Karlsruhe Institute of Technology

Kontextsensitive Konvertierung von geflüsterter auf hörbare Sprache: Implementierung und Evaluation

Bachelorarbeit am Cognitive Systems Lab
Prof. Dr.-Ing. Tanja Schultz
Fakultät für Informatik
Karlsruher Institut für Technologie

von

Maximilian Vogel

Betreuer:
Prof. Dr.-Ing. Tanja Schultz
Dipl. Math. Michael Wand

Tag der Anmeldung: 1. März 2013
Tag der Abgabe: 30. Juni 2013

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 30. Juni 2013.

Zusammenfassung

Die Konvertierung von geflüsterter in hörbare Sprache kann in verschiedenen Szenarien Anwendung finden: Das Telefonieren in der Öffentlichkeit kann angenehmer gemacht werden. Viel interessanter und nützlicher ist es, denjenigen Menschen, die aufgrund verschiedener Umstände nur noch in der Lage sind, zu flüstern, wieder die Möglichkeit zu geben, sich normal artikulieren zu können. Das bestehende System am CSL basiert auf gemeinsamen Gaußmischmodellen (GMMs) und die Konvertierungsfunktion minimiert den quadratischen Fehler. Da die ausschnittsweise Konvertierung den Kontext, in welchem ein Ausschnitt geäußert wurde, nicht berücksichtigt, entstehen Artefakte, die das Ergebnis oft abgehackt oder nicht zusammenpassend klingen lassen. Im Rahmen dieser Arbeit wurde daher das bestehende System um ein Verfahren erweitert, welches Informationen über den Kontext, die benachbarten Ausschnitte, im Konvertierungsprozess berücksichtigt. Anschließend wurde das Verfahren getestet und unter anderem mit einem Hörtest evaluiert.

Inhaltsverzeichnis

1. Einleitung

In dieser Arbeit wird die Konvertierung verschiedener Artikulationsarten behandelt, unter hörbarer bzw. normal geäußerter Sprache versteht man dabei den Klang der Sprache, wenn man sich normal in einer geräuschfreien Umgebung miteinander unterhält. Die Anwendung von Konvertierung von geflüsterter zu hörbarer Sprache kann folgendermaßen beschrieben werden: Der Benutzer flüstert in ein Mikrofon und mit möglichst geringer Verzögerung hört man das Gesagte in normal geäußerter Sprache. Diese Anwendung ist vor allem für diejenigen Menschen hilfreich, die aus verschiedenen Gründen nur noch in der Lage sind, zu flüstern, denn diesen könnte man mithilfe eines tragbaren Geräts die Artikulation normal geäußerter Sprache wieder ermöglichen. Damit sind vor allem Patienten gemeint, die eine Krebserkrankung im Kehlkopf- oder Rachenbereich haben und aufgrund eines operativen Eingriffs nicht mehr in der Lage sind, normal zu sprechen. Für solche Fälle gibt es bereits einige Lösungen, allerdings sind diese mit hohem Aufwand verbunden oder das Ergebnis ist nicht zufriedenstellend. Ein weiterer Einsatzbereich sind öffentliche Transportmittel, hier kann das Telefonieren sowohl für die telefonierende Person als auch für die Fahrgäste in der unmittelbaren Umgebung annehmlicher gemacht werden.
Am Cognitive Science Lab (CSL) von Prof. Tanja Schultz werden bereits Prototypen in ähnlichen Bereichen entwickelt, so zum Beispiel die Konvertierung von EMG-Signalen in Fundamentalfrequenz[NJWS11] und Sprache [WSJS13, JWNS12]. Die Methoden zur Konvertierung folgen dem aktuellen Stand der Forschung [KM98, SCM98] und verwenden bei der Konvertierung die Parameter von zuvor trainierten, gemeinsamen Gaußmischmodellen (GMM). Diese Methode kann unabhängig von der konkreten Anwendung verwendet werden, solange die Quell- und Zieldaten miteinander korrelieren. Am CSL wird mit dieser Basis auch die Konvertierung von geflüsterter zu hörbarer Sprache realisiert und experimentiert, ob dieser Ansatz zukunftsfähig ist.
Die Konvertierung ist dabei nur ein Teilschritt, um das gesamte Verfahren zu rea-

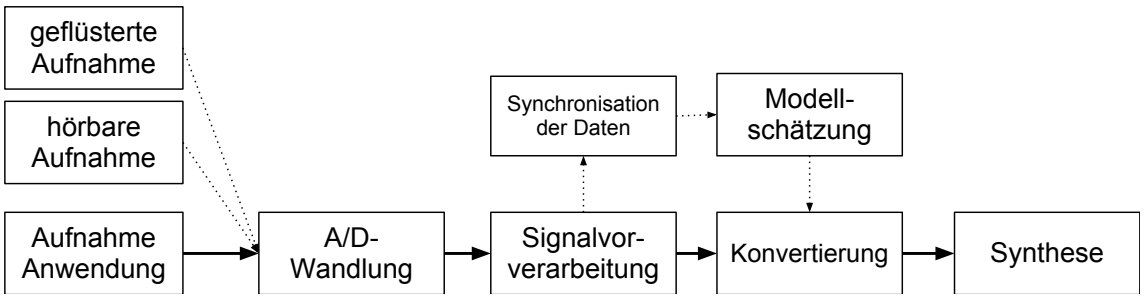

Abbildung 1.1: Verarbeitungskette bei der Stimmenkonvertierung

lisieren, wie in Abbildung 1.1 illustriert. Um das angesprochene GMM schätzen zu können, benötigt man synchronisierte, für die Anwendung vorverarbeitete Trainingsdaten. Das heißt, dass Aufnahmen derselben Sätze in beiden Artikulationsarten gemacht werden müssen. Diese werden digitalisiert und anschließend Merkmale extrahiert, die die Sprache charakterisieren. Einander entsprechende Merkmale der geflüsterten und hörbaren Sprache werden bei der Synchronisation gefunden. Ist das

Modell trainiert, werden die Parameter des Modells verwendet, um Aufnahmen in geflüsterter Sprache in hörbare zu konvertieren. Schließlich muss das Ergebnis der Konvertierung noch in ein akustisches Format umgewandelt werden.

1.1 Zielsetzung der Arbeit

In dieser Arbeit wird ein Verfahren implementiert und evaluiert, das bei der Konvertierung Informationen über den Kontext, aus dem Ausschnitt herausgenommen wird, berücksichtigt. Davon wird erwartet, dass die konvertierten Ausschnitte besser zueinander passen und die synthetisierte Audiodatei weniger abgehackt klingt. Letztlich wird eine Steigerung der Qualität vor allem hinsichtlich Verständlichkeit und Natürlichkeit angestrebt.

1.2 Gliederung der Arbeit

In Kapitel 2 werden die mathematischen Grundlagen und damit die theoretische Basis für die Anwendung eingeführt. Es wird auf einzelne Komponenten aus Abbildung 1.1 eingegangen und einige dafür notwendige Komponenten und Algorithmen werden aufgeführt und erläutert, so auch das Verfahren, das im Rahmen dieser Arbeit implementiert und getestet wird. Anschließend wird mit Kapitel 3 ein Überblick über die aktuelle Forschung in diesem Bereich gegeben. Das Framework BioKIT, das am CSL entwickelt und mit dem geforscht wird, wird in Kapitel 4 vorgestellt und auf die für die Implementierung relevanten Schnittstellen, Klassen und Funktionen eingegangen. Im folgenden Kapitel 5 wird diese dann beschrieben und erläutert, worauf geachtet werden musste. Kapitel 6 enthält die Experimentbeschreibung sowie die Ergebnisse, die kritisch diskutiert werden. Schließlich werden die Erkenntnisse der Arbeit in Kapitel 7 zusammengefasst und ein Ausblick gegeben.

2. Grundlagen

In diesem Kapitel werden einige Verfahren für die Bestandteile Vorverarbeitung, Synchronisation, Modellschätzung und Konvertierung aus Abbildung 1.1 eingeführt. Ein Verständnis von der Entstehung von Sprache ist essenziell, um die Anwendung zu realisieren und motiviert die Vorverarbeitung, welche lediglich kurz skizziert wird. Bei den Abschnitten 2.5 und 2.6 handelt es sich um Verfahren zur Konvertierung einer Aufnahme bzw. Sequenz x aus dem Quellbereich in den Zielbereich, wobei das Verfahren aus Abschnitt 2.5 bereits am CSL implementiert ist und es sich bei dem Verfahren aus Abschnitt 2.6 um das zu implementierende und zu testende handelt.

2.1 Sprachproduktion der menschlichen Stimme

Ursprung der menschlichen Stimme sind die Stimmbänder beziehungsweise die Glottis (siehe „Larynx“, englisch für Kehlkopf in Abbildung 2.1). Strömt Luft aus der Lunge, so werden die Stimmbänder in Vibration versetzt und es entsteht der Grundton. Dieser kann durch den Zustand der umliegenden Muskeln beeinflusst werden: Ist die Glottis kleiner und angespannter, vibriert sie schneller, was in einem höheren Ton resultiert. Tiefere Tönen entstehen, wenn die Muskeln schlaff sind und die Stimmbänder viel Raum zum Schwingen haben. Diesen Ton nennt man Fundamentalfrequenz, oft mit F0 abgekürzt. Im Vokaltrakt, dem Kiefer-, Mund- („oral

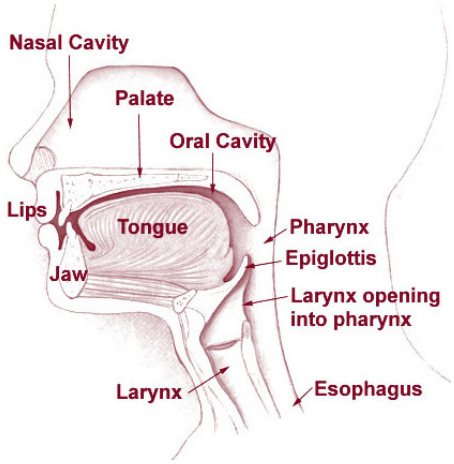

Abbildung 2.1: Schematische Abbildung des Vokaltrakts; freies Bild von Wikimedia Commons[wik]

cavity“) und Nasenbereich („nasal cavity“), wird der Grundton durch Bewegung und Form der Lippen („lips“), der Zunge („tonge“) und des Kiefers („ jaw“) zu Sprech- und Singlauten verändert. Als Formanten versteht man Frequenzbereiche, die im Spektrum stark ausgeprägt sind und über die Formanten F1 und F2 lassen sich die Vokale charakterisieren.

Diese Entstehung der Sprache wird auch von dem aus der Theorie der Signalverarbeitung stammenden Source-Filter-Modell für Sprache (vgl. Abb. 2.2) übernommen: Dabei interpretiert man den Vokaltrakt als Filter, welcher auf einem Signal angewandt wird und somit Sprache erzeugt. Bei stimmhafter Sprache ist die Quelle des Signals die Fundamentalfrequenz, wohingegen bei stimmloser Sprache weißes

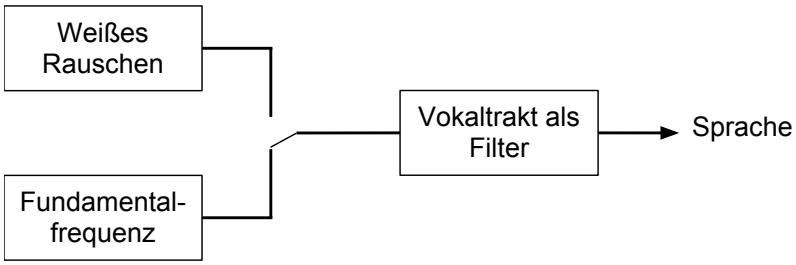

Abbildung 2.2: Source-Filter-Model

Rauschen als Quelle angenommen wird. Geflüsterte Sprache unterscheidet sich von hörbarer Sprache vor allem darin, dass bei geflüsterter Sprache keine Fundamentalfrequenz vorhanden ist. Des Weiteren verschieben sich die Formanten, detailliertere Informationen finden sich in [SMA10].

2.2 Vorverarbeitung

In der Signalvorverarbeitung werden die digitalen Rohdaten im Hinblick auf die Anwendung aufbereitet. Für unsere Anwendung sind folgende Kriterien wichtig:

- Die Extraktion von Merkmalen, welche die Sprache charakterisieren,
- die (Rück-)Umwandlung in ein akustisches Signal soll möglich sein,
- ein Zusammenhang zwischen den verschiedenen Artikulationsarten und
- die Verringerung der Dimensionalität beziehungsweise
- ein Kompromiss zwischen Rechenaufwand und Qualität.

Die Vorverarbeitung für die vorgestellte Anwendung teilt sich auf in die Abschätzung der Fundamentalfrequenz F0 und die Berechnung der „warped cepstra“-Merkmale bzw. Features, kurz WCEPs. Grundlegend für die Vorverarbeitung ist das Fenstern der Daten, damit ist das Aufteilen einer Aufnahme in Ausschnitte, auch „Frames“ genannt, gemeint. Diese Lokalisierung ist notwendig, um die Eigenschaften von Sprache zu einem bestimmten Zeitpunkt innerhalb einer Aufnahme zu extrahieren. Das Fenstern der Daten wird auch überlappend durchgeführt, um keine Übergänge zu verlieren. 16 ms als Länge eines Frames und eine Verschiebung von 10 ms haben sich als praktikable Parameter für Sprachanwendungen herausgestellt. Diese Dauer ist hinreichend kurz, um die Dynamik in der Sprache zu erfassen. Eine Aufnahme x besteht dann aus der Konkatenation der T Frames: $x = x_1 x_2 \dots x_T$.

2.2.1 Fundamentalfrequenz

Eine gängige Methode, um aus einem Signal die Fundamentalfrequenz F0 zu extrahieren, ist die Autokorrelation, welche für die Signalverarbeitung leicht modifiziert wird:

$$r_t(\tau) = \sum_{j=t+1}^{t+W-\tau} x_j x_{j+\tau}.$$

Die Funktion vergleicht das Signal x_t mit der um τ verschobenen Folge, wobei W die Breite des Kontexts angibt und trivialerweise verfügt $r_t(0)$ über den höchsten

Wert. Bei der Autokorrelation treten aber in Abhängigkeit des Parameters τ einige Fehler auf. A. de Cheveigné and H. Kawahara haben dieses Verfahren evaluiert, die Fehlertypen analysiert und schrittweise Optimierungen eingeführt, die die verschiedenen Fehler der Autokorrelation reduziert. Ihre Erkenntnisse haben sie in [DCK02] festgehalten.

