergeht:

$$\log S(n) = \log H(n) + \log P(n). \tag{7}$$

Diese Addition bleibt bei der Rücktransformation erhalten; dabei bezeichnet d eine Verzögerung (Delay) in Bezug auf einen fiktiven Zeitpunkt t_0:

$$s(d) = h(d) + p(d). \tag{8}$$

Die auf $P(n)$ entfallenden Signalanteile befinden sich nun oberhalb der Grundperiode T_0. Es läßt sich jedoch insbesondere bei weiblichen Sprechern nicht ausschließen, daß sich noch Signalanteile von $H(n)$ oberhalb von T_0 befinden.

2.3.2 Methoden der Sprechererkennung

Bei der Sprechererkennung unterscheidet man zwischen der *Sprecheridentifikation* und der *Sprecherverifikation*. Bei der Verifikation behauptet der Sprecher, eine bestimmte Person zu sein. Das Erkennungsverfahren muß nun diese Behauptung überprüfen. Bei der Identifikation sucht das Verfahren in einer Signaturdatenbank nach einer Signatur, die der des eingehenden Signals gleicht.

Weiterhin unterscheidet man zwischen *textabhängigen* und *textunabhängigen* Verfahren. Bei den textabhängigen Verfahren muß der Sprecher einen vordefinierten Satz sagen, während bei den textunabhägigen Verfahren der konkrete Wortlaut nicht relevant ist.

Um die Sicherheit des Verfahrens zu erhöhen, können außerdem sogenannte *Antisprechermodelle* benutzt werden: Zu jedem in der Datenbank registrierten Sprecher wird eine Gruppe von Antisprechern gewählt, von denen das eingehende Sprachsignal abzugrenzen ist.

Im folgenden sollen einige der bekanntesten Sprechererkennungsverfahren kurz vorgestellt werden.

Pattern Matching. Zu den bekanntesten Pattern Matching-Verfahren gehören Dynamic Time Warping (DTW), Nearest Neighbor (NN) und stochastische Modelle (vgl. [Campbell 1999]).

Dynamic Time Warping. DTW-Verfahren berücksichtigen Schwankungen in der Sprechgeschwindigkeit. Dabei werden zeitbezogene Indizes j einer Signatur aus der Datenbank (Template) x durch eine Mappingfunktion $j(i)$ den Indizes i der Signatur des eingehenden Signals (Sample) x' zugeordnet, um diese Schwankungen auszugleichen. Die Zuordnung erfolgt mithilfe einer Distanzfunktion:

$$Dist = \sum_{i=1}^{M} d(x_i, x'_{j(i)}) \tag{9}$$

Der Algorithmus wählt die Mappingfunktion $j(i)$ derart, daß $Dist$ minimiert wird. Dabei müssen zusätzliche Bedingungen (Constraints) für den Abstand zwischen i und j eingehalten werden (vgl. Abbildung 4).

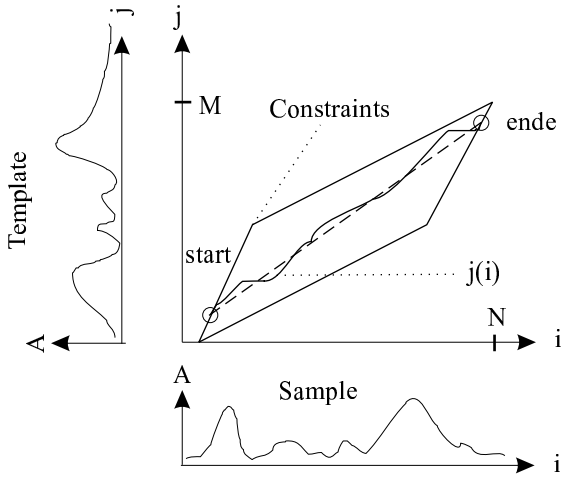

Abbildung 4: DTW-Trajektorie

Nearest Neighbor. Bei NN-Verfahren werden aus den Trainingsdaten Cluster gebildet, die einen Sprecher repräsentieren. Dabei können z.B. auch die Trainingsvektoren ohne weitere Bearbeitung als Cluster dienen. Aus datenschutztechnischen Gründen sollten in diesem Fall jedoch nicht die Rohdaten, sondern lediglich anhand der Trainingsdaten berechnete Signaturen gespeichert werden (vgl. Abschnitt 5.3). Beim Vergleich wird nach dem Cluster mit dem kleinsten Abstand zum eingehenden Sample gesucht.

Ein bekanntes NN-Verfahren ist das *VQ Source Modelling*. Für jeden phonetischen Cluster eines Sprechers (z.B. für die verschiedenen Laute seiner Sprache) wird ein Template x' im Frequenzbereich gebildet und in einem Codebuch C abgelegt, daß diesen Sprecher repräsentiert. Für den Frequenzgang jedes Frames x_i eines Samples wird dann im Codebuch der Eintrag x' mit der minimalen Distanz d ermittelt und die Summe über diese Distanzen gebildet:

$$Dist = \sum_{i=1}^{M} \min_{x' \in C} \{d(x_i, x')\} \tag{10}$$

VQ-Verfahren berücksichtigen im Gegensatz zu DTW-Verfahren keine temporalen Informationen.

Stochastische Verfahren. Stochastische Verfahren berechnen im Gegensatz zu Pattern Matching-Verfahren keine Distanz, sondern die Wahrscheinlichkeit, daß das eingehende Signal (Beobachtung) von einem bestimmten in einer Datenbank registrierten Sprecher stammt. Die Wahrscheinlichkeitsdichte einer Beobachtung ist vom Sprecher abhängig und wird anhand von Trainingsdaten ermittelt (vgl. [Campbell 1999], [Zhang 2000]).

Ein weitverbreitetes stochastisches Verfahren ist das *Hidden Markov Modelling (HMM)*. Ein HMM besteht aus einem endlichen Automaten, dessen Zustandsübergänge durch eine Wahrscheinlichkeitsdichte $a_{ij} = p(s_i|s_j)$ gesteuert werden. Die Wahrscheinlichkeit, daß in einem Zustand s_i eine Beobachtung x gemacht wurde, wird durch eine dem Zustand individuell zugeordnete Wahrscheinlichkeitsdichte $p(x|s_i)$ berechnet (vgl. Abbildung 5). Die Wahrscheinlichkeit für eine Folge von Frames x_i in Bezug auf ein Modell M berechnet sich dann durch:

$$p(x(1;L)|M) = \sum_{AlleZustandsfolgen} \prod_{i=1}^{L} p(x_i|s_i)p(s_i|s_{i-1}) \tag{11}$$

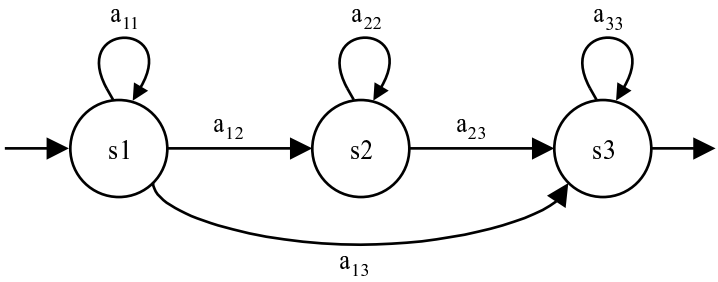

Abbildung 5: Beispiel für ein HMM, $a_{ij} = p(s_i|s_j)$

Ein *Gaussian Mixture Model (GMM)* nimmt im Gegensatz zum HMM keine Abhängigkeiten zwischen verschiedenen phonetischen Klassen an. Es wird für jeden Sprecher ein Modell λ trainiert (12). Dabei sind $\vec{\mu}_i$ der Erwartungsvektor und Σ_i die Kovarianzmatrix einer mehrdimensionalen Gaußschen Normalverteilung.

$$\lambda = \{p_i, \vec{\mu}_i, \Sigma_i\}, \qquad \text{mit } i = 1, \ldots, M. \tag{12}$$

Die Wahrscheinlichkeit eines beobachteten D-dimensionalen Vektors $\vec{x}$ unter dem Modell λ berechnet sich aus den Komponentendichten (component densities) $b_i(\vec{x})$ und den Gewichten (mixture weights) p_i (13). Bei den

Komponentendichten $b_i(\vec{x})$ handelt es sich um D-dimensionale Gaußfunktionen (14).

$$p(\vec{x}|\lambda) = \sum_{i=1}^{M} p_i \cdot b_i(\vec{x}), \qquad \text{mit } \sum_{i=1}^{M} p_i = 1,\ i = 1, \ldots, M. \tag{13}$$

$$b_i(\vec{x}) = \frac{1}{(2\pi)^{D/2} \cdot \det(\Sigma_i)^{1/2}} \cdot e^{-\frac{1}{2}(\vec{x}-\vec{\mu}_i)^T \Sigma_i^{-1} (\vec{x}-\vec{\mu}_i)}. \tag{14}$$

Es gibt drei verschiedene Möglichkeiten, den Modellen Kovarianzmatrizen zuzuweisen:

- Eine eigene Kovarianzmatrix für jede Gaußsche Komponente $b_i(\vec{x})$ (nodal covariance),
- eine gemeinsame Kovarianzmatrix für alle Gaußschen Komponenten (grand covariance), oder
- eine gemeinsame Kovarianzmatrix für alle Sprechermodelle (global covariance).

Dabei kann die Kovarianzmatrix eine reguläre oder eine Diagonalmatrix sein (vgl. [Reynolds & Rose 1995]).

Künstliche Neuronale Netze. Bei Verfahren, die Künstliche Neuronale Netze (KNN) verwenden werden die bekannten Sprecher nicht wie z.B. beim DTW durch individuelle Templates repräsentiert. Stattdessen wird die Funktion trainiert, die alle bekannten Sprecher am besten voneinander abgrenzt. Da es nach wie vor keinen Konsenz darüber gibt, welche Merkmale eines Sprachsignals einen Sprecher am besten charakterisieren haben KNN den Vorteil, daß diese Merkmale nicht exakt modelliert werden müssen. Ein wesentlicher Nachteil von KNN liegt darin, daß sie bei jedem neu hinzukommenden Sprecher neu trainiert werden müssen. Bekannte Verfahren sind hier *Time Delay Neural Networks (TDNN)*, *Recurrent Networks* und *Hybride Netze* (vgl. [Zhang 2000])

3 Implementation eines Sprechererkennungsverfahrens mit MATLAB

Im Rahmen dieses Projekts wurde ein Dynamic Time Warping-Algorithmus in MATLAB implementiert. Dazu wurde eine aus einem Cepstrum und aus Delta-Koeffizienten bestehende Signatur berechnet (vgl. [Shieh 1995], [Furui 1981]). Es handelt sich hierbei um ein textabhängiges Verfahren.

3.1 Signaturberechnung

Die Vorverarbeitung eines aufgezeichneten Sprachsignals durchläuft mehrere Schritte:

- Als erstes muß das Sprachsignal aus der Gesamtsignallänge *extrahiert* werden, denn der Sprecher fängt erst nach einer gewissen Reaktionszeit an zu sprechen, und in der Regel dauert das Sprachsignal auch nicht bis zum Ende des aufgezeichneten Signals an (vgl. Abbildung 6).

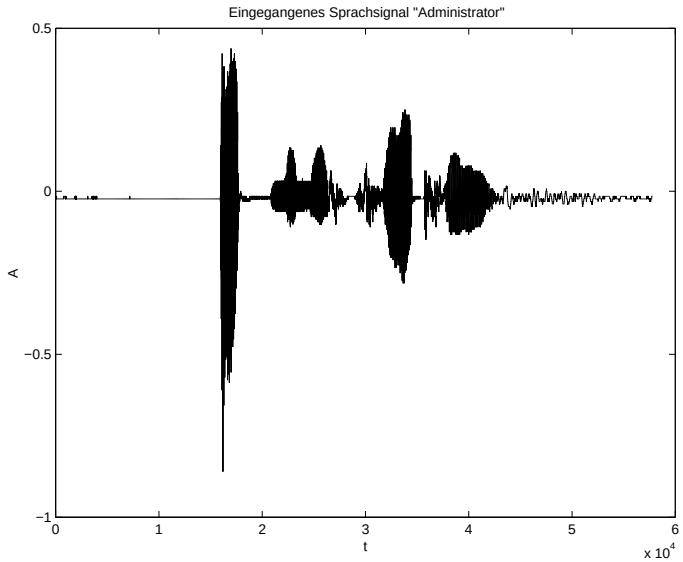

Abbildung 6: Beispiel für ein eingehendes Sprachsignal

Um das Sprachsignal zu extrahieren wird das Gesamtsignal zuerst in Frames eingeteilt. Dabei entspricht die Energie eines einzelnen Frames der Autokorrelation 0. Grades (vgl. [Shieh 1995], S. 8). Das Signal wird dann abgeschnitten, wenn die Energie unter einen definierten Schwellwert sinkt (vgl. Abbildung 7).

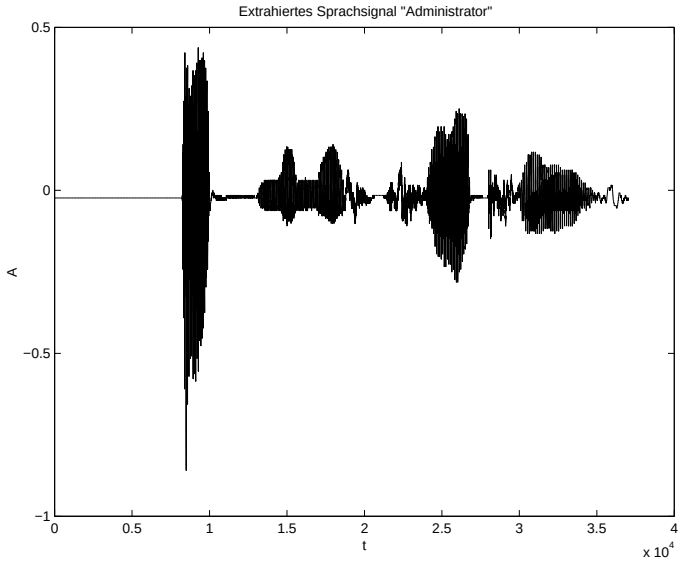

Abbildung 7: Beispiel für ein extrahiertes Sprachsignal

- Nach der Extraktion des Sprachsignals werden die hohen Frequenzen durch eine *Preemphasisfunktion* (15) angehoben, da hochfrequente Signalanteile weniger Energie besitzen als die niedrigeren Frequenzen. Dafür wird das Signal im Frequenzbereich mit der Preemphasisfunktion multipliziert. Üblicherweise wird hier $\alpha = 0.95$ gewählt (vgl. [Shieh 1995], S. 10; Abbildung 8).

$$H(z) = 1 - \alpha z^{-1}, \qquad 0.9 \leq \alpha \leq 1. \tag{15}$$

- Das mit Präemphase versehene Signal wird in Frames eingeteilt und die einzelnen Frames anhand eines *Hammingfensters* (16) gefiltert.

$$hamm(n) = 0.54 - 0.46\cos(\frac{2\pi n}{N-1}), \qquad 0 \leq n \leq N-1. \tag{16}$$

- Für jeden Frame t werden die *Autokorrelationskoeffizienten* (17) des Grades p berechnet (vgl. [Shieh 1995], S. 12; Abbildung 9). An dieser Stelle findet eine Datenreduktion statt; das Originalsignal ist nicht mehr rekonstruierbar.

$$r_t(m) = \sum_{n=0}^{N-1-m} x_t(n)x_t(n+m), \qquad m = 0, 1, \ldots, p. \tag{17}$$

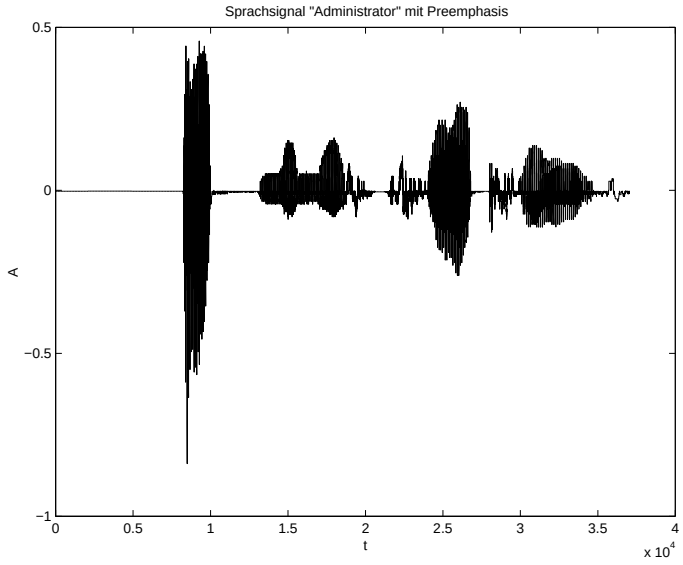

Abbildung 8: Beispiel für ein Sprachsignal nach Anhebung der hohen Frequenzen

- Aus den Autokorrelationskoeffizienten wird ein *Cepstrum* ermittelt (vgl. Abbildung 10; Abschnitte 2.3.1, 3).
- Anhand des Cepstrums werden Polynomialkoeffizienten 1. Grades (*Delta-Koeffizienten*) (18) berechnet, die für jedes Frame j die Steigung der Cepstralkoeffizienten innerhalb 9 umliegender Frames repräsentieren. Die Delta-Koeffizienten werden den Cepstralkoeffizienten hinzugefügt (vgl. [Furui 1981], S. 256).

$$b = \frac{\sum_{j=1}^{9} x_j(j-5)}{\sum_{j=1}^{9}(j-5)^2}. \tag{18}$$

- Zum Abschluß wird das Signal anhand einer *Gewichtsfunktion* (27) normalisiert.