2.2.2 „warped cepstrum“-Merkmale (WCEP)

In der Sprachverarbeitung haben sich Merkmale bewährt, die aus dem „short time spectrum“, dem Spektrum eines Ausschnitts x_t extrahiert werden. Das Power-Spektrum erhält man, indem man den Betrag des Ergebnis der Fouriertransformation über das Eingabesignal quadriert: $|\mathcal{F}\{x(t)\}|^2$. Man kann dann über die Mel- oder Bark-Skala den Frequenzbereich der Wahrnehmung des menschlichen Gehörs anpassen und auf eine vorgegebene Anzahl Koeffizienten reduzieren. Zu einem Zeitpunkt t wird das akustische Signal von den Koeffizienten repräsentiert und wir bezeichnen den Ausschnitt x_t auch als Merkmalvektor.
Eine verfeinerte Form der Vorverarbeitung ist das „Cepstrum“, das definiert ist als die inverse Fouriertransformation des Logarithmus des Spektrums:

$$|\mathcal{F}^{-1}\{\log|\mathcal{F}\{x(t)\}|^2\}|^2$$

Die Extraktion der Merkmale aus dem Cepstrum ist im Bereich der Sprachanwendungen weit verbreitet. Ein Algorithmus zur effizienten Berechnung dieser Merkmale ist mit „An adaptive algorithm for mel-cepstral analysis of speech[FTKI92]“ von Fokuda et al gegeben. Dieser Algorithmus lässt sich wie folgt skizzieren:

1. 70hz Hochpassfilter über das Eingabesignal,
2. Einteilen in Frames bzw. Ausschnitte der Länge 10ms mittels Blackman-Fenster,
3. Power-Spektrum berechnen,
4. Umwandlung in Cepstrum durch Logarithmieren und Anwenden der inversen Fouriertransformation,
5. Frequenzen auf Mel-Skala verformen und auf eine vorgegebene Anzahl Koeffizienten reduzieren,
6. Optional: iterative Optimierung.

2.3 Synchronisation: Dynamic Time Warping

Für die Schätzung eines gemeinsamen Modells ist es essenziell, dass man weiß, welche Merkmalvektoren der geflüsterten Sprache denen in hörbarer Sprache entsprechen. Sprache wird allerdings nicht immer exakt identisch artikuliert, weder die Dauer noch die Frequenzen der einzelnen Wortteile. Eine naive Stapelung der Merkmalvektoren x_t und y_t ist daher sehr fehleranfällig. Der „Dynamic Time Warping“-Algorithmus dient ursprünglich zur Berechnung der „Kosten“, die notwendig sind, um mithilfe der Operationen Einfügen, Löschen und Ersetzen von einer Hypothese zu einer Referenz zu gelangen. Er wird in Sprachanwendungen jedoch oft dafür eingesetzt, einander entsprechende Merkmalvektorenpaare x_t und y_t zu finden. Dazu wird die euklidische Distanz der Merkmalvektoren herangezogen und innerhalb eines Fensters um den

Zeitpunkt t nach dem Paar mit dem geringsten Abstand gesucht. Dieser Algorithmus sollte für die verschiedenen Artikulationsarten verwertbare Ergebnisse liefern, da die Fundamentalfrequenz in den Merkmalen nicht enthalten ist und sich sonst lediglich die Formanten verschieben.

2.4 Modelschätzung: Gaußmischmodelle

2.4.1 Normalverteilung

Gaußmischmodelle basieren auf der Gauß'schen Normalverteilung beziehungsweise ihrer Wahrscheinlichkeitsdichtefunktion, die im D-Dimensionalen Fall wie folgt definiert ist:

$$\mathcal{N}(x;\mu;\Sigma) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right], \tag{2.1}$$

Eine Normalverteilung wird durch ihre Parameter, den D-dimensionalen Mittelwertsvektor μ und die $D \times D$-dimensionale Kovarianzmatrix Σ, beschrieben. Wir bezeichnen die Auswertung der Normalverteilung an Stelle x mit $\mathcal{N}(x;\mu;\Sigma)$ und modelltheoretisch handelt es sich um die a-posteriori Wahrscheinlichkeit $P(x)$, dass x aus der Normalverteilung stammt.

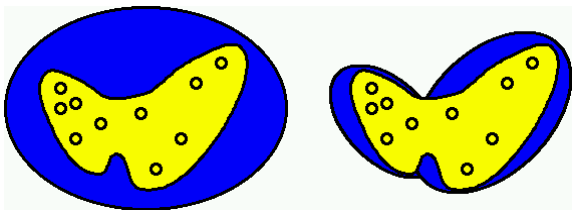

Abbildung 2.3: Gaußmischmodelle mit mehreren Verteilungen sind präziser. Links wurde nur eine Normalverteilung verwendet und Punkte, die den blauen Bereich treffen und den gelben nicht, würden fälschlicherweise als der Menge zugehörig eingestuft werden. Die Form der gelben Verteilung wird durch zwei Normalverteilungen im rechten Teil deutlich präziser nachgebildet.

2.4.2 Gaußmischmodelle

Die Daten der Sprache, die wir durch die Vorverarbeitung erhalten, sind nicht normalverteilt, daher ist es sinnvoll, wie in Abbildung 2.3 dargestellt, mehrere Gaußkomponenten zu verwenden, um die Genauigkeit des Modells zu steigern. Ein Mischmodell λ bestehe aus M Gaußkomponenten. Die a-posteriori Wahrscheinlichkeit $P(x)$ erhält man dann durch die Auswertung der M Gaußkomponenten multipliziert mit dem Gewicht von $\mathcal{N}_m$:

$$P(x|\lambda) = \sum_{m=1}^{M} w_m \mathcal{N}(x;\mu_m;\Sigma_m) \tag{2.2}$$

Für die angestrebte Anwendung wird ein gemeinsames Modell formuliert, um einen Zusammenhang zwischen Quelle und Ziel festzuhalten. Dazu werden zuvor synchronisierte Frames x_t und y_t zu einem gemeinsamen Frame $z_t = \begin{bmatrix} x_t \\ y_t \end{bmatrix}$ gestapelt. Die Parameter einer beliebigen Verteilung $\mathcal{N}_m$ lassen sich dann partitioniert in x und y betrachten $\mu_m^{(z)} = \begin{bmatrix} \mu_m^{(x)} \\ \mu_m^{(y)} \end{bmatrix}$, $\Sigma_m^{(z)} = \begin{pmatrix} \Sigma_m^{(xx)} & \Sigma_m^{(xy)} \\ \Sigma_m^{(yx)} & \Sigma_m^{(yy)} \end{pmatrix}$.

2.4.3 Maximum-Likelihood-Schätzmethode

Unter der Annahme, dass die aufgenommenen und gestapelten Trainingsdaten z_t die Stimme beziehungsweise Sprache einer Person akkurat repräsentieren, suchen wir nun ein Gaußmischmodell λ, das seinerseits die Trainingsdaten präzise abbildet. Eine gängige Methode ist es, die Parameter des Modells mithilfe der Trainingsdaten so abzuschätzen, dass die Trainingsdaten eine hohe Wahrscheinlichkeit auf dem Modell haben. Das lässt sich durch folgenden Ausdruck fassen:

$$\hat{\lambda} = \arg\max_{\lambda} \sum_{n=1}^{N} \log[P(z_n|\lambda)].$$

Daraus resultiert eine zu optimierende Zielfunktion $Q = \sum_{n=1}^{N} \log[P(z_n|\lambda)]$. Der „Maximum Likelikhood(ML)“-Schätzer bestehend aus den Parametern μ und Σ, der die Zielfunktion Q maximiert, lässt sich über die Extremstellen der Funktion bestimmen. Dazu wird die Ableitung der Zielfunktion nach μ null gesetzt: Der von μ unabhängige fällt weg $\frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}}$ weg und die Exponentialfunktion und der Logarithmus heben sich auf, sodass dann gilt:

$$0 = \Sigma^{-1}(z_1 - \mu) + \Sigma^{-1}(z_2 - \mu) + \cdots + \Sigma^{-1}(z_N - \mu) \tag{2.3}$$

$$0 = \Sigma^{-1} \underbrace{(\sum_{n=1}^{N} z_n - N * \mu)}_{H} \tag{2.4}$$

Hieraus folgt, wenn wir davon ausgehen, dass die Kovarianzmatrix Σ regulär ist:

$$0 = \sum_{n=1}^{N} z_n - N * \mu \tag{2.5}$$

$$\mu = \frac{1}{N} \sum_{n=1}^{N} z_n. \tag{2.6}$$

Diese Herleitung kann man auch für die Kovarianz durchführen, sodass der ML-Schätzer der Normalverteilung durch

$$\mu = \frac{1}{N} \sum_{n=1}^{N} z_n \tag{2.7}$$

$$\Sigma = \frac{1}{N} \sum_{n=1}^{N} (z_n - \mu)(z_n - \mu)^T \tag{2.8}$$

gegeben ist. Wir verwenden allerdings Gaußmischmodelle und müssen nicht nur einen Satz Parameter bestehend aus μ und Σ, sondern M Sätze abschätzen.
Um die Zustände in einem Gaußmischmodell mathematisch zu beschreiben, führen wir die Zuordnungswahrscheinlichkeit $\gamma_{m,k}$ ein. Sie beschreibt, mit welcher Wahrscheinlichkeit ein Vektor z_k aus der Normalverteilung $\mathcal{N}_m$ stammt und ist definiert als die a-posteriori Wahrscheinlichkeit von z_k unter $\mathcal{N}_m$ normiert durch die a-posteriori Wahrscheinlichkeit von z_k unter λ:

$$\gamma_{m,k} = \frac{P(z_k|m)}{P(z_k|\lambda)} = \frac{\mathcal{N}(z_k; \mu_m; \Sigma_m)}{\sum_{m=1}^{M} \mathcal{N}(z_k; \mu_m; \Sigma_m)}. \tag{2.9}$$

Legt man nun wieder die Zielfunktion des ML-Schätzers als Kriterium an, stellt man fest, dass der Wert der Zielfunktion Q von den konkreten Parametern μ_m und Σ_m der M Normalverteilungen abhängt. Wie man bei der oben hergeleiteten Abschätzung sehen kann, werden die Parameter jedoch durch die Trainingsvektoren z_n bestimmt, welche nun nicht mehr absolut zugeordnet werden. Eine analytische Lösung dieses Optimierungsproblems bei dem zwei Kriterien optimiert werden sollen, ist nicht bekannt.
Eine Möglichkeit, dieses Problem zu lösen, ist der „Expectation-Maximization"-Algorithmus, welcher bereits 1977 von Dempster et al [DLR77] formuliert wurde. Es handelt sich dabei um einen iterativen Algorithmus, der abwechselnd die Schritte Expectation (= Abschätzen) und Maximization (= Maximierung) durchführt.

Expectation Jedes Element der Trainingsdaten wird den einzelnen Verteilungen probabilistisch zugeordnet (vgl. $\gamma_{m,k}$).

Maximization Die Parameter der Verteilungen werden mithilfe der Zuordnungen aus dem Expectationschritt aktualisiert und dadurch der Wert der Zielfunktion verbessert.

Dieser Algorithmus lässt sich konzeptionell für GMMe in Pseudocode formulieren und wird der Übersichtlichkeit halber mit Hilfsfunktionen beschrieben. Die Hilfsfunktionen sind dabei dem Maximization-Schritt zuzuordnen wohingegen die vorangegangenen verschachtelten for-Schleifen den Expectation-Schritt realisieren. Neben den Parametern μ_m und Σ_m werden zusätzlich Gewichte w_m für die M Gaußkomponenten berechnet. Das Gewicht einer Gaußkomponente $\mathcal{N}_m$ gibt uns an, wie viele Datenpunkte auf diese Komponente fallen. Der Wert ist allerdings keine ganze Zahl, da die Zuordnungen $\gamma_{m,k}$ probabilistisch sind. Damit die Gewichte auch in Relation zueinander stehen, werden sie durch die Gesamtwahrscheinlichkeit des Modells λ normiert: $w_m = \frac{P(m)}{P(\lambda)} = \frac{\sum_{k=1}^{N} \gamma m,k}{\sum_{k=1}^{N} \sum_{m=1}^{M} \gamma m,k}$. Als Abbruchkritierium kann man eine Anzahl Iterationen festsetzen oder es wird abgebrochen, wenn der Betrag der Differenz der ausgewerteten Zielfunktion zwischen zwei Iterationen einen festgelegten Grenzwert *threshold* unterschreitet: $|Q^{(i)} - Q^{(i-1)}| < threshold$.

```
function EMTraining(K, datasamples)
    initialisiere K Mittelwertsvektoren beliebig
    i = 1
    repeat
        for all n = 1,...,N do
            for all m in M do
                likelihood = N(z_n; μ_m, Σ_m)   // expectation
                γ_n[m] = likelihood             // expectation
                sum += likelihood               // expectation
            end for
            γ /= sum                            // expectation
            sum = 0
        end for
        calculateWeights()                      // maximization
        calculateMeans()                        // maximization
        calculateCovariance()                   // maximization
        i++
```

```
    until i = maxIteration or |Q^(i) − Q^(i−1)| < threshold
    return [w, μ, Σ]
end function
function calculateWeights( )
    sumOfWeights = Σ_{n=1}^{N} Σ_{m=1}^{M} γ_n[m]
    for all m in M do
        w_m = (Σ_{n=1}^{N} γ_n[m])/ sumOfWeights
    end for
end function
function calculateMeans( )
    for all m in M do
        for all n = 1,...,N do
            μ_m += γ_n[m]*z_n
            weightOfSamples += γ_n[m]
        end for
        μ_m /= weightOfSamples
    end for
end function
function calculateCovariances( )
    for all m in M do
        for all n = 1,...,N do
            Σ_m += γ_n[m] * (z_n − μ_m)(z_n − μ_m)^T
            weightOfSamples += γ_n[m]
        end for
        Σ_m /= weightOfSamples
    end for
end function
```

Als Vorteile des Algorithmus sind anzuführen, dass er das beste GMM für eine gegebene Menge Trainingsdaten und eine gegebene Anzahl Gaußkomponenten findet. Allerdings ist es ein langsamer Algorithmus. Es ist daher üblich, den kMeans-Algorithmus zu verwenden, um eine Ausgangslösung zu erhalten, welches dann durch das inkrementell EM-Training verbessert wird.
Der kMeans-Algorithmus kann ebenfalls als Vertreter der EM-Algorithmen angesehen werden, allerdings werden die Datenpunkte den Verteilungen nicht probabilistisch zugeordnet, sondern absolut. Der Algorithmus berechnet für die übergebenen Trainingsdaten K Cluster, deren Zentren als Mittelwertsvektoren für das EM-Training verwendet werden können. Der Algorithmus beginnt mit der Initialisierung von K Mittelwertsvektoren. Dies kann per Algorithmus oder vollkommen beliebig geschehen, alternativ werden (zufällig) Vektoren aus den Trainingsdaten ausgewählt. Dann wird jeder Vektor aus den Trainingsdaten dem Zentrum zugeordnet, zu dem der euklidische Abstand am geringsten ist: $m = \arg\min_{k \in K} ||z_n - \mu^{(k)}||$. Anschließend werden die neuen Mittelwertsvektoren und Gewichte über die zugeordneten Vektoren bestimmt. Die Anzahl der Mittelwertsvektoren bzw. Verteilungen wird dem Algorithmus mit dem Parameter k übergeben, daher auch der Name kMeans. Folgende Abbruchkriterien können verwendet werden:

- Eine vorgegebene Anzahl Iterationen ist erreicht.
- Die neu berechneten Parameter ändern sich nicht oder nur geringfügig.