3.2 Distanzermittlung

Zum Berechnen der Mappingfunktion wurde das Verfahren aus [Furui 1981] verwendet. Die kürzere der zu vergleichenden Signaturen wird als *Guide* und die längere als *Slave* bezeichnet. Die Mappingfunktion $w(n)$ ist eine Funktion von den Guideindizes n, $n = 1, \ldots, N$ in die Slaveindizes m, $m = 1, \ldots, M$ (vgl. Abbildung 4). Dabei muß $w(n)$ folgende Constraints erfüllen:

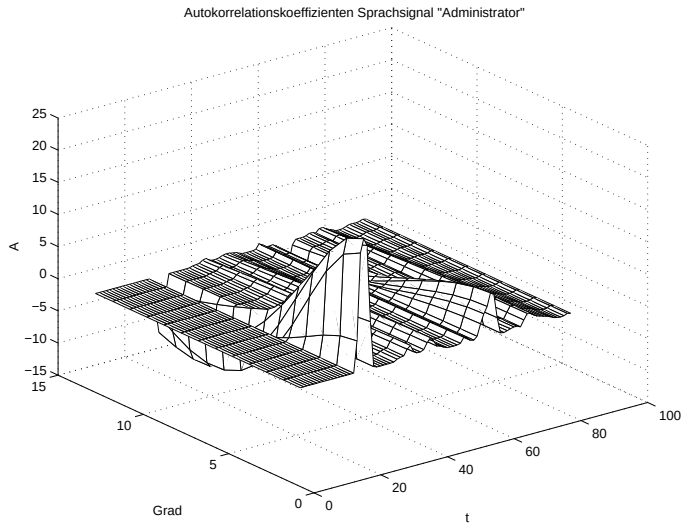

Abbildung 9: Beispiel für Autokorrelationskoeffizienten

$$w(n+1) - w(n) = 0, 1, 2, \qquad \text{falls } w(n) \neq w(n-1). \tag{19}$$

$$w(n+1) - w(n) = 1, 2, \qquad \text{falls } w(n) = w(n-1). \tag{20}$$

$$1 \leq w(1) \leq \delta + 1. \tag{21}$$

$$M - \delta \leq w(N) \leq M. \tag{22}$$

$$\max w(n) = M, \quad N - \delta \leq n \leq N. \tag{23}$$

$$\frac{M}{N} n - m_0 \leq w(n) \leq \frac{M}{N} n + m_0. \tag{24}$$

Die Constraints (19) - (20) bewirken, daß die Steigung der Mappingfunktion zwischen 0.5 und 2 liegt. In (21) - (23) wird der jeweilige maximale Unterschied zwischen Guide- und Slaveposition am Start und am Ende der Funktion durch δ ausgedrückt. Dabei kann der Endwert des Slaves vor dem Endwert des Guides erreicht werden; in diesem Fall werden höhere Guidewerte nicht mehr gemappt. Der letzte Constraint (24) bewirkt, daß sich die Trajektorie im Verlauf der Funktion nicht aus einem bestimmten Bereich herausbewegt.

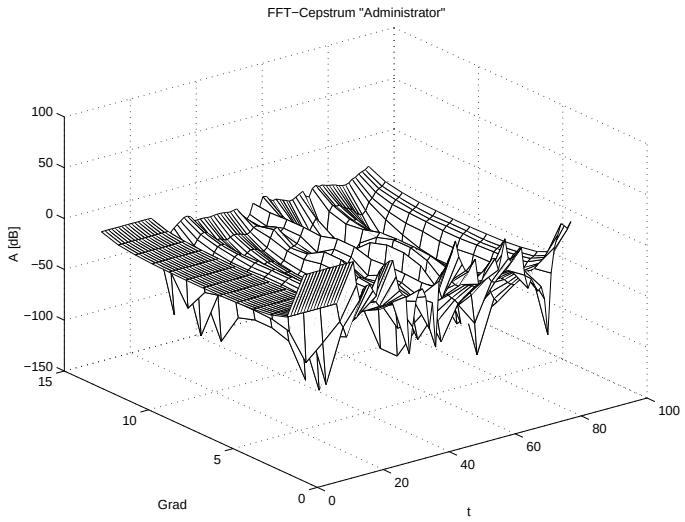

Abbildung 10: Beispiel für FFT-Cepstrum

Die Mappingfunktion $w(n)$ wird so gewählt, daß die Summe der Abstände D zwischen dem Guide an der Stelle n und dem Slave an der Stelle $w(n)$ minimal ist:

$$D_T = \min_{w(n)} \sum_{n=1}^{N} D(Guide(n), Slave(w(n))). \tag{25}$$

Als lokales Abstandsmaß D wurde (26) gewählt. Dabei bezeichnet der Index i die Nummer des verglichenen Merkmals, und g_i ist eine Gewichtsfunktion, die bei der Erstellung des Templates ermittelt wird (vgl. Abbildung 4.1).

$$D(Guide(n), Slave(w(n))) = \sum_{i=1}^{K} g_i^2 (guide_i(n) - slave_i(w(n)))^2. \tag{26}$$

3.3 Implementierung

Der in den Abschnitten 3.1 und 3.2 beschriebene Algorithmus wurde in MATLAB implementiert (Quellcode siehe Anhang A). Es wurde für sämtliche Vorverarbeitungsschritte eine Framerate von 10 ms und eine Framebreite von 30 ms gewählt.

Die für die Extraktion des Sprachsignals aus dem Gesamtsignal benötigten Parameter wurden experimentell ermittelt. Hierfür mußte als erstes ein geeigneter Schwellwert gefunden werden. Dafür wurde ausgenutzt, daß man mit einer Reaktionszeit des Sprechers von ca. 1 s rechnen kann: Die Energie der ersten drei Frames wurde gemittelt, als Grundgeräuschpegel betrachtet, und mit dem Faktor 10 als Schwellwert benutzt.

Um den Sprachanteil des Signals zu finden, wurde bei 1/6 der Signallänge begonnen, nach einem Frame zu suchen, dessen Energiewert den Schwellwert übersteigt. Dieser Frame wurde als Startpunkt gewählt, um das Signal nacheinander in beiden Richtungen zu durchsuchen, bis jeweils der Energiewert des aktuellen Frames unter den Schwellwert sank. Die Wahl des Startpunktes wurde aus Effizienzgründen getroffen und erschien als ausreichend, um die Sprachanteile im Signal zu finden.

Weiterhin mußten Sprechpausen berücksichtigt werden, in denen die Energie eines Frames zwar unter den gewählten Schwellwert sinkt, das Signal aber dort noch nicht abgeschnitten werden darf. Dazu wurde bei der Durchsuchung des Signals mit Sprüngen von 2 s begonnen, die jedesmal, wenn die Energie des nächsten zu untersuchenden Frames unter den Schwellwert sank in Schritten von einer Framebreite bis hinunter zur Framebreite verkleinert wurden.

Für die Berechnung des Cepstrums wurde im Gegensatz zu [Furui 1981] und [Shieh 1995] kein LPC-Cepstrum, sondern ein FFT-Cepstrum gewählt, da MATLAB eine sehr effiziente Implementation der Fast Fourier Transformation (FFT) und der Inversen FFT besitzt. Dazu wurden zuerst die Autokorrelationskoeffizienten des Grades 12 ermittelt. Danach wurde die Amplitude der FFT berechnet, logarithmiert und anhand der inversen FFT wieder zurück in den Zeitbereich transformiert.

4 Labortests und Bewertung

4.1 Labortests

Für die Labortests wurden fünf Sprecherinnen mit einer Samplingrate von 22.05 kHz aufgenommen und die Aufnahmen als WAV-Dateien gespeichert. Jede Sprecherin sagte die Wörter „Login“ und „Administrator“ je 10 Mal.

Davon wurden jeweils die ersten fünf Äußerungen für die Berechnung des Templates und die anderen fünf als Testdaten verwendet.