- Der Fehlerwert unterschreitet einen vorgegebenen Grenzwert: $err_{datasamples} = \sum_{k=1}^{K} \sum_{n=1}^{N_k} ||z_n^{(k)} - \mu^{(k)}||^2 < \text{threshold}$

function kMEANS(K, datasamples)
  initialisiere K Mittelwertsvektoren beliebig
  i = 0
  repeat
    for all n = 1,...,N **do**
      m = $\arg\min_{k \in K} ||z_n - \mu^{(k)}||$
      $\mu_{akku}^{(m)}$ += z_n // expectation
      $w_{akku}^{(m)}$ += 1 // expectation
    end for
    for k $\in K$ **do**
      $w^{(k)} = \frac{w_{akku}^{(k)}}{|\text{datasamples}|}$ // maximization
      $\mu^{(k)} = \frac{1}{w_{akku}^{(k)}} \mu_{akku}^{(k)}$ // maximization
    end for
    i++
  until i = maxIteration **or** $err_{samples} < \text{threshold}$ **or** $\mu = \mu_{akku}$
  return $[w, \mu]$
end function

Durch die fehlende Abschätzung der Kovarianzmatrizen und durch die absolute Zuordnung der Datenpunkte ist die Dauer einer Iteration des kMeans-Algorithmus deutlich kürzer als die des EM-Trainings. Außerdem konvergiert er schneller. Problematisch ist jedoch, dass der Algorithmus in einem lokalen Optimum verharrt, sollte er ein solches finden. Durch die nachträgliche Ausführung des eigentlichen EM-Trainings stellt dies kein Problem dar.

2.5 Minimierung des quadratischen Fehlers des Erwartungswerts (mmse)

Alexander Kain und Michael Macon beschreiben in [KM98], wie man das gemeinsame Mischmodell verwenden kann, um eine Konvertierung der Merkmale aus dem Quell- in den Zielbereich zu realisieren. Ihr Ziel war es, den quadratischen Fehler des Erwartungswertes, zu minimieren:

$$\epsilon_{mse} = E[||y - f(x)||^2]$$

Die Konvertierungsfunktion ist die Regression, die durch den bedingten Erwartungswert von y unter x gegeben ist:

$$E[y|x, \lambda^{(z)}] = \int \mathrm{d}y \; y \; P(y|x) = \sum_{m=1}^{M} P(m|x_t, \lambda^{(z)})[\mu_m^{(y)} + \Sigma_m^{(yx)} {\Sigma_m^{(xx)}}^{-1} (x_t - \mu_m^{(x)})]$$

wobei

$$P(m|x, \lambda^{(z)}) = \frac{P(m|x)}{P(x|\lambda^{(z)})} = \frac{w_m \mathcal{N}(x; \mu_m^{(x)}; \Sigma_m^{(xx)})}{\sum_{k=1}^{K} w_k \mathcal{N}(x; \mu_k^{(x)}; \Sigma_k^{(xx)})}$$

den Zuordnungswahrscheinlichkeiten allerdings lediglich im Quellbereich entspricht. Für einzelne Normalverteilungen $\mathcal{N}_m$ kann für einen bekannten Vektor x_t ein bedingte Wahrscheinlichtsdichtefunktion für einen beliebigen Vektor y_t beschrieben werden:

$$P(y_t|x_t, m, \lambda^{(z)}) = \mathcal{N}(y; E_{m,t}^{(y)}; D_m^{(y)}) \tag{2.10}$$

wobei

$$E_{m,t}^{(y)} = \mu_m^{(y)} + \Sigma_m^{(yx)} {\Sigma_m^{(xx)}}^{-1} (x - \mu_m^{(x)}) \tag{2.11}$$

dem oben genannten bedignten Erwartungswert entspricht und

$$D_m^{(y)} = \Sigma_m^{(yy)} - \Sigma_m^{(yx)} {\Sigma_m^{(xx)}}^{-1} \Sigma_m^{(xy)} \tag{2.12}$$

die Kovarianz zwischen Quell- und Zielbereich modelliert. Einen konvertierten Vektor $\overline{y}_t$ erhalten wir also wie folgt:

$$\overline{y}_t = \sum_{m=1}^{M} P(m|x, \lambda^{(z)}) E_{m,t}^{(y)} \tag{2.13}$$

Die Wahrscheinlichkeit eines konvertierten Vektors unter dem Modell $\lambda^{(z)}$ lässt sich durch

$$P(y_t|x_t, \lambda^{(z)}) = \sum_{m=1}^{M} P(m|x_t, \lambda^{(z)}) P(y_t|x_t, m, \lambda^{(z)}) \tag{2.14}$$

ermitteln.

2.6 Glätten der Trajektorien (mlpg)

Gaußmischmodelle wie sie bisher eingeführt und verwendet wurden, verfügen über keine Informationen über den Kontext, in dem die einzelnen Frames x_t und y_t geäußert wurden. Mit Kontext sind die benachbarten Frames gemeint: x_{t-1} und x_{t-1} für x_t. Nach [HG12, HSVG12] und [TBT07] entsteht ein schlechtes Syntheseergebnis vor allem durch den fehlenden Kontext, da die einzelnen Ergebnisse der Konvertierung nicht zusammenpassen. Getreu dem Titel der Arbeit wird nun ein Verfahren vorgestellt, welches geeignete Kontextinformationen verwendet, um den Einzelergebnissen der Umwandlung einen Kontext zu verschaffen. Es handelt sich dabei um den Abschnitt III.A „Conversion considering dynamic features“ aus [TBT07] von Toda et al. Zuerst wird beschrieben, wie man geeignete Kontextinformationen erhält. Danach wird das Gaußmischmodell neu formuliert und zwei Verfahren vorgestellt, die die Kontextinformationen verwenden, um die Trajektorien auf Spektrumebene zu glätten.

2.6.1 Auswahl und Berechnung der Kontextinformationen

Unter der Annahme, dass die Produktion und Äußerung der Sprache durch eine Funktion erfolgt, bietet sich die Ableitung als Information über den Kontext eines Frames an und kann als Dynamik dessen angesehen werden. Neben den Merkmalen eines Frames würde zusätzlich der Wert der Ableitung verwendet werden. Da keine Funktionsvorschrift für die Produktion von Sprache bekannt ist, lässt sich das Signal nur numerisch differenzieren, beispielsweise mit $(y_t)' \approx \frac{y_{t+1} - y_{t-1}}{2}$. Mit Δy_t ist die Approximation $(y_t)'$ gemeint und in Kapitel 6 werden weitere Approximationen vorgestellt.

Wir suchen nun eine lineare Abbildung, die die Merkmalsequenz y in eine Sequenz Y umwandelt, die zum Zeitpunkt Y_t aus Merkmalen und Kontextinformationen $\begin{bmatrix} y_t \\ \Delta y_t \end{bmatrix}$ besteht:

$$Y = Wy.$$

y_t habe D Dimensionen und sei ein vorverarbeiteter Merkmalvektor, y sei die vertikale Konkatenation aller Merkmalvektoren einer Sequenz y.

$$Y = \underbrace{\begin{array}{c} \\ y_1 \\ \Delta y_1 \\ y_2 \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & \dots & & T \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{2} & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 & \dots \end{pmatrix} \end{array}}_{W} \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_T \end{bmatrix}}_{y} \tag{2.15}$$

Gleichung 2.15 stellt die Matrix W schematisch dar. Die Identitätsmatrix I ist aus $\mathbb{R}^{DT \times DT}$ und ein Block der Größe $D \times D$ auf der Diagonalen ist die Identitätsmatrix $I^{(D)}$ der Größe $D \times D$. Daraus folgt aber auch, dass man in Zeilenblock i durch Setzen der Spaltenblöcke $i-1$ und $i+1$ mit $-aI^{(D)}$ und $aI^{(D)}$ eine Berechnung der Form $a(y_{i+1} - y_{i-1})$ realisieren kann. Man erhält schließlich die Kontextinformationen, indem man für a die Koeffizienten einsetzt und um die Repräsentation von Y_t zu erreichen, wechseln sich in Matrix W die Zeilenblöcke für y_t und Δy_t ab. Bei den einzelnen Einträgen aus der schematischen Abbildung der Matrix $W \in \mathbb{R}^{2DT \times DT}$ handelt es sich folglich um $aI^{(D)}$:

$$\begin{pmatrix} a & 0 & \dots & 0 \\ 0 & a & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a \end{pmatrix} \text{ mit } a \in \{0, 1, -\frac{1}{2}, \frac{1}{2}\}$$

2.6.2 Gaußmischmodell

Das aus 2.4.2 bekannte gemeinsame Gaußmischmodell wird um die Kontextinformationen erweitert. Die Vektoren bestehen nun aus den Cepstrum-Merkmalen und der Approximation der Ableitung, also $X_t = \begin{bmatrix} x_t \\ \Delta x_t \end{bmatrix}$ und $Y_t = \begin{bmatrix} y_t \\ \Delta y_t \end{bmatrix}$. An der Konkatenation ändert sich nichts: $Z_t = \begin{bmatrix} X_t \\ Y_t \end{bmatrix}$. Die Parameter der einzelnen Normalverteilungen $\mathcal{N}_m$ lassen sich analog zu denen aus Abschnitt 2.4.2 formulieren:

$$\mu_m^{(Z)} = \begin{bmatrix} {\mu_m}^{(X)} \\ {\mu_m}^{(Y)} \end{bmatrix}, \Sigma_m^{(Z)} = \begin{pmatrix} \Sigma_m^{(XX)} & \Sigma_m^{(XY)} \\ \Sigma_m^{(YX)} & \Sigma_m^{(YY)} \end{pmatrix}.$$

Mit den bekannten Verfahren wird ein Modell $\lambda^{(Z)}$ trainiert und die Parameter der bedingten Wahrscheinlichkeitsdichtefunktion

$$P(Y_t|X_t, \lambda^{(Z)}) = \sum_{m=1}^{M} P(m|X_t, \lambda^{(Z)}) \underbrace{P(Y_t|X_t, m, \lambda^{(Z)})}_{\mathcal{N}(Y_t; E_{m,t}^{(Y)}; D_m^{(Y)})} \tag{2.16}$$

lauten analog:

$$E_{m,t}^{(Y)} = \mu_m^{(Y)} + \Sigma_m^{(YX)} \Sigma_m^{(XX)^{-1}} (X_t - \mu_m^{(X)})$$

$$D_m^{(Y)} = \Sigma_m^{(YY)} - \Sigma_m^{(YX)} \Sigma_m^{(XX)^{-1}} \Sigma_m^{(XY)}.$$

Auch die Konvertierungsfunktion nach Gleichung 2.13

$$\overline{Y}_t = \sum_{m=1}^{M} P(m|X_t, \lambda^{(Z)}) E_{m,t}^{(Y)}$$

ändert sich nicht. Allerdings bedeutet dies, dass wir für eine Aufnahme x erst die zugehörigen Kontextinformationen Δx berechnen müssen, bevor wir die Konvertierungsfunktion anwenden können. Als Ergebnis erhalten wir einen konvertierten Frame $\overline{Y}_t = \begin{bmatrix} \overline{y}_t \\ \Delta \overline{y}_t \end{bmatrix}$, der ebenfalls aus Merkmalen und Kontextinformationen besteht. Allerdings handelt es sich bei den Kontextinformationen lediglich um die aufgrund des Modells $\lambda^{(Z)}$ und des beobachteten Merkmalvektors X_t zu *erwartenden* Kontextinformationen. Dies wirkt sich möglicherweise insoweit limitierend aus, da die Konvertierung einerseits den quadratischen Fehler minimiert, aber dennoch fehlerbehaftet ist.