Zum *Erstellen der Templates* wurde für jede der ersten fünf Äußerungen ein Cepstrum berechnet (`calcepstrum.m`): In einem ersten Schritt wurde das Sprachsignal extrahiert (`isolate.m`), dann die Präemphase hinzugerechnet (`preemphasis.m`) und die Autokorrelationskoeffizienten ermittelt (`frameblock.m`), um zum Schluß die Amplitude des FFT-Cepstrums zu erhalten (`fftcepstrum.m`). Anschließend wurden auf Basis der Cepstralkoeffizienten die Delta-Koeffizienten berechnet (`deltakoeff.m`).

Anhand dieser Signaturen wurde dann ein normiertes Template und die Gewichtsfunktion berechnet (`maketemplate.m`). Dazu wurden die ersten beiden Signaturen mit einer neutralen Gewichtsfunktion gemappt (`warp.m`) und anhand des Mappings ein Mittelwert berechnet. Dieser Mittelwert wurde dann mit der dritten Signatur verglichen und ein zweiter Mittelwert aus dem ersten Mittelwert und der dritten Signatur gebildet. Aus diesem wurde schließlich zusammen mit dem Mittel aus der vierten und fünften Signatur ein Template gebildet.

Die Gewichtsfunktion g_i wird für jeden Sprecher individuell ermittelt. Für einen Koeffizienten i berechnet sich g_i aus dem Mittelwert der Unterschiede zwischen den Signaturen eines Einzelsprechers (intraspeaker variability, vgl. [Furui 1981]):

$$g_i = 1/E_{j,k\neq l} \sum_{n=1}^{N} (t_{ijk}(n) - t_{ijl}(w(n)))^2. \tag{27}$$

Dabei bezeichnet n den aktuellen Frame der k. Äußerung des Sprechers j und E den Mittelwert.

Um die Gewichtsfunktion zu erhalten wurde das Template mit den Signaturen der ersten fünf Äußerungen verglichen (`makeweight.m`). Dazu wurde anhand einer neutralen Gewichtsfunktion die Distanz (26) zwischen dem Template und der jeweiligen Signatur berechnet und das arithmetische Mittel über diese Distanzen gebildet. Zum Abschluß wurde das Template anhand der Gewichtsfunktion normalisiert (`normalise.m`).

Für den *Vergleich* einer Äußerung mit einem Template wurde *a priori* kein Threshold gewählt, sondern lediglich die Distanz berechnet (`calcdist.m`), um im weiteren Verlauf die ERR zu ermitteln. Dazu wurden wiederum zuerst ein Cepstrum (`calcepstrum.m`) und die Delta-Koeffizienten berechnet (`deltakoeff.m`). Danach wurde diese Signatur anhand der Gewichtsfunktion normalisiert (`normalise.m`) und mit dem Template verglichen (`warp.m`).

4.1.1 Tests ohne Störsignale

Von den Testpersonen wurden zwei verschiedene Wörter aufgenommen, um das Verfahren mit zwei verschiedenen Signallängen zu testen. Dabei war die Äußerung „Administrator“ ca. 1,8 s, und die Äußerung „Login“ ca. 1,4 s lang. Pro Testperson wurden für jeden Test jeweils fünf autorisierte und acht nicht autorisierte Loginversuche simuliert. Die Berechnungen wurden auf einer Sun Blade 100 ausgeführt. Dort dauerte das Erstellen eines Templates für die Äußerung „Administrator“ ca. 4,4 min und für die Äußerung „Login“ ca. 3,6 min, und ein Vergleich dauerte für die Äußerung „Administrator“ ca. 1,7 min und für die Äußerung „Login“ ca. 0,4 min.

Beim ersten Test wurde die ERR für die Menge der Ergebnisse für alle Sprecher gebildet. Dadurch ergab sich für die Äußerung „Administrator“ eine Equal Error Rate von 50 %, und für die Äußerung „Login“ eine Equal Error Rate von 37,5,% (vgl. Tabelle 1; Abbildung 11).

EER für alle Sprecher	
„Administrator“	50,0 %
„Login“	37,5 %
Gesamt	43,75 %

Tabelle 1: Equal Error Rate für einheitlichen Threshold

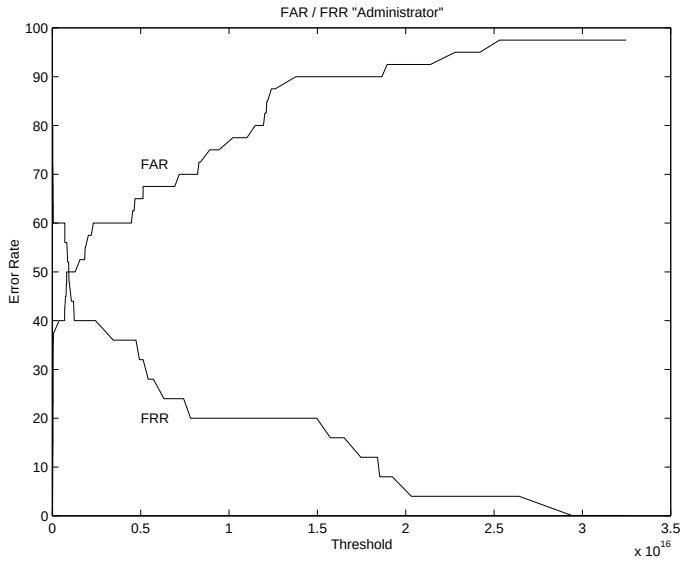

Abbildung 11: FAR und FRR für die Äußerung „Administrator“

Weist man jeder Testperson einen individuellen Threshold zu, bekommt man erheblich bessere Resultate (vgl. Tabelle 2): Für die Äußerung „Administrator“ ergab sich im Mittel eine Equal Error Rate von 35,5 %, und für die Äußerung „Login“ von 28,1 %.

Insgesamt läßt sich also feststellen, daß die ERR bei der Äußerung „Administrator“ trotz der größeren Länge des Signals oberhalb der ERR bei der Äußerung „Login“ liegt.

EER			
„Administrator“	Framerate 10ms	Framerate 5ms	ohne Delta-Koeff.
Sprecher 1	0,0 %	0,0 %	0,0 %
Sprecher 2	40,0 %	60,0 %	40,0 %
Sprecher 3	50,0 %	62,5 %	60,0 %
Sprecher 4	50,0 %	50,0 %	50,0 %
Sprecher 5	37,5 %	25,0 %	50,0 %
Alle Sprecher	35,5 %	39,5 %	40,0 %
„Login“	Framerate 10ms	Framerate 5ms	ohne Delta-Koeff.
Sprecher 1	50 %	50,0 %	60,0 %
Sprecher 2	12,5 %	25,0 %	12,5 %
Sprecher 3	15,5 %	40,0 %	100,0 %
Sprecher 4	37,5 %	25,0 %	40,0 %
Sprecher 5	25,0 %	20,0 %	20,0 %
Alle Sprecher	28,1 %	32,0 %	46,5 %
Gesamt	31,8 %	35,75 %	43,25 %

Tabelle 2: Equal Error Rate für individuellen Threshold

Für einen zweiten Testlauf wurde die Framerate auf 5 ms herabgesetzt. Das Erstellen eines Templates dauerte für die Äußerung „Administrator“ ca. 8,7 min und für die Äußerung „Login“ ca. 6,5 min, und ein Vergleich dauerte für die Äußerung „Administrator“ ca. 2,2 min und für die Äußerung „Login“ ca. 0,7 min. Es ergab sich für die Äußerung „Administrator“ eine Equal Error Rate von 39,5 %, und für die Äußerung „Login“ eine Equal Error Rate von 32 %, was oberhalb der ERR bei den Tests mit einer Framerate von 10 ms liegt (vgl. Tabelle 2).

Bei einem weiteren Test wurde bei der Erstellung der Signaturen auf die Berechnung der Delta-Koeffizienten verzichtet. Das Erstellen eines Templates dauerte für die Äußerung „Administrator“ ca. 4 min und für die Äußerung

„Login“ ca. 3,2 min, und ein Vergleich dauerte für die Äußerung „Administrator“ ca. 1,6 min und für die Äußerung „Login“ ca. 0,3 min. Durch den Verzicht auf die Delta-Koeffizienten in der Signatur verschlechterten sich die Ergebnisse wie erwartet für beide Äußerungen (vgl. Tabelle 2): Für die Äußerung „Administrator“ ergab sich im Mittel eine Equal Error Rate von 40 %, und für die Äußerung „Login“ eine Equal Error Rate von 46,5 %.