2.6.3 Maximum Likelihood Schätzer

Wir suchen eine Merkmalsequenz y, die die Wahrscheinlichkeit von Y unter einer Aufnahme x und unter dem Mischmodell $\lambda^{(Z)}$ maximiert:

$$\hat{y} = \arg\max_y P(Y|X, \lambda^{(Z)}) \tag{2.17}$$

Toda schlägt hierzu einen EM-Algorithmus vor, bei dem folgende Hilfsfunktion maximiert werden soll:

$$Q(\hat{Y}, Y) = \sum_{m=1}^{M} P(m|X, Y, \lambda^{(Z)}) \underbrace{\log P(\hat{Y}, m|X, \lambda^{(Z)})}_{P(m|\hat{Y}, \lambda^{(Z)}) P(\hat{Y}|X, \lambda^{(Z)})} \tag{2.18}$$

Diese Hilfsfunktion ist über die gesamte Sequenz definiert, wohingegen die Methode aus Abschnitt 2.5 nur ausschnittsweise konvertiert. Ähnlich der Herleitung des

ML-Schätzers beim Training wird nun die Definition eingesetzt und die Funktion anschließend abgeleitet und null gesetzt, um das Optimum zu finden.

$$
\begin{aligned}
&Q(\hat{Y}, Y) && (2.19)\\
&= \sum_{m=1}^{M} P(m|X, Y, \lambda^{(Z)}) \log P(\hat{Y}, m|X, \lambda^{(Z)}) && (2.20)\\
&= \sum_{t=1}^{T} \sum_{m=1}^{M} \underbrace{P(m|X_t, Y_t, \lambda^{(Z)})}_{\gamma_{m,t}} \log P(\hat{Y}, m|X, \lambda^{(Z)}) && (2.21)\\
&= \sum_{t=1}^{T} \sum_{m=1}^{M} \gamma_{m,t}(-\frac{1}{2}\hat{Y}_t^T D_m^{(Y)^{-1}} \hat{Y}_t + \hat{Y}_t^T D_m^{(Y)^{-1}} E_{m,t}^{(Y)}) + \underbrace{\overline{K}}_{\log \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma|^{\frac{1}{2}}}} && (2.22)\\
&= \sum_{t=1}^{T} -\frac{1}{2}\hat{Y}_t^T \overline{D}_t^{(Y)^{-1}} \hat{Y}_t + \hat{Y}_t^T \overline{D}_t^{(Y)^{-1}} \overline{E}_t^{(Y)} + \overline{K} && (2.23)\\
&= -\frac{1}{2} \underbrace{\hat{Y}^T}_{y^T W^T} \overline{D}^{(Y)^{-1}} \hat{Y} + \hat{Y}^T \overline{D}^{(Y)^{-1}} \overline{E}^{(Y)} + \overline{K} && (2.24)\\
&= -\frac{1}{2}\hat{y}^T W^T \overline{D}^{(Y)^{-1}} W\hat{y} + \hat{y}^T W^T \overline{D}^{(Y)^{-1}} \overline{E}^{(Y)} + \overline{K} && (2.25)
\end{aligned}
$$

Die maximierende Sequenz Y aus Gleichung 2.17 erfüllt die Bedingung der Kontextinformationen $\Delta y_t = \frac{1}{2}(y_{t+1} - y_{t-1})$ nicht ohne weiteres. Allerdings kann das erzwungen werden, indem die umgeformte Funktionsvorschrift aus Gleichung 2.25 nach $\hat{y}$ anstatt Y maximiert wird. Die Bedingung der Kontextinformationen ist dann erfüllt, da Y durch die lineare Abbildung $Y = Wy$ gebildet wird. Dazu wird die Funktion nach $\hat{y}$ abgeleitet und null gesetzt:

$$
\begin{aligned}
0 &= -W^T \overline{D}^{(Y)^{-1}} W\hat{y} + W^T \overline{D}^{(Y)^{-1}} \overline{E}^{(Y)} && (2.26)\\
W^T \overline{D}^{(Y)^{-1}} W\hat{y} &= W^T \overline{D}^{(Y)^{-1}} \overline{E}^{(Y)} && (2.27)\\
\hat{y} &= (W^T \overline{D}^{(Y)^{-1}} W)^{-1} W^T \overline{D}^{(Y)^{-1}} \overline{E}^{(Y)} && (2.28)
\end{aligned}
$$

mit

$$
\overline{D}^{(Y)^{-1}} = \begin{pmatrix} \overline{D}_1^{(Y)^{-1}} & 0 & \cdots & 0 \\ 0 & \overline{D}_2^{(Y)^{-1}} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \overline{D}_T^{(Y)^{-1}} \end{pmatrix} \text{ und } \overline{E}^{(Y)} = \begin{bmatrix} \overline{E}_1^{(Y)} \\ \overline{E}_2^{(Y)} \\ \vdots \\ \overline{E}_T^{(Y)} \end{bmatrix},
$$

wobei gilt

$$
\overline{D}_t^{(Y)^{-1}} = \sum_{m=1}^{M} \gamma_{m,t} D_m^{(Y)^{-1}}, \overline{E}_t^{(Y)} = \sum_{m=1}^{M} \gamma_{m,t} E_{m,t}^{(Y)}
$$

Bei $D_m^{(Y)}$ und $E_{m,t}^{(Y)}$ handelt es sich um die bekannten Parameter der bedingten Wahrscheinlichkeitsdichtefunktion aus dem vorangegangenen Abschnitt:

$$
\begin{aligned}
D_m^{(Y)} &= \Sigma_m^{(YY)} - \Sigma_m^{(YX)} \Sigma_m^{(XX)^{-1}} \Sigma_m^{(XY)} \\
E_{m,t}^{(Y)} &= \mu_m^{(Y)} + \Sigma_m^{(YX)} \Sigma_m^{(XX)^{-1}} (X_t - \mu_m^{(X)})
\end{aligned}
$$

Zusammenfassend kann man das Verfahren folgendermaßen beschreiben: Zuerst wird die bekannte Konvertierung aus Abschnitt 2.5 angewendet und anschließend werden die Kontextinformationen in der Berechnung von Gleichung 2.28 dazu verwendet, die einzelnen $\overline{y}_t$ mit dem Kontext ihrer Nachbarn zu modifizieren. Dies wird auch deutlich, wenn man Gleichung 2.15 betrachtet. In dieser Form berechnet W die Kontextinformationen, formuliert man jedoch die Gleichung zu $W^T Y = y$ um, werden die Kontextinformationen auf die Merkmale angerechnet.
Würde man keine Kontextinformationen verwenden, so wäre W die Einheitsmatrix und man würde die bekannte Konvertierung verwenden. Der einzige Unterschied bestünde darin, dass statt der gewichteten Wahrscheinlichkeiten $P(m|X_t, \lambda^{(Z)})$ aus dem Quellbereich die Zuordnungswahrscheinlichkeiten $\gamma_{m,t}$ verwendet werden würden.

2.6.4 Komponentensequenz

In der Referenz [TBT07] dient diese Variante zur Bestimmung einer Ausgangslösung für den ML-Schätzer aus dem vorangegangenen Abschnitt. Man nutzt zur Konvertierung eines Frames aus dem Quellbereich nicht das gesamte Modell $\lambda^{(Z)}$, sondern nur die Verteilung $\mathcal{N}_m$, die die höchste aposteriori-Wahrscheinlichkeit für den beobachteten Vektor X_t aus dem Quellbereich besitzt:

$$\hat{m}_t = \arg\max_m P(m|X_t, \lambda^{(Z)}) \tag{2.29}$$

Statt der Hilfsfunktion 2.18, die über alle M Gaußkomponenten iteriert, gilt es nun, folgende Funktion zu maximieren:

$$L = \log P(\hat{m}|X, \lambda^{(Z)}) P(Y|X, \hat{m}, \lambda^{(Z)}). \tag{2.30}$$

An der Form der Lösung

$$\hat{y} = (W^T \overline{D}^{(Y)^{-1}} W)^{-1} W^T \overline{D}^{(Y)^{-1}} \overline{E}^{(Y)} \tag{2.31}$$

ändert sich nichts, jedoch werden anstatt $\overline{D}_t^{(Y)^{-1}}$ und $\overline{E}_t^{(Y)}$ nun $D_{\hat{m}_t}^{(Y)^{-1}}$ und $E_{\hat{m}_t}^{(Y)}$ verwendet. In Kapitel 6 werden beide Varianten evaluiert.

2.7 Synthese

Grundlage für die Synthese, die Umwandlung in ein akustisches Format, das durch einen Computer abgespielt werden kann, ist der MLSA-Filter[ISF83]. Dieser schätzt das logarithmierte Mel-Spektrum anhand der Koeffizienten und benötigt als Eingabe sowohl eine konvertierte Merkmalsequenz $\overline{y}$ als auch eine F0-Sequenz.

3. Stand der Forschung

Die Arbeiten [AMS08, SMA09] und [SMA10] von Hamid Reza Sharifzadeh et al haben es zum Ziel, den Menschen, denen der Kehlkopf entfernt wurde, normale Artikulation zu ermöglichen. Ihr Ansatz ist dabei der „Code-excited Linear Prediction (CELP)"-Codec, der in einer Reihe von Voice-Chat-Programmen zum Einsatz kommt. Der Codec beruht auf dem Source-Filter-Modell, das auch im vorangegangenen Kapitel vorgestellt wurde und trennt das akustische Signal in die Anregung der Stimmbänder („excitation"), die Tonhöhe („pitch") und den Einfluss des Vokaltraktes. Des Weiteren verwendet der Codec Wörterbücher, um das Anregungssignal zu bilden: Es wird aus einem fixen Wörterbucheintrag (= codebook) und einem adaptiven Wörterbucheintrag, der sich nach den vergangenen Frames richtet, gebildet. Ein weiterer Bestandteil des Codecs ist die Verwendung von „linear predictive coding": Das aktuelle Signal $x[n]$ wird als Summe der vorangegangenen Signale $\sum_{i=1}^{n-1} a_i x[n-i]$ ausgedrückt. In der LPC-Analyse geht vor allem darum die a_i so zu optimieren, dass der Fehler zwischen dem Originalsignal und dem kodierten möglichst gering ist. Um aus geflüsterter Sprache mithilfe des CELP-Codecs hörbare zu machen, modifizieren die Autoren in ihrer letzten Arbeit zu diesem Thema [SMA10] den CELP-Codec folgendermaßen: Die geflüsterten Phoneme werden klassifiziert, um die Formanten der Vokale zu erkennen und entsprechend auf Spektrumebene den Formanten von normal geäußerten Vokalen anzupassen. Dies schlägt sich dann bei der Kodierung in den LPC Merkmalen nieder. Desweiteren werden anhand des geflüsterten Signals „pitch"- und „excitation"-Signale geschätzt und sich der Wörterbucheinträge bedient. Nach dem unveränderten Dekodieren durch das CELP-Verfahren wird das Ergebnis noch geglättet.
Helander et al [HSVG12] beschäftigen sich nicht speziell mit geflüsterter Sprache, sondern versuchen die Stimmenkonvertierung allgemein zu verbessern. Ihr Ausgangspunkt ist dabei, dass die gängigen Verfahren die einzelnen Sprachausschnitte unabhängig voneinander konvertieren und dadurch Störungen durch unstetige Übergänge entstehen. In ihrer Veröffentlichung „Voice Conversion Using Dynamic Kernel Partial Least Squares Regression" versuchen sie diesem Umstand entgegenzuwirken, indem benachbarte Sprachausschnitte bei der Konvertierung für einen Sprachausschnitt berücksichtigt werden. Ihr Vorgehen ist das folgende: Mithilfe eines Clustering-Algorithmus werden C Referenzvektoren bestimmt. Daraufhin wird für alle N Ausschnitte der Wert des Gaußkernels zu jedem Referenzvektor bestimmt und man erhält eine Matrix K, die aus C Zeilen und N Spalten besteht, wobei die n-te Spalte dem Kernelvektor k_n des n-ten Vektors im Quellbereich entspricht. Die Kernelwerte werden schließlich auf den Basiswert 0 verschoben, indem die Mittelwerte für jede Zeile berechnet und von den Werten in der Zeile abgezogen werden. Die einzelnen Mittelwerte werden in einem Vektor μ abgespeichert. Anschließend wird die Dynamik der einzelnen Frames modelliert indem der zum Zeitpunkt t betrachtete, zentrierte Kernel mit seinen beiden Nachbarn zu einem Vektor $x_n = [k_{t-1}, k_t, k_{t+1}]^T$ gestapelt wird. Die Konvertierung wird schließlich über die Regression $y_n = \beta x_n + e_n$ vorgenommen, wobei y_n Frames aus dem Zielbereich sind. Die Regressionsmatrix β wird durch die Lösung des „Partially Least Squares (PLS)"-Problems berechnet und nicht durch die lineare Regression, die bei der Konvertierung mit GMMe verwendet

wird. PLS wird beschrieben als Regressionsmodell, das Zusammenhänge zwischen zwei Matrizen X und Y modelliert. Das Ziel ist es, ähnlich der Hauptkomponentenanalyse (engl. principal component analysis (PCA)), die Komponenten aus X zu erhalten, die nützlich sind, um Komponenten aus Y vorherzusagen, allerdings sollen bei PLS Komponenten aus X und Y gewonnen werden. Um die Regression zu lösen, werden neue, versteckte Variablen eingeführt, die aus Linearkombinationen der x_n bestehen. Als Metrik verwenden sie in ihrer Evaluation die „Spectral Distortion“ und erreichen mit ihrem Verfahren bessere Werte als auf Gaußmischmodellen basierende Verfahren.

4. BioKIT

In diesem Kapitel wird das Framework BioKIT des Institutes „Cognitive Science Lab“ von Prof. Tanja Schultz vorgestellt und auf die für die Implementierung relevanten Teile eingegangen. BioKIT ist ein Framework, das viele notwendigen Komponenten und Schnittstellen für verschiedene Aufgaben rund um Biosignale integriert. So ist es möglich, das Framework zur Spracherkennung und Stimmenkonvertierung zu verwenden. Andere Entwicklungszweige ermöglichen Gestenerkennung. Darüber hinaus wird an kognitiven Systemen gearbeitet, die feststellen können, ob eine Person gerade gestresst ist.
Im Folgenden wird auf den aktuellen Stand der Implementierung für die GMM-basierte Konvertierung eingegangen.