4.1.2 Tests mit Störsignalen

Um Umweltgeräusche zu simulieren wurde zu den Testsignalen beim Test auf autorisierte Identifikationsversuche ein Türschlagen hinzuaddiert. Die ERR verschlechterte sich wie erwartet für die Äußerung „Login“; für die Äußerung „Administrator“ jedoch ergab sich eine Verbesserung von 14 % (vgl. Tabelle 3). Es ergab sich im Mittel für die Äußerung „Administrator“ eine Equal Error Rate von 21,5 % und für die Äußerung „Login“ eine Equal Error Rate von 34 %.

EER		
„Administrator“	Türschlagen	Hintergrundgespräch
Sprecher 1	0,0 %	–
Sprecher 2	20,0 %	–
Sprecher 3	50,0 %	–
Sprecher 4	0,0 %	–
Sprecher 5	37,5 %	–
Alle Sprecher	21,5 %	
„Login“	Türschlagen	Hintergrundgespräch
Sprecher 1	50,0 %	50,0 %
Sprecher 2	0,0 %	–
Sprecher 3	20,0 %	–
Sprecher 4	60,0 %	–
Sprecher 5	40,0 %	–
Alle Sprecher	34 %	–
Gesamt	27,75 %	–

Tabelle 3: Equal Error Rate bei Störgeräuschen

Für eine Sprecherin lagen Daten vor, die durch eine Unterhaltung im Hintergrund gestört wurden. Diese Daten wurden an einem anderen Tag aufgenommen als die für das Template verwendeten Daten ohne Hintergrundgeräusch. Es ließ sich in diesem Fall keine Verschlechterung feststellen (vgl. Tabelle 3).

4.2 Bewertung

Insgesamt ließ sich feststellen, daß die bei diesen Tests vorliegenden kurzen Signallängen nicht auszureichen scheinen, um eine zufriedenstellende Robustheit des hier implementierten Verfahrens zu erhalten (ERR von 28,1 % im besten Fall). Es ließ sich jedoch feststellen, daß die Delta-Koeffizienten eine deutliche Verbesserung der Robustheit bewirken: Sie verursachten eine Senkung der ERR im Mittel um 11,45 %. Ebenso wurde eine Verbesserung durch die Definition individueller Thresholds erlangt (Senkung der ERR um 11,95 %). Eine Erhöhung der Framerate auf 5 ms brachte jedoch keine Verbesserung: In diesem Fall trat insgesamt eine Verschlechterung sowohl der Robustheit als auch der Effizienz ein.

Darüberhinaus wurde das Verfahren auf Empfindlichkeit gegenüber Störgeräuschen getestet. Bei Tests mit aufaddiertem Türschlagen ließen sich keine eindeutigen Aussagen machen: Für die Äußerung „Login“ verschlechterte sich die ERR um 5,9 %, während für die Äußerung „Administrator“ eine Verbesserung um 14 % eintrat. Beim Test mit durch ein Hintergrundgespräch gestörten Signalen blieb die Fehlerrate konstant, jedoch lagen diese Daten nur für einen Sprecher vor. Es wäre interessant zu testen, auf welche Art möglicher Hintergrundgeräusche das Verfahren empfindlich reagiert. Da man aber nicht von vornherein festlegen kann, welche Störgeräusche in realen Einsatzgebieten entstehen, wären hierfür Realtests erforderlich.

Weiterhin sollte getestet werden, wie robust das System gegenüber Veränderungen in der Stimme eines Sprechers ist, die z.B. durch Erkältung, Stimmbruch, Tageszeit und Stimmung hervorgerufen werden können. Auch hierfür wären Realtests hilfreich.

Weiterhin ist die hier vorgestellte MATLAB-Implementation sicherlich noch zu ineffizient (Dauer eines Vergleichs ca. 1,7 min bei einer Signallänge von ca. 1,8 s), um in dieser Form Akzeptanz bei den Nutzern zu finden.

5 Zur Eignung der Sprechererkennung für die biometrische Authentikation

5.1 Framework für Realtests

Realtests sind nicht nur wichtig, um die Robustheit eines Systems gegenüber Umwelteinflüssen zu beurteilen, sondern man muß für die Verbesserung eines Verfahrens auch die Ursachen einer Fehlentscheidung bestimmen können. Weiterhin ist auch die Geschwindigkeit des biometrischen Algorithmus' wichtig, um die Benutzbarkeit des Systems zu verbessern. Weitere zu bewertende Kriterien wären Nutzerverhalten und Handhabbarkeit des Systems sowie die Robustheit des Authentikationsprozesses gegenüber Angriffen (vgl. Abschnitt 5.3). Zu diesem Zweck wäre eine Einbindung des Verfahrens in den Logon-Prozess eines Betriebssystems hilfreich. Dafür wurde am Fachbereich Informatik der Universität Hamburg im Rahmen des Projekts „Biometrik" ein Framework entwickelt (vgl. [Brömme et al 2001]):

Um ein biometrisches Verfahren für den Authentikationsprozess eines Betriebssystems nutzbar machen zu können, muß als erstes auf das für die Aufzeichnung der Rohdaten benötigte Sensorsystem zugegriffen werden können. Je nach Verfahren muß vom Sensorsystem ein Videobild (z.B. für Iriserkennung), ein Video-Datenstrom (z.B. für Lippenbewegung), ein Audio-Datenstrom (z.B. für Sprechererkennung), ein einfacher Scan (z.B. für Fingerabdruck) oder ein Scan über die Zeit (z.B. für EKG und EEG) geliefert werden.

Um den Ablauf des Authentikationsprozesses für eine spätere Auswertung protokollieren zu können, benötigt man die Möglichkeit, im Verlauf des Verfahrens die Daten aufzuzeichnen, die durch den jeweils vorangegangenen Verarbeitungsschritt entstanden sind. Prinzipiell werden bei der biometrischen Authentikation folgende Verarbeitungsschritte durchgeführt:

1. *Start des Logins* → Login-Dialog
2. *Erfassung der Rohdaten durch das jeweilige Sensorsystem* → Biometrische Rohdaten & Sensorkalibrierungsdaten
3. *Vorverarbeitung* → Vorverarbeitete Daten
4. *Qualitätscheck und Normalisierung* → Normalisierte Daten; bei nicht ausreichender Datenqualität Abbruch des Loginprozesses
5. *Signalverarbeitung & Signaturberechnung* → Biometrische Signatur
6. *Vergleich mit einer Signaturdatenbank* → Ergebnis (ja/nein)

Für eine Beurteilung der Geschwindigkeit eines biometrischen Algorithmus' sind Zeitstempel für den Beginn und den Abschluss jedes Verarbeitungsschrittes hilfreich, um genauer abschätzen zu können, an welchen Stellen hier Verbesserungen möglich sind. Ein Effizienzgewinn wird bereits durch den Qualitätscheck erreicht, der bei einer unzureichenden Datenqualität dafür sorgt, daß der Loginversuch sofort abgebrochen wird.

Um ein biometrisches Authentikationsverfahren für eine bestimmte Umgebung und ihre spezifischen Umwelteinflüsse zu entwickeln, werden folgende Schritte vorgeschlagen:

1. Wahl des biometrischen Verfahrens.
2. Einbindung des biometrischen Authentikationsprozesses in das jeweilige Betriebssystem.
3. Implementation & Parametrisierung des biometrischen Verfahrens.
4. Labortests; ggf. Evaluation und Wiederholung ab Schritt 3.
5. Realtests; ggf. Evaluation und Wiederholung ab Schritt 3.
6. Abschließende Evaluation; wenn das Verfahren als prinzipiell für die gewählte Anwendungsumgebung ungeeignet erscheint Wiederholung ab Schritt 1.

5.2 Bewertung von Realtests

Um die Ursachen einer Fehlentscheidung des hier implementierten Sprechererkennungsalgorithmus' bestimmen zu können, wären die durch ein Ablaufprotokoll aufgezeichneten Daten sehr hilfreich. So kann z.B. bereits die Tatsache, daß sich aus einem eingehenden Signal kein Sprachsignal ausreichender Länge extrahieren ließ, zu einer fehlerhaften Zurückweisung des Benutzers durch das System führen. Die Kürze des extrahierten Signals kann wiederum z.B. durch einen zu geringen Pegel des Nutzsignals im Verhältnis zum Grundgeräuschpegel oder durch zu lange Sprechpausen verursacht werden. Für eine Evaluierung des hier implementierten DTW-Verfahrens wären in diesem Zusammenhang folgende Daten von Interesse:

- Bei der Vorverarbeitung und Signaturberechnung entstandene Daten
  - Eingegangenes Audiosignal
  - Extrahiertes Sprachsignal
  - Autokorrelationskoeffizienten
  - FFT-Cepstrum
  - Signatur
- Durch den Vergleichsalgorithmus entstandene Daten
  - Mappingfunktion
  - Distanzwert

Hierbei ist zu beachten, daß es sich bei dem eingegangenen Audiosignal und dem extrahierten Sprachsignal um biometrische Rohdaten handelt, deren permanente Speicherung unter Datenschutzgesichtspunkten bedenklich ist (vgl. Abschnitt 5.3). Alternativ könnte man sich hier auf die Protokollierung der Länge des extrahierten Sprachsignals beschränken.

Ein Problem, das sich bei Realtest ergibt ist festzustellen, ob das System richtig entschieden hat oder ob eine Fehlentscheidung vorliegt. Hierfür wäre es hilfreich, die Loginvorgänge vorort zu beobachten. Ein weiterer Gesichtspunkt, der sich nicht automatisiert erfassen läßt ist die Akzeptanz des Verfahrens durch die Benutzer und die Einfachheit der Handhabung.

5.3 Biometrische Authentikation und Datenschutz

Bei biometrischen Autentikationsverfahren werden biometrische Rohdaten der Benutzer gesammelt. Kombiniert man diese Daten mit personenbezogenen Daten, die – ggf. mithilfe nicht im System enthaltener Zusatzdaten – auf

die einzelnen Benutzer schließen lassen, ist die Wahrung der Rechte der Benutzer bezüglich des Datenschutzes nicht mehr gewährleistet. Deshalb sollte beim Entwurf und bei der Nutzung eines biometrischen Authentikationssystems folgendes beachtet werden (vgl. [Brömme 2001]):

- Es sollten nur solche Daten gesammelt werden, die für die Funktionalität des Systems unabdingbar sind.
- Die gesammelten Daten sollten nur zweckgebunden genutzt, d.h. ausschließlich für die Authentikation verwendet und insbesondere nicht an dritte weitergegeben werden.
- Die biometrischen Signaturen sollten derart berechnet werden, daß sich aus ihnen die biometrischen Rohdaten nicht rekonstruieren lassen.
- Es sollten keine biometrischen Rohdaten permanent gespeichert werden, insbesondere nicht in Verbindung mit personenbezogenen Daten.

Weiterhin muß darauf geachtet werden, daß das System bezüglich Angriffen wie Replay-Attacken, Einbruch in die Signaturdatenbank und Täuschungsversuchen (Kopie, Fälschung oder Ähnlichkeit eines eingehenden Signals) sicher ist.

6 Zusammenfassung und Ausblick

Unter dem Blickwinkel der biometrischen Authentikation wurde ein Dynamic Time Warping-Algorithmus für die Sprechererkennung in MATLAB implementiert. Weiterhin wurde der Nutzen von Realtests für die Weiterentwicklung von biometrischen Verfahren erörtert und die möglichen Auswirkungen biometrischer Authentikationsverfahren auf den Datenschutz diskutiert:

Bei der Verwendung von biometrischen Authentikationsverfahren ist darauf zu achten, daß die Rechte der Benutzer bezüglich des Datenschutzes gewahrt werden. In diesem Zusammenhang ist vor allem die permanente Speicherung von biometrischen Rohdaten bedenklich, und das System muß gegen Diebstahl von biometrischen Rohdaten und Signaturen geschützt werden.

Weiterhin muß berücksichtigt werden, daß eine Person eine biologische Charakteristik z.B. durch einen Unfall oder eine Krankheit verlieren kann oder eventuell erst gar nicht besitzt (z.B. dürfte es einem Taubstummen wohl schwerfallen, sich durch seine Stimme zu identifizieren). Für solche Personen müssen alternative Verfahren für die Authentikation zur Vefügung stehen.

Darüberhinaus können sich biologische Charakteristika durch den Alterungsprozess verändern. Diesem Effekt kann durch eine regelmäßige Neuregistrierung des Benutzers entgegengetreten werden. Das System sollte hier jedoch eine ausreichende Robustheit aufweisen, damit die Benutzbarkeit hierunter nicht leidet. Dieser Aspekt kann nur durch geeignete Studien erfaßt werden.

Um ein biometrisches Authentikationsverfahren abschließend bewerten zu können, sind umfangreiche Realtests erforderlich. Nur so kann das Verfahren gegenüber real auftretenden Störsignalen robust gemacht werden und beurteilt werden, ob seine Effizienz und Handhabbarkeit ausreichend ist, um Akzeptanz bei den Nutzern zu finden.

Literatur

[Brömme 2001] Brömme, Arslan: „A Discussion on Privacy Needs and (Mis)Use of Biometric IT-Systems". In: *Proceedings of the IFIP WG 9.6/11.7 Working Conference*,15. – 16. Juni 2001, Bratislava, 2001.

[Brömme et al 2001] Brömme, Arslan, Kronberg, Marcel, Ellenbeck, Oliver & Kasch, Oliver: *A Conceptual Framework for Testing Biometric Algorithms within Operating Systems' Authentication*, Fachbereich Informatik der Universität Hamburg, 2001.

[Campbell 1999] Campbell, Joseph P.: „Speaker Recognition". In: Jain, Anil K., Bolle, Ruud & Pankanti, Shrath (eds.): *Biometrics – Personal Identification in Networked Society*, Kluwer Academic Publishers, 1999.

[Fischer-Hübner 2001] Fischer-Hübner, Simone: *IT-Security and Privacy: Design and Use of Privacy-Enhancing Security Mechanisms*. LNCS Vol. 1958, Springer, 2001.

[Furui 1981] Furui, Sadaoki: „Cepstral Analysis Technique for Automatic Speaker Verification". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-29, No. 2*, 1981.

[Hess 1998] Hess, Wolfgang: *Grundlagen der Sprachsignalverarbeitung*. Vorlesungsskript, Institut für Kommunikationsforschung und Phonetik der Rheinischen Friedrich-Wilhelms-Universität Bonn, 1998.

[Hess 2000] Hess, Wolfgang: *Sprachsignalverarbeitung 1, 2*. Vorlesungsskript, Institut für Kommunikationsforschung und Phonetik der Rheinischen Friedrich-Wilhelms-Universität Bonn, 2000.

[Pfleeger 1997] Pfleeger, Charles P.: *Security in Computing*, Prentice-Hall, 2. Auflage, 1997.

[Reynolds & Rose 1995] Reynolds, Douglas A. & Rose, Richard C.: „Robust Text-Independent Speaker Identification Using Gaussian Mixture Speaker Models". In: *IEEE Transactions on Speech and Audio Processing, Vol. 3, No. 1*, 1995.

[Oppenheim & Schafer 1999] Oppenheim, Alan V. & Schafer, Ronald W.: *Zeitdiskrete Signalverarbeitung*. Oldenbourg Verlag, 3. durchgesehene Auflage, 1999.

[Shieh 1995] Shieh, Woei-Chyang: *Automatic Speaker Recognition Using Neural Networks and Vector Quantization*. Dissertation, Universität Bremen, 1995.

[Zhang 2000] Zhang, David D.: *Automated Biometrics – Technologies and Systems*, Kluwer Academic Publishers, 2000.

A MATLAB-Code

A.1 addnoise.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inputs:
%%%% Signal1 - Signal row vector
%%%% Noise - noise signal row vector
%%%% output: signal + noise
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function[Signal] = addnoise(Signal1,Noise)

[n,m]=size(Signal1);
[p,o]=size(Noise);

if m>o
  Noise=[zeros(p,floor((m-o)/2))
         Noise
         zeros(p,ceil((m-o)/2))];
elseif m<o
  Noise=Noise(:,floor((o-m)/2):o-ceil((o-m)/2)-1);
end

Signal=[];
for i=1:p
  temp=Signal1(i,:) + Noise(i,:);
  Signal=[Signal; temp];
end
```

A.2 autocorr.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inputs:
%%%%  oder - order of autocorrelation
%%%%  signal - row vector
%%%% output: autocorrelation coefficient of order order
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ACoeff]= autocorr(order,signal)

[row,N] = size(signal);
ACoeff=0;
for n=1:(N-order),
  ACoeff=ACoeff+signal(1,n)*signal(1,(n+order));
end
```

A.3 autocorriter.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inputs:
%%%%  order - order of autocorrelation
%%%%  signal - row vector
%%%% output: autocorrelation coefficients from order 0
%%%%         to order order - column vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ACoeffs]= autocorriter(order,signal)

ACoeffs=[];
for n=0:order,
  ACoeffs = [ACoeffs;autocorr(n, signal)];
end
```

A.4 calcdist.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inputs:
%%%%  template - row vector
%%%%  signal - row vector
%%%%  weight - weighting function column vector
%%%%  samplingRate - sampling rate of signal [Hz]
%%%%  windowWidth - width of frames [ms]
%%%%  frameRate - [ms]
%%%%  order - order of Cepstral & delta coefficients
%%%% output: template-signal distance
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [dist]= calcdist(template,signal,weight,
                 samplingRate,windowWidth,frameRate,order)

signal=calcepstrum(signal,samplingRate,windowWidth,
                   frameRate,order);

signal=[signal ; deltakoeff(signal)];
signal=normalise(signal,weight);

[warped1,warped2, dist]= warp(template,signal,weight);
```