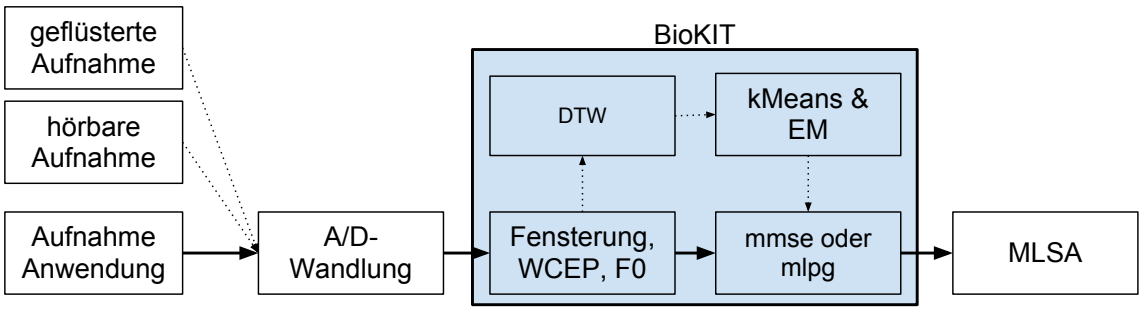

Abbildung 4.1: BioKIT-Implementation der Verarbeitungskette

4.1 Aufbau von BioKIT

Alle Algorithmen und Konstrukte, zum Beispiel um Gaußmischmodelle zu repräsentieren und zu verwalten, sind in Form von C++-Klassen und C++-Funktionen implementiert. Über ein Python-Interface sind die Schnittstellen der C++-Klassen für Python verfügbar und die einzelnen Schritte aus der Verarbeitungskette werden über Python-Skripte verbunden. In Abbildung 4.1 wird dargestellt, welche Komponenten der Verarbeitungskette in BioKIT implementiert sind. Der Aufruf der Implementierung des MLSA-Filters ist ebenfalls in den Skripten integriert. Eine Anleitung findet sich in Anhang A.

4.2 Vorhandene Infrastruktur

In BioKIT sind die in Kapitel 2 vorgestellten Algorithmen mit Ausnahme des zu implementierenden Verfahren aus Abschnitt 2.6 bereits entwickelt und funktionsfähig. Die C++-Klassen und deren relevante Schnittstellen werden nun schematisch erläutert.

4.2.1 `class` NumericMatrix, `class` NumericVector

Diese beiden Klassen stellen den mathematischen Unterbau bereit, um die Algorithmen zu implementieren. So sind die Operatoren zur Addition, Subtraktion, Multiplikation und Division überladen und die genannten Operationen sind zwischen

Instanzen dieser Klassen möglich. Desweiteren existieren Funktionen, die eine Matrix transponieren, invertieren, die Determinante berechnen und eine Spalte oder Zeile mit Werten füllen oder als Vektor zurückgeben. Intern ist die Matrix als eindimensionales Feld des Typs double umgesetzt. Legt man eine Instanz der Klasse an, so übergibt man dem Konstruktor die Anzahl Zeilen und Spalten. Es wird dann ein Feld der Größe $rows * columns$ angelegt, in dem die Zeilen hintereinander abgespeichert werden. Neben den genannten Funktionen kann man auch einzelne Indizes abfragen, dafür wird der abgefragte Index (x, y) nach $x * rows + y$ umgerechnet, um auf den richtigen Eintrag im Feld zuzugreifen. Eine Instanz von `NumericVector` verwaltet intern ein Feld des Typs double und man übergibt dem Konstruktor die Dimension des Vektors. Ferner sind Funktionen zum Setzen und Abfragen einzelner Elemente vorhanden.

4.2.2 `class` FeatureSequence, `class` FeatureVector

Diese beiden Klassen repräsentieren einen Merkmalvektor und eine Merkmalsequenz und sind intern mit Instanzen von `NumericVector` respektive `NumericMatrix` realisiert. Während `NumericMatrix` und `NumericVector` nur als mathematische Konstrukte gedacht sind, sollen `FeatureSeqeuence` und `FeatureVector` explizit für Merkmalvektoren und -sequenzen verwendet werden. Zudem werden sie als Referenzen verwendet.

4.2.3 `class` SingleGaussian

Die Klasse `SingleGaussian` repräsentiert eine Gauß'sche Normalverteilung und verfügt daher über Attribute, um die Parameter Gewicht, Mittelwert und Kovarianz vorzuhalten. Für das in Abschnitt 2.4.3 eingeführte Training existieren Attribute, die als Akkumulatoren der Parameter dienen.
Für das EM-Training gibt es die Funktionen `accumulate`() und `doUpdate`(), diese implementieren allerdings nicht den konzeptionellen Algorithmus aus Abschnitt 2.4.3. Stattdessen wird eine effizientere Vorgehensweise verwendet: In der Methode `accumulate`() werden die Akkumulatoren verwendet, um den Maximization-Schritt vorzubereiten. Der Expectation-Schritt wird in der gleichnamigen Funktion der Klasse `GaussianMapping` vorgenommen. Die Funktion `doUpdate`() finalisiert die Berechnung der neuen Parameter. Für die Anwendung existiert die Funktion `mapSingleFrame`(), die Funktionen `getLikelihood`() und `scoreSourceVector`() werden an verschiedenen Stellen benötigt.

FeatureVector `mapSingleFrame`(**FeatureVector** *source*)
Die Funktion `mapSingleFrame`() bildet einen übergebenen Merkmalvektor x_t aus dem Quellbereich nach Gleichung 2.11 ab.

function MAPSINGLEFRAME(FeatureVector source)
  return $(\mu^{(y)} + \Sigma^{(yx)}\Sigma^{(xx)^{-1}}(source - \mu^{(x)}))$
end function

double `getLikelihood`(**FeatureVector** *source*)
Die Funktion `getLikelihood`() wertet die Normalverteilung mit dem übergebenen Merkmalvektor *source* aus und multipliziert sie mit dem Gewicht der Normalverteilung.

function GETLIKELIHOOD(FeatureVector source)
  return $w * N(source; \mu^{(x)}; \Sigma^{(xx)})$
end function

double `scoreSourceVector`(**FeatureVector** *source*)
Die Funktion `score`() berechnet den Logarithmus der Wahrscheinlichkeit.

function SCORE(FeatureVector source)
  return $N(source; \mu^{(x)}; \Sigma^{(xx)})$
end function

void `accumulate`(**NumericVector** *datapoint*, **double** *weight*)
Die Funktion `accumulate`() gehört zum Maximization-Schritt aus dem EM-Training und bereitet die Berechnung der neuen Parameter vor.

function ACCUMULATE(NumericVector datapoint, double weight)
  w_{akku} += weight
  μ_{akku} += μ_{akku} + likelihood∗datapoint
  Σ_{akku} += likelihood $*$ (datapoint)(datapoint)^T
end function

void `doUpdate`(**double** *sumOfWeights*)
Die Funktion `doUpdate`() ist ebenfalls dem Maximization-Schritt zuzuordnen, sie finalisiert die Berechnung der neuen Parameter und setzt die internen Variablen.

function DOUPDATE(double sumOfWeights)
  $w_{new} = \frac{1}{sumOfWeights} * w_{akku}$
  $\mu_{new} = \frac{1}{w_{akku}} * \mu_{akku}$
  $\Sigma = \frac{1}{w_{akku}} \Sigma_{akku} - \mu_{new}\mu^T - \mu\mu_{new}^T + \mu\mu^T$
  $w = w_{new}$
  $\mu = \mu_{new}$
end function

4.2.4 `class` GaussianMapping

Die Klasse `GaussianMapping` verwaltet eine zum Konstruktor übergebene Anzahl Instanzen der Klasse `SingleGaussian` und repräsentiert ein GMM. Im Wesentlichen ist die Klasse ein Container für einzelne Gaußkomponenten und generalisiert die Funktionen aus der Klasse `SingleGaussian` auf ein GMM. Ferner existiert eine Funktion, die das kMeans-Training durchführt. `mapSequence`() konvertiert eine Merkmalsequenz.

NumericVector `getGammas`(**NumericVector** *x*)
Die Funktion `getGammas` berechnet die einzelnen $\gamma_{m,k}$ für einen gegebenen Vektor z_k und wird für das Training verwendet.

function GETGAMMAS(FeatureVector x)
  for all m=1,..,M **do**
    likelihood = getLikelihood$_m$(x)
    γ[m] = likelihood
    sum += likelihood
  end for
  if sum **not equals** 0 **then**
    γ *= $\frac{1}{sum}$
  else
    for all m=1,..,M **do**
      γ[m] = $\frac{1}{M}$
    end for
  end if

```
    return γ
end function
```

FeatureVector `mapSingleFrame`(**FeatureVector** *source*)
Die Funktion `mapSingleFrame`() bildet einen einzelnen Vektor x_t auf den Zielvektor $\overline{y}_t$ nach Gleichung 2.13 ab.

```
function mapSingleFrame(FeatureVector source)
    γ = getGammas(source)
    for all m in M do
        expectation += γ[m]*mapSingleFrame_m(source)
    end for
    return expectation
end function
```

FeatureSequence `mapSequence`(**FeaturesSequence** *sourcesequence*)
`mapSequence`
Die Funktion `mapSequence`() iteriert über eine gegebene Sequenz x und konvertiert sie ausschnittsweise zu $\overline{y}$.

```
function mapSequence(FeatureSequence sourcesequence)
    for all t = 1...|sourcesequence| do
        target[t] = mapSingleFrame(sourcesequence[t])
    end for
    return target
end function
```

double `getLiklihood`(**Feature*** *source*)
Die Funktion `getLikelihood` berechnet die Wahrscheinlichkeit eines Vektors x_t oder einer Sequenz x unter dem Modell $\lambda^{(z)}$.

```
function getLikelihood(FeatureVector source)
    likelihood = 0
    for all m in M do
        likelihood += getLikelihood_m(source)
    end for
    return likelihood
end function
function getLikelihood(FeatureSequence sourcesequence)
    likelihood = 0
    for all t = 1...|sourcesequence| do
        likelihood += getLikelihood(sourcesequence[t])
    end for
    return likelihood
end function
```

double `score`(**FeatureSequence** *sourcesequence*)
Die Funktion `score`() berechnet die logarithmierte Wahrscheinlichkeit einer Sequenz x.

```
function score(FeatureSequence sourcesequence)
    score = 0
    for all t = 1...|sourcesequence| do
        score += log getLikelihood(sourcesequence[t])
    end for
```

```
    score /= |sourcesequence|
    return score
  end function
```

void `accumulate`(**FeatureSequence** *trainingData*)
Diese Funktion generalisiert die Schnittstelle `accumulate`() der einzelnen Gaußkomponenten zu der des Mischmodells und erwartet die komplette Menge an Trainingsdaten als Eingabe. Die Zuordnungswahrscheinlichkeiten werden berechnet und den einzelnen Gaußkomponenten zur Berechnung der neuen Parameter mitgegeben.

```
  function ACCUMULATE(FeatureSequence trainingData)
    for all n = 1...|trainingData| do
      weight = getGammas(trainingData[n])
      for all m in M do
        accumulate_m(trainingData[n], weight[m])
      end for
    end for
  end function
```

void `doUpdate`()
Diese Funktion generalisiert die Schnittstelle `doUpdate`() der einzelnen Gaußkomponenten und trägt die Gewichte der Gaußkomponenten zusammen.

```
  function DOUPDATE
    sumOfWeights = 0
    for all m in M do
      sumOfWeights += getWeightAccu_m()
    end for
    for all m in M do
      doUpdate_m(sumOfWeights)
    end for
  end function
```

5. Implementierung

In diesem Kapitel wird erläutert, welche Funktionen der Implementierung hinzugefügt wurden und wie sie implementiert wurden.

5.1 Vorgenommene Erweiterungen an BioKIT

5.1.1 `class` SingleGaussian

NumericMatrix `getInvCovXY`(**int** *dimFeature*)
Wie in Abschnitt 2.6 erläutert, werden die invertierten Kovarianzmatrizen für das Verfahren benötigt. Dafür wird die Methode `getInvCovXY`() hinzugefügt, die die Kovarianzmatrix $D_m^{(Y)}$ nach 2.12 berechnet, invertiert und zurückgibt. Um die Teilmatrizen richtig auslesen zu können, wird der Parameter *dimFeature* übergeben.

function GETINVCOVXY( )
  return $\left(\Sigma^{(yy)} - \Sigma^{(yx)}\Sigma^{(xx)^{-1}}\Sigma^{(xy)}\right)^{-1}$
end function

5.1.2 `class` GaussianMapping

Das Training des GMM ist unabhängig der verwendeten Daten, sodass an den dafür zuständigen Funktionen keine Änderungen notwendig sind, lediglich die Python-Skripte werden angepasst. Für die Konvertierung wird eine Funktion namens `mapSequenceDynFeatures`() hinzugefügt. Darüber hinaus wurde eine weitere Hilfsfunktion notwendig.
int[] `getIndexSequence`(**FeatureSequence** *x*)
Berechnet für eine übergebene Merkmalsequenz x die einzelnen Gaußkomponenten $\hat{m}_t$ für die „Komponentensequenz"-Variante aus Abschnitt 2.6.4.

```
function getIndexSequence(FeatureSequence x)
    for t = 1,...,T do
        index = 0
        index_likelihood = −∞
        for all m = 1,..,M do
            current_likelihood = getLikelihood_m(x_t)
            if current_likelihood > index_likelihood then
                index = m
                index_likelihood = current_likelihood
            end if
        end for
        component_sequence[t] = index
    end for
    return component_sequence
end function
```

FeatureSequence `mapSequenceDynFeatures`(**FeatureSequence** *source*,
  bool *viterbi*, **int** *dimSource*, **NumericVector** *window*)
Die Funktion `mapSequenceDynFeatures`() konvertiert eine Sequenz x nach einer der

beiden vorgestellten Varianten aus Abschnitt 2.6 in eine Sequenz $\overline{y}$. Die Form der Lösung nach Gleichung 2.28 lautet:

$$\overline{y} = \underbrace{(W^T D^{(Y)^{-1}} W)^{-1}}_{A^{-1}} \underbrace{W^T \overbrace{D^{(Y)^{-1}} E^{(Y)}}^{DE}}_{b} \tag{5.1}$$

$$\overline{y} = A^{-1} b \tag{5.2}$$

$$A\overline{y} = b \tag{5.3}$$

Eine erste, naive Implementierung bei der die von BioKIT bereitgestellten mathematischen Operatoren verwendet und die Matrizen wie in 5.1 konstruiert und berechnet werden, stellt sich als ineffizient heraus. Schritt für Schritt wurden daher einige Optimierungen vorgenommen. Die Multiplikation $A^{-1}b$ wurde in das lineares Gleichungssystem 5.3 überführt: Das Lösen eines Gleichungssystems benötigt bei einer $n \times n$-Matrix im schlechtesten Fall n^3 Operationen wohingegen für die Invertierung und die anschließende Multiplikation $n^3 + n^2$ Operationen notwendig werden. Durch die vermiedene Invertierung ist das Verfahren numerisch stabiler.
Ferner werden nicht alle $\overline{D_t}^{(Y)^{-1}}$ und $\overline{E_t}^{(Y)}$ gesammelt und dann die entsprechende Operation durchgeführt, sondern direkt beim Iterieren über die Merkmalsequenz, wenn $\overline{D_t}^{(Y)^{-1}}$ und $\overline{E_t}^{(Y)}$ bekannt sind. Diese Aufgaben werden von den Hilfsfunktionen `calcSequence`() und `calcMatrix`() übernommen.
Sind die Bestandteile A und b berechnet, wird die externe Bibliothek LAPACK[lap] für die Lösung des Gleichungssystems verwendet. Dabei wird die Routine dgbsv[dgb] eingesetzt, die für Bandmatrizen optimiert ist, denn bei A handelt es sich nach der Referenz[TBT07] um eine Bandmatrix. Die Lösung, die von dem Aufruf zurückgegeben wird, muss schließlich noch in eine Merkmalsequenz, also eine Instanz der Klasse `FeatureSequence`, umgewandelt werden, bevor sie zurückgegeben wird.
Da intern keine Konfiguration verwaltet wird, müssen der Funktion neben der Merkmalsequenz x noch weitere Parameter übergeben werden: *viterbi* gibt an, welche der beiden vorgestellten Varianten aus Abschnitt 2.6 verwendet werden soll. In *window* ist die Zusammensetzung der Kontextinformationen gespeichert und *dimSource* gibt die Dimensionalität der Daten aus dem Quellbereich an.

```
function mapSequenceDynFeatures(x, window, dimSource, viterbi)
    if viterbi then
        component_sequence = getIndexSequence(x)
        for t = 1,...,T do
            m = component_sequence[t]
            expectation = mapSingleFrame_m(x_t)
            covariance = getInvCovXY_m(dimSource)
            DE = covariance*expectation
            calcMatrix(A, t, covariance, window)
            calcSequence(b, t, DE, window)
        end for
    else
        for t = 1,...,T do
            gamma = getGammas(x_t)
            expectation = 0
            covariance = 0
```

```
            for m = 1,...,M do
                expectation += gamma[m]*mapSingleFrame_m(x_t)
                covariance += gamma[m]*getInvCovXY_m(dimSource)
            end for
            DE = covariance*expectation
            calcMatrix(A, t, covariance, window)
            calcSequence(b, t, DE, window
        end for
    end if
    LAPACK->dgbsv(A,b)
    b.reshapeInplace(dimFeature)
    return b
end function
```

5.1.3 `class` NumericMatrix

In den verschiedenen Evolutionsstufen der Implementierung sind in der Klasse `NumericMatrix` einige Hilfsfunktion entstanden, letztlich werden nur die folgenden verwendet. Um den Code übersichtlicher zu gestalten, bezeichnet der Ausdruck *valid(index)* die Überprüfung, ob ein gegebener Index gültig ist und kürzt damit einen Ausdruck der Form [*index* $\geq$ *0 && index* $<$ *T*] oder [*index* $\geq$ *0 && index* $<$ *rows*] ab.

NumericMatrix `getMatrixWithDeltas`(**NumericVector** *window*)

Berechnet nach übergebenem Vektor *window* Kontextinformationen. Der zu übergebene Vektor *window* gibt an, wie sich die Kontextinformationen zusammensetzen und ist für $\Delta y_t = \frac{1}{2}(y_{t+1} - y_{t-1})$ wie folgt aufgebaut:

Index	0	1	2
Koeffizient	$-\frac{1}{2}$	1	$\frac{1}{2}$

Der Zeitpunkt t ist durch den Index $center = \lfloor \frac{|window|}{2} \rfloor$ repräsentiert. Dadurch erfährt man ebenfalls, wie viele Merkmalvektoren in jede Richtung in die Berechnung der Kontextinformationen einfließen. Iteriert man über den Vektor mit i, so beschreibt $offset = i - center$ die Verschiebung, die für die Berechnung des Zeitindizes $deltaTime = t + offset$ benötigt wird. Der Index $deltaTime$ gibt dann den Vektor an, der mit dem i-ten Koeffizienten multipliziert werden muss.

Eine Merkmalsequenz lässt sich als Matrix realisieren, in der jeder Frame als Zeile abgespeichert ist. Die Funktion übernimmt dann die Berechnung $Y = Wy$ aus Abschnitt 2.6.1, indem eine neue Matrix angelegt wird, die die doppelte Menge an Spalten gegenüber der Eingabematrix besitzt und in den neuen Spalten die Kontextinformationen speichert.

```
function getMatrixWithDeltas(window)
    center = |window|/2
    for row = 0,...,rows-1 do
        for column = 0,...,columns-1 do
            newMat[row][column] = oldMat[row][column]
            for coef = 0,...,|window|-1 do
                offset = coef - center
                deltaRow = row + offset
                deltaColumn = oldColumns + column
                if valid(deltaRow) and coef ≠ center then
                    delta = window[coef] * oldMat[row][column]
                    newMat[deltaRow][deltaColumn] += delta
```

```
            end if
        end for
    end for
  end for
  return newMat
end function
```

void `calcSequence`(**NumericMatrix** b, **int** t, **NumericVector** *window*, **NumericVector** DE)

Diese Funktion berechnet den Teil $b = W^T DE$ aus Gleichung 5.3. Da durch die Merkmalsequenz durchiteriert wird, ist immer nur ein Vektor $DE_t = D_t^{(Y)^{-1}} E_t^{(Y)}$ bekannt. Die Berechnung $b_t = DE_t - \frac{1}{2}\Delta DE_{t-1} + \frac{1}{2}\Delta DE_{t+1}$ für die Standardkontextinformationen wird daher aufgespalten in Teilberechnungen, die mit dem zum Zeitpunkt t vorhandenen Teilergebnis ausgeführt werden können:

$$
\begin{aligned}
b_{t-1} &= b_{t-1} + \frac{1}{2}\Delta DE_t \\
b_t &= b_t \quad + \quad DE_t \\
b_{t+1} &= b_{t+1} - \frac{1}{2}\Delta DE_t
\end{aligned}
$$

Damit diese Berechnungen so durchgeführt werden können, muss der Funktion neben dem Vektor DE noch die aktuelle Position t und die Zusammensetzung der Kontextinformationen *window* übergeben werden.

```
function calcSequence(b,t,DE,window)
  T = rows / dimFeature
  center = |window| / 2
  for f = 0,...,dimFeature-1 do
    for coef = 0,...,|window|-1 do
      offset = coef − center
      deltaTime = t + offset
      if valid(deltaTime) then
        b[deltaTime*dimFeature+f] += window[coef] * DE[f]
      end if
    end for
  end for
end function
```

void `calcMatrix`(**NumericMatrix** A, **int** t, **NumericVector** *window*, **NumericMatrix** $\overline{D_t}^{(Y)^{-1}}$)

Diese Funnktion berechnet den Teil $A = W^T \overline{D}^{(Y)^{-1}} W$ aus Gleichung 5.3. Wie schon bei der zuvor erläuterten Funktion `calcSequence`() verfügt man in einem Iterationsschritt nur über die aktuelle Matrix $D_t^{(Y)^{-1}}$. Für eine effiziente Implementierung wurde die Berechnung für die Standardkontextinformationen analysiert: Die Blöcke aus W besitzen die Dimension $D \times D$, $D_t^{(Y)^{-1}}$ jedoch die Dimension $2D \times 2D$ und daher betrachten wir wieder die vier Teilmatrizen

$$
\overline{D_t}^{(Y)^{-1}} = \begin{bmatrix} D_t^{(11)} & D_t^{(12)} \\ D_t^{(21)} & D_t^{(22)} \end{bmatrix},
$$

um die Berechnung veranschaulichen zu können. Das Zwischenergebnis $H = \overline{D}^{(Y)^{-1}} W$ berechnet sich dann aus

$$\overbrace{\begin{pmatrix} D_1^{(11)} & D_1^{(12)} & 0 & \dots & & \dots & 0 \\ D_1^{(21)} & D_1^{(22)} & 0 & \ddots & & & \vdots \\ 0 & 0 & D_2^{(11)} & D_2^{(12)} & \ddots & & \\ \vdots & \ddots & D_2^{(21)} & D_2^{(22)} & & & \vdots \\ & & \ddots & & \ddots & & 0 \\ \vdots & & & & & D_T^{(11)} & D_T^{(12)} \\ 0 & \dots & & \dots & 0 & D_T^{(21)} & D_T^{(22)} \end{pmatrix}}^{D^{(Y)^{-1}}} \overbrace{\begin{pmatrix} 1 & 0 & \dots & & & \\ 0 & \frac{1}{2} & 0 & \dots & & \\ 0 & 1 & 0 & \dots & & \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 & \dots & \\ 0 & 0 & 1 & 0 & \dots & \\ 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 & \dots \\ & & & \ddots & & \end{pmatrix}}^{W}$$

und hat folgende Gestalt:

$$H = \begin{array}{c} \\ i-1 \\ \vdots \\ i \\ \vdots \\ i+1 \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccc} & \dots & i-1 & i & i+1 & \dots \end{array} \\ \begin{pmatrix} \dots & -\frac{1}{2}D_{i-1}^{(12)} & D_{i-1}^{(11)} & \frac{1}{2}D_{i-1}^{(12)} & \dots & 0 \\ \dots & -\frac{1}{2}D_{i-1}^{(22)} & D_{i-1}^{(21)} & \frac{1}{2}D_{i-1}^{(22)} & \dots & 0 \\ 0 & \dots & -\frac{1}{2}D_{i}^{(12)} & D_{i}^{(11)} & \frac{1}{2}D_{i}^{(12)} & 0 \\ 0 & \dots & -\frac{1}{2}D_{i}^{(22)} & D_{i}^{(21)} & \frac{1}{2}D_{i}^{(22)} & 0 \\ 0 & 0 & \dots & -\frac{1}{2}D_{i+1}^{(12)} & D_{i+1}^{(11)} & \frac{1}{2}D_{i+1}^{(12)} \\ & 0 & \dots & -\frac{1}{2}D_{i+1}^{(22)} & D_{i+1}^{(21)} & \frac{1}{2}D_{i+1}^{(22)} \end{pmatrix} \end{array}$$

Schließlich gilt es, die Multiplikation $W^T H$ zu analysieren. W^T verfügt über folgende Gestalt:

$$\begin{array}{c} \ddots \\ i-1 \\ i \\ i+1 \end{array} \begin{array}{c} \begin{array}{ccccccccc} i-1 & & i & & i+1 & & & & \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & -\frac{1}{2} & 0 & \dots & & & \\ 0 & \frac{1}{2} & 1 & 0 & 0 & -\frac{1}{2} & 0 & \dots & \\ & \dots & 0 & \frac{1}{2} & 1 & 0 & 0 & -\frac{1}{2} & 0 & \dots \end{pmatrix} \end{array}$$

Die Skalarprodukte der i-te Zeile aus W^T und den Spalten aus H sind ungleich null für die Spalten $i-2$ bis einschließlich $i+2$. Die Ergebnisse setzen sich wie folgt zusammen:

$$\begin{aligned} A_{i,i-2} &= -\frac{1}{4}D_{i-1}^{(22)} \\ A_{i,i-1} &= \frac{1}{2}D_{i-1}^{(21)} - \frac{1}{2}D_{i}^{(12)} \\ A_{i,i} &= \frac{1}{4}D_{i-1}^{(22)} + D_{i}^{(11)} + \frac{1}{4}D_{i+1}^{(22)} \\ A_{i,i+1} &= \frac{1}{2}D_{i}^{(12)} - \frac{1}{2}D_{i+1}^{(21)} \\ A_{i,i+2} &= -\frac{1}{4}D_{i+1}^{(22)} \end{aligned}$$

Anhand der Zusammensetzungen der Blockeinträge lässt sich nun feststellen, was mit den Teilmatrizen aus $\overline{D_t}^{(Y)^{-1}}$ geschieht:

$$\underbrace{\begin{array}{cc} \ddots & \begin{array}{c} -1 \\ t \end{array} \\ t & \begin{pmatrix} D_t^{(11)} & D_t^{(12)} \\ D_t^{(21)} & D_t^{(22)} \end{pmatrix} \end{array}}_{D^{(Y)}} \Rightarrow \underbrace{\begin{array}{c|ccccc} & t-2 & t-1 & t & t+1 & t+2 \\ \hline t-1 & & \frac{1}{4}D_t^{(22)} & -\frac{1}{2}D_t^{(21)} & & \\ t & -\frac{1}{4}D_t^{(22)} & -\frac{1}{2}D_t^{(12)} & D_t^{(11)} & \frac{1}{2}D_t^{(12)} & \frac{1}{4}D_t^{(22)} \\ t+1 & & & \frac{1}{2}D_t^{(21)} & \frac{1}{4}D_t^{(22)} & \end{array}}_{A}$$

Diese Berechnungen werden durch die Funktion flexibel anhand des übergebenen Vektors *window* realisiert. Dazu benötigt die Funktion neben der Matrix $\overline{D_t}^{(Y)^{-1}}$ und dem Vektor *window* noch den die aktuelle Position t und die Dimension D als Parameter.
Die Ergebnismatrix A, wenn alle T Iterationen abgeschlossen sind, wird direkt an die bereits erwähnte LAPACK-Routine dgbsv übergeben, deren Eigenschaften beachtet werden müssen: Zum einen ist LAPACK in Fortran implementiert und Matrizen sind dort spalten- und nicht zeilenweise in einem eindimensionalen Feld realisiert und zum anderen erwartet dgbsv die Matrix in einem optimierten Format. Das Format ist durch die Transformation:

$$A_{band}[KL + KU + i - j, j] = A[i, j],$$

wobei KU die Anzahl der Diagonalen oberhalb und KL die Anzahl der Diagonalen unterhalb der Hauptdiagonalen sind, gegeben[dgb]. KL und KU sind in unserem Fall gleich und lassen sich durch $(|window| * D) - 1$ berechnen. Um weiteren Aufwand zu sparen, übernimmt die Funktion neben der Berechnung auch die Vorbereitung des Datenformats auf die LAPACK-Routine, indem A direkt in das notwendige Format gebracht wird.
Wir nutzen dabei aus, dass wir über die Transposition die zeilenweise abgelegte Matrix in eine spaltenweise abgelegte überführen können und legen folglich das Ergebnis des Skalaprodukts der i-ten Zeile aus W^T mit der j-ten Spalte aus H an Position $A[j, KL + KU + i - j]$ ab.
Die Hilfsfunktionen `multXY`, `multYX`, `multYY` dienen hier lediglich der Übersicht. Man beachte, dass mit A[row][column] der Wert aus Zeile *row* und Spalte *column* der Matrix A gemeint ist und die Variablen *currentRow*, *currentCol* auch die tatsächlich aktuellen Zeilen- und Spaltenindizes aus A enthalten. Allerdings werden diese jeweils um die durch den Vektor *window* gegebene Verschiebung angepasst. In der Variable *idxBase* wird der Basisindex für die genannte Transformation in das Bandmatrixformat gespeichert. Die Transposition wird schließlich durch das Vertauschen von Zeile und Spalte der berechneten Position erreicht.