A.5 calcepstrum.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inputs:
%%%% - Speech signal row vector
%%%% output: abs of FFT-Cepstrum
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Cepstrum] =
 calcepstrum(Signal,samplingRate,windowWidth,frameRate,order)

Signal=isolate(Signal,samplingRate,windowWidth);
Signal=preemphasis(Signal);
Signal=frameblock(Signal,windowWidth,
                  samplingRate,frameRate,order);
Cepstrum=abs(fftcepstrum(Signal));
```

A.6 deltakoeff.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% input: Cepstrum - cepstral coefficients
%%%% output: delta coefficioents
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function[Delta] = deltakoeff(Cepstrum)

Delta=[];
[m,n]=size(Cepstrum);
Zero=zeros(m,4);
Cepstrum = [Zero Cepstrum];
Zero=zeros(m,5);
Cepstrum = [Cepstrum Zero];
[m,n]=size(Cepstrum);
Sum2=0;
for j=1:9
 Sum2=Sum2+(j-5)^2;
end
for i=1:n-9
  Frame=Cepstrum(:,i:i+9);
  Sum1=0;
  for j=1:9
   Sum1=Sum1+Frame(:,j).*(j-5);
  end
  Delta=[Delta Sum1./Sum2];
end
```

A.7 fftcepstrum.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inputs:
%%%%  signal - speech signal row vector
%%%% output: FFTCepstrum - ifft(log(abs(fft(signal))))
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [FFTCepstrum]= fftcepstrum(signal)

FFTCepstrum=ifft(log(abs(fft(signal))));
```

A.8 frameblock.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inputs:
%%%%  signal - mono speech signal row vector
%%%%  samplingRate - sampling rate of signal [Hz]
%%%%  windowWidth - width of frames [ms]
%%%%  frameRate - [ms]
%%%%  order - order of autocorrelation coefficients
%%%% output: autocorrelation coefficients
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [reduced]= frameblock(signal,windowWidth,
                               samplingRate,framerate,order)

framesize=floor(windowWidth*samplingRate/1000);
framerate=floor(framerate*samplingRate/1000);

reduced=[];
[m,length]=size(signal);
n=1;

while n<(length-framesize),
  frame=signal(1,n:(n+framesize)).*hamm(windowWidth,samplingRate);
  reduced = [reduced autocorriter(order,frame)];
  n=n+framerate;
end
```

A.9 hamm.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inputs:
%%%%  windowWidth - width of the hamming window [ms]
%%%%  samplingRate - sampling rate of the speech signal
%%%% output: Hamming Window [0;windowWidth]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [hammingWindow]= hamm(windowWidth,samplingRate)

windowWidth=windowWidth*samplingRate/1000;
t=0:1/samplingRate:windowWidth/samplingRate;
p=windowWidth/samplingRate;

hammingWindow=(0.54-0.46*cos(2*pi*t/(p)));
```

A.10 isolate.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inputs:
%%%%  signal - mono speech signal row vector
%%%%  samplingRate - samplingRate of this signal [Hz]
%%%%  windowWidth - frame size for energy calculation [ms]
%%%% output: extracted mono speech signal row vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [extracted]= isolate(signal,samplingRate,windowWidth)

[m,signallength]=size(signal);
length=floor(windowWidth*samplingRate/1000)*2;
skip=windowWidth;
initbigskip=floor(samplingRate*2);

energy1=autocorr(0,signal(1,1:length));
energy2=autocorr(0,signal(1,length+1:2*length));
energy3=autocorr(0,signal(1,2*length+1:3*length));
energy=(energy1+energy2+energy3)/3;
threshold=10*energy;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% find end of signal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
start=floor(signallength/6);
bigskip=initbigskip;

energy=autocorr(0,signal(1,start:(start+length-1)));
```

```
while energy<threshold & (start+2*skip+length-1)<signallength
  start=start+2*skip;
  energy=autocorr(0,signal(1,start:(start+length-1)));
end
n=start;

for i=1:2
  while(bigskip>2*skip)
    while ((((n+2*bigskip)+length-1)<signallength) & (energy>threshold)),
      n=n+bigskip;
      energy=autocorr(0,signal(1,(n+bigskip):((n+bigskip)+length-1)));
    end
    bigskip=bigskip-skip;
  end

  while ((((n+2*skip)+length-1)<signallength) & (energy>threshold)),
    n=n+skip;
    energy=autocorr(0,signal(1,n:(n+length-1)));
  end
  energy=threshold+1;
  bigskip=initbigskip;
end
endpoint=n;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% find beginning of signal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
n=start;
bigskip=200*skip;
energy=autocorr(0,signal(1,n:(n+length-1)));

while(bigskip>2*skip)
  while (((n-2*bigskip)>0) & (energy>threshold)),
    n=n-bigskip;
    energy=autocorr(0,signal(1,(n-bigskip):((n-bigskip)+length-1)));
  end
  bigskip=bigskip-skip;
end

while (((n-2*skip)>0) & (energy>threshold)),
  n=n-skip;
  energy=autocorr(0,signal(1,(n-skip):((n-skip)+length-1)));
end

extracted=signal(1,n:endpoint);
```

A.11 maketemplate.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% input: Cepstra+Deltakoeff
% outputs: Normalised template + weighting function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Normtemplate,Weight]=maketemplate(C1,C2,C3,C4,C5);

[m,n]=size(C1);
tempweight=ones(m,1);
[warp1,warp2,cost]=warp(C1,C2,tempweight);
Template=(warp1+warp2)/2;
[warp1,warp2,cost]=warp(Template,C3,tempweight);
Template=(warp1+warp2)/2;
[warp1,warp2,cost]=warp(C4,C5,tempweight);
Template2=(warp1+warp2)/2;
[warp1,warp2,cost]=warp(Template,Template2,tempweight);
Template=(warp1+warp2)/2;

Weight=weight(Template,C1,C2,C3,C4,C5);
Normtemplate=normalise(Template,Weight);
```

A.12 makeweight.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inputs:
%%%%  cepstral (+ Delta) coefficients:
%%%%  - Template
%%%%  - Samples used in template calculation
%%%% output: deviation column vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Weight] = makeweight(Template,C1,C2,C3,C4,C5)

[m,temp]=size(Template);
g=ones(m,1);
Weight=[];

[Warped1,Warped2,cost]=warp(Template,C1,g);
Weight=[Weight weight(Warped1,Warped2)];

[Warped1,Warped2,cost]=warp(Template,C2,g);
Weight=[Weight weight(Warped1,Warped2)];
```

```
[Warped1,Warped2,cost]=warp(Template,C3,g);
Weight=[Weight weight(Warped1,Warped2)];

[Warped1,Warped2,cost]=warp(Template,C4,g);
Weight=[Weight weight(Warped1,Warped2)];

[Warped1,Warped2,cost]=warp(Template,C5,g);
Weight=[Weight weight(Warped1,Warped2)];

Weight=sum(Weight')'./5;
```

A.13 normalise.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inputs:
%%%%  Cepstrum - Cepstrum (& Deltakoeff)
%%%%  Weight - weighting function
%%%% output: Normalized Cepstrum (& Deltakoeff)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function[Norm] = normalise(Cepstrum,Weight)

[m,n]=size(Cepstrum);
for i=1:n
  Cepstrum(:,i)=Cepstrum(:,i).*Weight;
end
Norm=Cepstrum;
```

A.14 preemphasis.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inputs:
%%%%  signal - speech signal row vector
%%%% output: signal preemphasized by 1-0.95/z
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [presig]= preemphasis(signal)

[m,n]=size(signal);
z=1:n;
z=0.95./z;
z=1-z;
presig=abs(ifft(fft(signal).*z)).*sign(signal);
```

A.15 warp.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%inputs:
% order of autocorrelation
% signal1,2 - row vector of feature vectors
%output: warped signals + normalized cost
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Warped1,Warped2,normedCost] =
  warp(signal1,signal2,weight)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Initialisierung

[row1,n] = size(signal1);
[row2,m] = size(signal2);
[row3,temp] =size(weight);
Warp=[];
Warped1=[];
Warped2=[];
normedCost=realmax;