function CALCMATRIX(t,D_t^{-1},window)
  diagonals = 2 ∗ (|window|∗dimFeature−1)
  length = $\frac{rows}{dimFeature}$
  center = $\frac{|window|}{2}$
  for row = 0,...,dimFeature-1 **do**
    currentRow = t∗dimFeature + row%dimFeature
    for column = 0,...,dimFeature-1 **do**
      currentColumn = t∗dimFeature + column%dimFeature

```
            idxBase = diagonals + currentRow - currentColumn
            x = D_t^{-1}[row][column]
            switch x do
                case x ∈ Σ_11:
                    A[currentColumn][idxBase] += x
                case x ∈ Σ_12: multXY(x)
                case x ∈ Σ_21: multYX(x)
                case x ∈ Σ_22: multYY(x)
        end for
    end for
end function
function MULTXY(x)
    for coef = 0,...,|window|-1 do
        columnOffset = coef − center
        deltaTime = t + columnOffset
        if valid(deltaTime) and coef ≠ center then
            rowIdx = idxBase − columnOffset∗dimFeature
            columnIdx = currentColumn + columnOffset∗dimFeature
            A[columnIdx][rowIdx] += x ∗ window[coef]
        end if
    end for
end function
function MULTYX(x)
    for coef = 0,...,|window|-1 do
        rowOffset = coef − center
        deltaTime = t + rowOffset
        if valid(deltaTime) and coef ≠ center then
            rowIdx = idxBase + rowOffset∗dimFeature
            A[currentCol][rowIdx] += x ∗ window[coef]
        end if
    end for
end function
function MULTYY(x)
    for rowCoef = 0,...,|window|-1 do
        rowOffset = rowCoef − center
        rowTime = t + rowOffset
        for columnCoef = 0,...,|window|-1 do
            columnOffset = columnCoef − center
            columnTime = t + columnOffset
            if valid(rowTime) and valid(columnTime) and coef ≠ center then
                rowIdx = idxBase + (rowOffset−columnOffset)∗dimFeature
                columnIdx = currentColumn + columnOffset∗dimFeature
                A[columnIdx][rowIdx] += x ∗ window[rowCoef] ∗ window[columnCoef]
            end if
        end for
    end for
end function
```

void `reshapeByRowInplace`(**int** *rows*, **int** *column*)
Im Stil von MATLAB existiert bereits eine Funktion namens `reshapeByRow`(), die die Matrix zeilenweise in die übergebenen Dimensionen umformt. Allerdings allokiert diese neuen Speicher und gibt eine neue Instanz zurück. Die LAPACK-Routine dgbsv gibt die Lösung des Gleichungssystem über den Eingabeparameter b zurück. In der Implementation ist b als NumericMatrix-Instanz umgesetzt, die über eine Spalte und $T * D$ Zeilen verfügt, wobei T die Anzahl Frames in der Merkmalsequenz und D die Anzahl Merkmale pro Ausschnitt sind. Um effizient zu bleiben, werden nur die internen Attribute verändert, die angeben, wie viele Zeilen und Spalten die Matrix hat.

```
function reshapeByRowInplace(dimFeatureVector)
    rows = rows/dimFeatureVector
    columns = dimFeatureVector
end function
```

6. Evaluation

6.1 Laufzeitverhalten

Die naive Implementierung nach 5.1 benötigte zum Konvertieren einer Merkmalsequenz auf einem gewöhnlichen Desktopcomputer mehrere Minuten. Durch die Umstellung auf das Lösen des Gleichungssystems 5.3 konnte die Konvertierung auf durchschnittlich 30 Sekunden gesenkt werden. Doch auch diese Ausführungszeit für die Konvertierung war deutlich zu langsam gemessen an der Konvertierungszeit des konventionellen Algorithmus von ca. 200ms. Mit den weiteren Optimierungen, wie dem Nutzen des Bandmatrixformats und der direkten Berechnung der Bestandteile A und b konnte eine Ausführungszeit von durchschnittlich 730ms erreicht werden, sodass man nur noch um ungefähr Faktor 3,5 langsamer ist. Allerdings steigt die Ausführungszeit deutlich an, wenn statt der Komponentensequenzvariante alle Gaußkomponenten konvertieren und je mehr benachbarte Frames bei der Zusammensetzung der Kontextinformationen eine Rolle spielen. Das Verfahren zum Glätten der Trajektorien ist notwendigerweise langsamer als der konventionelle Algorithmus, da neben der Konvertierung noch die Matrix A berechnet werden muss und anschließend das Gleichungssystem gelöst werden muss.
Da kein exklusiver Zugriff auf die Computer am CSL bestand, ist auch nicht gewährleistet, dass die genannten Werte absolut präzise sind, geben jedoch eine Größenordnung vor.

6.2 Experimentaufbau und Parameter

Der Testkorpus besteht aus zwölf Sprechern, davon sind sieben männlich und fünf weiblich, die alle die selben 200 Sätze aufgenommen haben. Davon wurden 140 für das Training verwendet, 30 zum Testen. Die Aufnahmen wurden mit einem Samsung Galaxy S2 durchgeführt und in einem Rohdatenformat abgespeichert. Auf diesen Rohdaten wurde die Vorverarbeitung aus Abschnitt 2.2 angewendet und 25 WCEP-Koeffizienten pro Frame berechnet. Daraufhin werden die Daten mittels DTW synchronisiert (vgl. 2.3) und die Frames werden für das bevorstehende Training gestapelt. Je nach verwendetem Verfahren müssen in diesem Schritt auch die die Kontextinformationen berechnet werden. Anschließend werden für jeden Sprecher und für jedes Verfahren GMMe mit kMeans- & EM-Algorithmus mit verschiedener Anzahl Gaußkomponenten trainiert. Dabei wird die Zahl der Gaußkomponenten ausgehend von einer Verteilung bis 32 immer verdoppelt. Schließlich wurden die 30 Aufnahmen zum Testen konvertiert und deren MCD-Wert zur synchronisierten Referenz berechnet.
Die fehlende Fundamentalfrequenz bei geflüsterter Sprache wurde für die Synthese folgendermaßen kompensiert: Ein weiteres GMM wird trainiert, dieses besteht aus Merkmalvektoren der geflüsterten Aufnahmen und F0-Werte aus den Aufnahmen in hörbarer Sprache. Hier wird das Verfahren zur Glätten der Trajektorien ebenfalls angewendet. Darüber hinaus wird ein Hidden-Markov-Modell trainiert, welches zwischen stimmhaft und stimmlos unterscheiden kann und das Ergebnis der F0-Konvertierung wird nur dann genommen, wenn das HMM einen Frame auch als stimmhaft dekodiert. Auch hier war das neue Verfahren eingesetzt, die Anzahl der

Gaußkomponenten ist jedoch festgesetzt auf acht.

Die verschiedenen, getesteten Approximationen der Ableitung sind in Tabelle 6.1

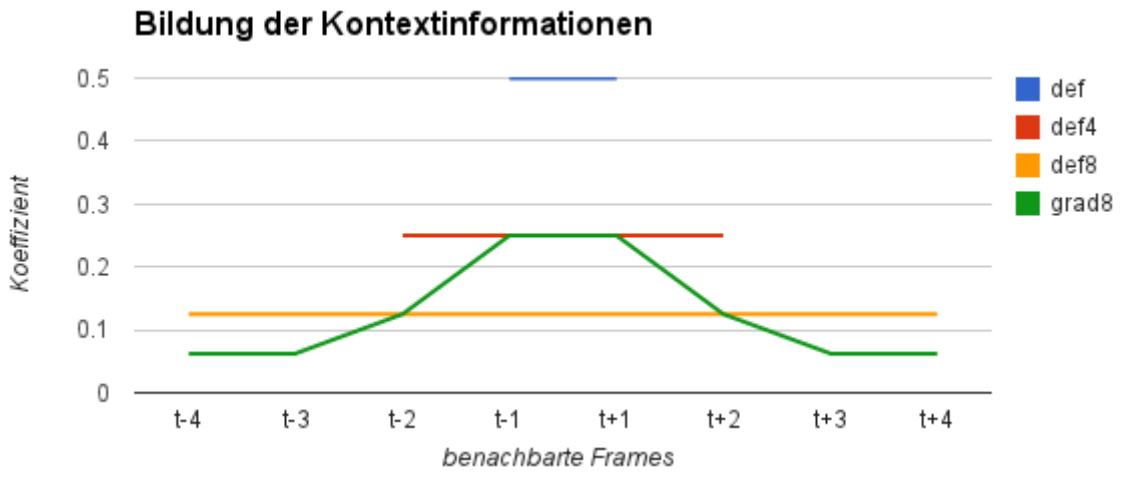

Abbildung 6.1: Approximation der Ableitung

gegeben und in Abbildung 6.1 dargestellt. Bei „def“ handelt es sich um die Kontextinformationen aus der Rerferenz [TBT07], „def 4“ und „def 8“ bilden die Kontextinformationen auf die gleiche Art und Weise, allerdings auf 4 bzw. 8 benachbarte Frames erweitert. „grad 8“ gewichtet die benachbarten Frames nach ihrem Abstand zum Frame t.

Das bereits vorhandene Verfahren zur Konvertierung nach Abschnitt 2.5 wird in

Tabelle 6.1: Zusammensetzung der Kontextinformationen

Name	**Zusammensetzung**
def	$\frac{1}{2}(x_{t+1} - x_{t-1})$
def4	$\frac{1}{4}(\sum_{k=1}^{2} x_{t+k} - x_{t-k})$
def8	$\frac{1}{8}(\sum_{k=1}^{4} x_{t+k} - x_{t-k})$
grad8	$\frac{1}{4}(x_{t+1} - x_{t-1}) + \frac{1}{8}(x_{t+2} - x_{t-2}) + \frac{1}{16}(\sum_{k=3}^{4} x_{t+k} - x_{t-k})$

den Abschnitten 6.4 und 6.5 als „conv“ bezeichnet und das im Rahmen der Arbeit implementierte Verfahren mit „mlpg vit *window*“, wobei anstelle von *window* die Namen der Zusammensetzungen der Kontextinformationen aus Tabelle 6.1 stehen. Die Kontextinformationen werden sowohl im Quell- als auch im Zielbereich berechnet. Der ML-Schätzer für „mlpg“ wurde ebenfalls getestet, dazu mehr im folgenden Abschnitt.

6.3 Maximum-Likelihood-Schätzer

Der Maximum-Likelihood-Schätzer nach 2.6.3 wurde ebenfalls getestet. Die Wahrscheinlichkeit einer konvertierten Merkmalsequenz stieg zwar für wenige Iterationen an, doch die MCD-Werte wurden deutlich schlechter als die der Frame-basierten Konvertierung. Des Weiteren ist die Implementierung der Gaußmischmodelle nicht robust genug, da es bei der Konvertierung mit 32 Gaußkomponenten keine verwertbaren Ergebnisse gab. Das Problem hierbei sind möglicherweise die $\gamma_{m,t}$: Werden diese sehr klein, kann es bei den Gleitkommaoperationen zu Problemen kommen.

Toda et al erachten das Ergebnis der Komponentensequenzmethode als gut genug und beobachteten ferner, dass eine Zielsequenz y unter dem trainierten Modell $\lambda^{(Z)}$ eine geringere Wahrscheinlichkeit erzielt als eine konvertierte Sequenz $\overline{y}$. Durch die iterative Form des Algorithmus und das Konvertieren mit allen Gaußkomponenten ist der ML-Schätzer ohnehin deutlich langsamer als die Komponentensequenzmethode und wird daher nicht weiter ausgewertet.

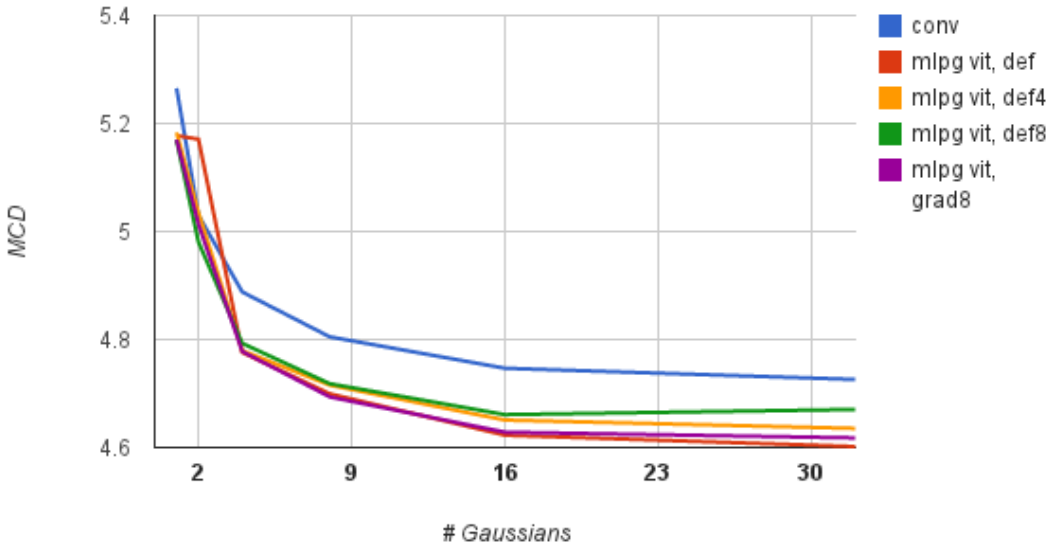

Abbildung 6.2: „mlpg“ mit verschiedenen Kontextinformationen resultiert in geringeren MCD-Werten als „conv“

6.4 Objektive Evaluation

Als Maß für eine objektive Auswertung der Konvertierungsergebnisse gibt es die „Mel-Cepstral-Distortion“, die Störung des auf die Mel-Skala verformten Cepstrum. Es handelt sich dabei um eine Metrik, die den Unterschied zwischen den einzelnen Cepstrum-Koeffizienten eines Referenzframes und eines konvertierten Frames logarithmisch darstellt:

$$MCD[dB] = \frac{10}{\ln 10}\sqrt{2\sum_{k=1}^{25}(y_{ref}[k] - \overline{y}[k])}$$

Für Abbildung 6.2 wurde der Durchschnitt über alle Sprecher auf dem Testset und unter den verschiedenen Konfigurationen gemittelt. Die Komponentensequenzmethode ist dabei unabhängig von der Anzahl der Gaußkomponenten besser als das bereits vorhandene Verfahren. Die durchschnittliche Verbesserung von etwa 0,1 findet man auch in der Referenz [TBT07] wieder. In Tabelle 6.2 sind die konkreten Werte angegeben. Ferner kann man beobachten, dass die Erweiterung des Kontexts auf mehr als nur die umgebenden Frames keine weitere Verbesserung bringt. Einzig die Variante, die die weiter entfernten Frames geringer gewichtet, kann ähnliche oder

Sprecher	conv	def2	def4	def8	grad8
101	4,226	4,159	4,163	*4,152	*4,142
102	4,534	4,433	4,475	4,496	*4,457
103	4,450	4,310	4,381	4,419	4,363
104	4,478	4,377	4,397	4,429	4,387
105	4,921	4,821	4,868	4,864	4,853
106	4,646	4,510	4,542	4,577	4,521
107	4,599	4,428	4,482	4,496	4,465
108	5,186	*5,012	*5,045	*5,056	*5,012
109	4,913	4,758	4,783	4,827	4,787
110	5,170	5,041	5,045	5,034	5,002
111	4,544	4,468	4,493	4.51	4,467
113	5,038	4,891	4,923	4,971	4,926

Tabelle 6.2: Die besten Durchschnittswerte der MCD für die einzelnen Sprecher auf den 30 Testsätzen wurden in der Regel mit 32 Gaußkomponenten erreicht, mit (*) gekennzeichneten Werte hingegen mit 16 Gaußkomponenten.

sogar besser Ergebnisse als die der Standardkontextinformationen vorweisen, siehe Specher 101 und 109. Allerdings sind die Abweichungen oft in der zweiten oder dritten Nachkommastelle festzustellen und daher keineswegs signifikant. Generell lässt sich festhalten, dass tendenziell „gute“ Sprecher wie 101 und 111 deutlich weniger von dem Verfahren profitieren als „schlechte“ Sprecher wie 108 und 109. Berechnet man den Durchschnitt der Veränderung, so ergibt sich ein Unterschied von 0,128 bei einer Standardabweichung von 0,03, sodass die Verbesserung als robust angenommen werden kann.

6.5 Hörtests

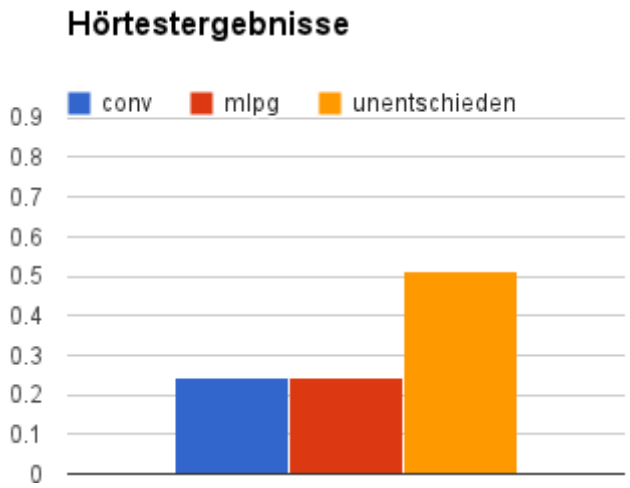

Abbildung 6.3: Ergebnisse der Hörtests

Aus den MCD-Werten lassen sich keine Aussagen über die Verständlichkeit und den Klang des Konvertierungsergebnisses ableiten. Die Verbesserung durch „mlpg“ ist messbar, doch wichtiger ist es, eine Verbesserung in Sprachqualität zu erhalten. Es wird ein Hörtest durchgeführt, indem beide Verfahren gegeneinander getestet

werden, um feststellen zu können, ob ein Unterschied hörbar ist und wie dieser ausfällt. Der Aufbau des Tests ist folgendermaßen: Die Testperson hört dieselbe synthetisierte Aufnahme eines Sprechers, einmal konvertiert mit „conv“ und einmal mit „mlpg vit grad8“. Der Person ist allerdings nicht bekannt, welche Audiodatei durch welches Verfahren entstanden ist. Anschließend stimmt die Testperson ab, welche Version besser klingt. Es gibt auch eine Antwortmöglichkeit für den Fall, dass kein Unterschied zu erkennen ist. Für jeden der zwölf Sprecher sind vier Sätze aus dem Testset ausgewählt worden. Insgesamt acht Personen haben den Hörtest durchgeführt.
Wie in Abbildung 6.3 zu sehen ist, konnten die Probanden in ein bisschen mehr als 50% der Fälle keinen Unterschied feststellen, sodass anzunehmen ist, dass das Ergebnis beider Verfahren sehr ähnlich klingt und die Verbesserung von 0,1 in MCD nicht notwendigerweise in einem hörbaren Unterschied resultiert. Die andere Hälfte teilen sich beide Verfahren nahezu gleich auf. Man kann daher festhalten, dass „mlpg“ zwar dazu in der Lage ist, ein besseres Syntheseergebnis zu erzielen, allerdings kann keine generelle und deutliche Verbesserung erreicht werden.

7. Zusammenfassung und Ausblick

Es hat sich gezeigt, dass die Einbringung von Kontextinformationen bei der Konvertierung von geflüsterter auf hörbare Sprache eine messbare Verbesserung bringt. Die Auswertung der Hörtests hingegen zeigt jedoch, dass die Verbesserung nur selten in einem besseren Klang resultiert. In der Hälfte der Fälle kann gar kein Unterschied festgestellt werden, sodass nicht von einer robusten bzw. generellen Verbesserung gesprochen werden kann. Von einem Einsatz des Verfahrens ist daher und aufgrund der Tatsache, dass der Algorithmus deutlich langsamer ist, abzusehen.
Abgesehen davon gibt es einige Beobachtungen, deren Untersuchung interessant sein kann: Die Konvertierung der einzelnen Sprecher reagiert teilweise unterschiedlich auf die verschiedenen Kontextinformationen. Des Weiteren stellt sich die Frage, ob die Verbesserung der MCD mit Erhöhung der Trainingsdaten und der Anzahl der Gaußkomponenten weiterhin skaliert. Aufgrund der Tatsache, dass Zielsequenzen geringere Wahrscheinlichkeiten als konvertierte Sequenzen erzielen, kann man experimentieren, ob das Trainieren der GMM mit anderen Kriterien die Konvertierung verbessern kann. Darüber hinaus sollte untersucht werden, ob die Erkenntnisse aus anderen Arbeiten (vgl. Kapitel 3) genutzt werden können, um das Verfahren zu verbessern.
Auf der Ebene der Implementierung besteht noch Optimierungspotenzial: Eine eigenständige Implementierung, die sich nur auf die notwendigen Komponenten und Algorithmen konzentriert, wäre effizienter und schneller, da durch die fehlenden Abhängigkeiten zu anderen Komponenten aggressivere Optimierungen möglich werden, die die Flexibilität verlieren. Dies gilt jedoch auch für den konventionellen Algorithmus. Außerdem könnte man die Eigenschaft ausnutzen, dass die Bandmatrix auch symmetrisch ist. Legt man sich auf ein bestimmte Variante und auf eine bestimmte Zusammensetzung der Kontextinformationen fest, könnte man diese auch hart implementieren.

A. Anhang: Durchführung von Experimenten

Das folgende Python-Skript „MapWhisper.py“ ist ein Beispielskript, welches für Experimente verwendet werden kann. Alle Konfigurationsvariablen sind in ein anderes Skript ausgelagert und dort mit Standardwerten belegt. In der Variable *speakerList* gibt man an, für welche Sprecher die Modelle trainiert und über die Variable *directory* kann man den Pfad angeben, in dem die Ergebnisse abgespeichert werden sollen. Dort werden dann einige Unterordner angelegt. Schließlich sollte man die Standardwerte der Konfigurationen mit denen überschreiben, mit denen man experimentieren möchte. In der `for`-Schleife wird schließlich für jeden Sprecher ein Modell mit den angegebenen Parametern trainiert und die Sätze aus dem in *useDevEval* angegebenen Datensatz werden konvertiert, deren MCD-Werte berechnet und schließlich synthetisiert.

```
import os
import sys
import Config_Whisper      # Default configuration.
# For default values and documentation options, see there.
import trainWhisperData  # Mapping script

if __name__ == '__main__':
    # Create a default configuration object
    config = Config_Whisper.Config_Whisper()

    # List of speakers for training and testing
    speakerList = [101,102,103,104,105,106,107,
        108,109,110,111,113]

    # Destination root folder for the results
    directory = "/home/example/myExperimentResults/"

    # Create the destination root, if it does not exist
    if not os.path.exists(directory):
        os.makedirs(directory)
    config.ResultRootTrainWhisperData=directory

    # Overwrite config variables from Config_Whisper.py
    # depending on whatever settings you want,
    # for example:
    config.numGaussians = 16
    # Specify the number of gaussians for Feature GMM
    config.numF0Gaussians = 8
    # Specify the number of gaussians for F0 GMM
    config.useDevEval = 'dev'
    # Specify 'dev' or 'eval' data for mapping
```

```
config.useMLPG = 1
# Specify if MLPG should be used or not
config.useVitebri = 1
# Specify if the component sequence method
# or the ML-estimator should be used

# Train models - here, config.speakers and
# config.testSpeakers are set to contain one
# entry from the speaker list for each loop
# iteration, so that speaker-dependent models
# are trained
for spk in speakerList:
    # Speaker-Dependent models
    # train separately for each speaker
    config.speakers = [spk]
    config.testSpeakers = [spk]

    # Perform training with given configuration
    train = trainWhisperData.WhisperMapping(config)
    train.mapWhisperData()
```

Literaturverzeichnis

[AMS08] Farzaneh Ahmadi, Ian Vince McLoughlin, and Hamid Reza Sharifzadeh. Analysis-by-synthesis method for whisper-speech reconstruction. In *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, pages 1280–1283. IEEE, 2008.

[DCK02] Alain De Cheveigné and Hideki Kawahara. Yin, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, 111:1917, 2002.

[dgb] http://www.netlib.org/lapack/explore-html/d3/d49/group_double_g_bsolve.html#gafa35ce1d7865b80563bbed6317050ad7. [Online; zugegriffen am 15-06-2013].

[DLR77] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[FTKI92] Toshiaki Fukada, Keiichi Tokuda, Takao Kobayashi, and Satoshi Imai. An adaptive algorithm for mel-cepstral analysis of speech. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, volume 1, pages 137–140. IEEE, 1992.

[HG12] Elina Helander and Moncef Gabbouj. Voice conversion (overview). *SPEECH ENHANCEMENT, MODELING AND RECOGNITION–ALGORITHMS AND APPLICATIONS*, page 69, 2012.

[HSVG12] Elina Helander, Hanna Silén, Tuomas Virtanen, and Moncef Gabbouj. Voice conversion using dynamic kernel partial least squares regression. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(3):806–817, 2012.

[ISF83] Satoshi Imai, Kazuo Sumita, and Chieko Furuichi. Mel log spectrum approximation (mlsa) filter for speech synthesis. *Electronics and Communications in Japan (Part I: Communications)*, 66(2):10–18, 1983.

[JWNS12] Matthias Janke, Michael Wand, Keigo Nakamura, and Tanja Schultz. Further investigations on emg-to-speech conversion. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 365–368. IEEE, 2012.

[KM98] Alexander Kain and Michael W Macon. Spectral voice conversion for text-to-speech synthesis. In *Acoustics, Speech and Signal Processing,*

1998. Proceedings of the 1998 IEEE International Conference on, volume 1, pages 285–288. IEEE, 1998.

[lap] http://www.netlib.org/lapack/. [Online; zugegriffen am 15-06-2013].

[NJWS11] Keigo Nakamura, Matthias Janke, Michael Wand, and Tanja Schultz. Estimation of fundamental frequency from surface electromyographic data: Emg-to-f0. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 573–576. IEEE, 2011.

[SCM98] Yannis Stylianou, Olivier Cappé, and Eric Moulines. Continuous probabilistic transform for voice conversion. *Speech and Audio Processing, IEEE Transactions on*, 6(2):131–142, 1998.

[SMA09] Hamid Reza Sharifzadeh, Ian Vince McLoughlin, and Farzaneh Ahamdi. Voiced speech from whispers for post-laryngectomised patients. *IAENG International Journal of Computer Science*, 36(4), 2009.

[SMA10] Hamid R Sharifzadeh, Ian Vince McLoughlin, and Farzaneh Ahmadi. Reconstruction of normal sounding speech for laryngectomy patients through a modified celp codec. *Biomedical Engineering, IEEE Transactions on*, 57(10):2448–2458, 2010.

[TBT07] Tomoki Toda, Alan W Black, and Keiichi Tokuda. Voice conversion based on maximum-likelihood estimation of spectral parameter trajectory. *Audio, Speech, and Language Processing, IEEE Transactions on*, 15(8):2222–2235, 2007.

[wik] http://commons.wikimedia.org/wiki/File:Illu01_head_neck.jpg. [Online; zugegriffen am 10-06-2013].

[WSJS13] Michael Wand, Christopher Schulte, Matthias Janke, and Tanja Schultz. Array based electromyographic silent speech interface. *Submitted to Biosignals*, 2013.