%%%% Sicherheitsabfrage
if (row1~=row2)|(row1~=row3)
else
  signal1isguide=1;
  G=[];
  S=[];
  glength=0;
  slength=0;

    if m>n
      G=signal1;
      S=signal2;
      glength=n;
      slength=m;
    else
      G=signal2;
      S=signal1;
      glength=m;
      slength=n;
      signal1isguide=0;
    end

  mnull=20;
  delta=15;
```

```
  if delta+1 > slength
    startLastAt=slength;
  else
    startLastAt=delta+1;
  end

  %Bereich fuer den Start = 1<=spos<=delta+1;
  Warp=[];
  normedCost=realmax;
  for startSlaveAt=1:startLastAt
     NewPath=warpOnePath(G,S,glength,slength,
                         weight,startSlaveAt,mnull,delta);
     %contraints erfuellt?
     if NewPath(1,1) == glength | NewPath(2,1) == slength
       newCost=NewPath(3,1);
       newCost=newCost*glength/NewPath(2,1);
      if newCost < normedCost
        Warp=NewPath;
        normedCost=newCost;
      end
    end
  end

  [m,n]=size(Warp);
  Warped1=[];
  for i=1:n
    Warped1=[Warped1 G(:,Warp(1,i))];
  end
  Warped2=[];
  for i=1:n
   Warped2=[Warped2 S(:,Warp(2,i))];
  end
  if not(signal1isguide)
    Temp=Warped1;
    Warped1=Warped2;
    Warped2=Temp;
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Pfad fuer einen Startpunkt
%% Eine Position n im Vektor = Position im Guide
%% Ein Wert an der Stelle n = Position im Slave

function[Path] =
  warpOnePath(G,S,glength,slength,weight,startSlaveAt,mnull,delta)

Selected=[[0;0;0],[0;0;0]];
Visited=[Selected];
```

```
Frontier=[];
Neighbors=initNeighbors(startSlaveAt,slength);
Neighbors=addLocalCost(Neighbors,G,S,weight);
Frontier=addToFrontier(Neighbors,Frontier,Selected);
Selected=Frontier(:,1:2);
Frontier(:,1:2)=[];
Visited=Selected;

%Fuer jede Position im Guide die guenstigste
%Slave-Position bestimmen
i=2;
met=1;

while i<=glength & met

  %Neighbors: Guide-Positionen nicht doppelt
  if not(ismember(i,Visited(1,:)))

    %Frontier wird erweitert
    Neighbors=getNeighbors(i,Selected,glength,slength,Visited);
    Neighbors=addLocalCost(Neighbors,G,S,weight);
    Frontier=addToFrontier(Neighbors,Frontier,Selected);

    %Knoten wird gewaehlt
    didselect=0;
    while not(didselect) & size(Frontier) ~=0

    %verbleibende Constraints fuer naechstes Element abfragen
    met=constraints(Frontier(1,1),i,slength,glength,delta,mnull);

      if met
        %Wenn Guide-Position schon drin muss nachgeprueft werden,
        %welcher billiger ist
        if not(ismember(Frontier(1,1),Visited(1,:)))
          Selected=Frontier(:,1:2);
          Visited=[Selected Visited];
          didselect=1;
        else
          found=getVisited(Frontier(1,1),Visited);
          costToFound=cost(Visited(3,found));
          frontierNodeCost=cost([Visited(3,1) Frontier(3,1)]);
          if costToFound > frontierNodeCost
            Selected=Frontier(:,1:2);
            [temp,vlength]=size(Visited);
            Visited=[Selected Visited(:,found:vlength)];
            didselect=1;
            i=Selected(1,1);
          end
        end
```

```
      end
    Frontier(:,1:2)=[];
    end
  didselect=0;
  end
  i=i+1;
end
Path=getPath(Visited);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Node] = getNode(Selected)
Node = Selected(:,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Pred] = getPred(Selected)
Pred = Selected(:,2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [met] =
  constraints(spos,gpos,slength,glength,delta,mnull)

met=1;

% Bedingung fuer die Slaveposition am Endpunkt
if gpos == glength
  met = met & slength-delta<=spos<=slength;
end

% Bedingungen fuer den Endpunkt
met = met & spos<slength;
met = met & glength-delta<=spos<=glength;

% Bedingung fuer den Verlauf
temp=(slength/glength)*spos;
met = met & temp-mnull<=spos<=temp+mnull;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [c] = localCost(X,Y,weight)
[m,n]=size(X);
temp=[];
for i=1:m
  temp=[temp (X(i,1)-Y(i,1))^2*weight(i,1)^2];
end
c=sum(temp);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [c] = cost(Path)
c=sum(Path);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% input:
%% i: Guide-Position
%% Selected: aktueller Knoten
%% slength: Zeilenlaenge vom Slave
%% Visited: Bereits besuchte Knoten
%% output:
%% - 1. Zeile: Guide-Position
%% - 2. Zeile: Slave-Position

function [Neighbors] = initNeighbors(startSlaveAt,slength)

Neighbors=[];
j=startSlaveAt;

Neighbors=[1;j];

if j+1 <= slength
  Neighbors = [Neighbors  [1;(j+1)]];
  if j+2 <= slength
    Neighbors = [Neighbors  [1;(j+2)]];
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% input:
%% i: Guide-Position
%% Selected: aktueller Knoten
%% slength: Zeilenlaenge vom Slave
%% Visited: Bereits besuchte Knoten
%% output:
%% - 1. Zeile: Guide-Position
%% - 2. Zeile: Slave-Position

function [Neighbors] =
  getNeighbors(i,Selected,glength,slength,Visited)
[m,n]=size(Visited);
k=-1;
if n>=3
    k=Visited(:,3);
end
temp=getNode(Selected);
j=temp(2,1);

%fuer die Initialisierung
if j==0
  j=i;
end
```

```
Neighbors=[];

% Steigung zwischen 0.5 und 2
if k~=j
  Neighbors=[i;j];
end
if j+1 <= slength
  Neighbors = [Neighbors  [i;(j+1)]];
  if j+2 <= slength
    Neighbors = [Neighbors  [i;(j+2)]];
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% input:
%% - Neighbors: 1. Zeile: Guide-Position
%%              2. Zeile: Slave-Position
%% - G: Guide function
%% - S: Slave function
%% output:
%% - 1. Zeile: Guide-Position
%% - 2. Zeile: Slave-Position
%% - 3. Zeile: Kosten

function [Neighbors] = addLocalCost(Neighbors,G,S,weight)
[m,length]=size(Neighbors);
for i=1:length
  g=G(:,Neighbors(1,i));
  s=S(:,Neighbors(2,i));
  Neighbors(3,i)=localCost(g,s,weight);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% output: Paare von [node predecessor] aufsteigend sortiert
function [Frontier] =
  addToFrontier(Neighbors,Frontier,Selected)

[m,n]=size(Neighbors);
Nodes=[];
for i=1:n
  temp=getNode(Selected);
  Neighbors(3,i)=cost([Neighbors(3,i) temp(3,1)]);
  Nodes = [Nodes [Neighbors(:,i) temp]];
end
  Frontier = [Frontier Nodes];
  Frontier = sortByCost(Frontier);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% output: Priority Queue

function [Frontier] = sortByCost(Frontier)
[m,length]=size(Frontier);

for j=1:2:length-2;
  for i=1:2:length-2;
    if Frontier(3,i) > Frontier(3,i+2)
      temp=Frontier(:,i);
      temp2=Frontier(:,i+1);
      Frontier(:,i)=Frontier(:,(i+2));
      Frontier(:,i+1)=Frontier(:,i+3);
      Frontier(:,(i+2))=temp;
      Frontier(:,i+3)=temp2;
    end
  end
  length=length-2;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Found] = getVisited(Findme,Visited)
Found=0;
[m,length]=size(Visited);
i=1;
found=0;
while i<length & not(found)
  found=Findme==Visited(1,i);
  Found=i;
  i=i+2;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Path] = getPath(Visited)
Path=[];
[m,length]=size(Visited);
for i=1:2:length-1
  Path=[Path Visited(:,i)];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

A.16 weight.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inputs:
%%%%  cepstral (& delta) coefficients
%%%% output: deviation for 2 utterances column vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [weight] = weight(Warped1,Warped2)
[m,n]=size(Warped1);
Comp1=[];
Comp2=[];
Weight=[];
weight=[];

for j=1:m

  Comp1=Warped1(j,:);
  Comp2=Warped2(j,:);

  Tempcost=[];
  for i=1:n
    Tempcost=localCost(Comp1(1,i),Comp2(1,i));
    Weight=[Weight sum(Tempcost)];
  end
  weight=[weight;sum(Weight)];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [c] = localCost(X,Y)
     [m,n]=size(X);
temp=[];
for i=1:m
  temp=[temp;(X(i,1)-Y(i,1))^2];
end
c=sum(temp);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```