JAHRESARBEIT

SPRACHERKENNUNG UND IHRE REALISIERUNG (2004)

RAOUL MALM

GYMNASIUM AM KURFÜRTSLICHEN SCHLOSS MAINZ

Kurzfassung - Spracherkennung: Ihre Realisierung und Anwendung

Unser Programm „Vox-Mobil" ermöglicht die Steuerung eines Fahrzeuges durch die Sprache als natürliches Kommunikationsmittel. Der Benutzer verwendet ein Mikrofon, um dem Programm einzelne Worte (diskret gesprochene Steuerbefehle) mitzuteilen. Jedes einzelne Wort wird einer sprachlichen Analyse unterzogen, in der die für die Spracherkennung relevanten Informationen abstrahiert werden. Anhand dieser Informationen wird das Wort einem vorher definiertem Befehl zugeordnet oder es wird nicht erkannt. Wird ein Befehl erkannt, so wird dieser ausgeführt und das Fahrzeug bewegt sich entweder vorwärts, rückwärts, seitwärts nach links, seitwärts nach rechts oder es stoppt.

Zwei unipolare Schrittmotoren, die über die parallele Schnittstelle (Druckerport) angesteuert werden, dienen als Antrieb. Eine 2x4-Bit-Elektronik steuert mit Hilfe von Spannungsimpulsen die Motoren.

Mittels der Kepstralen Analyse erfolgt die Merkmalsgewinnung aus dem Sprachsignal. Der Vergleich der gewonnenen Merkmale mit den zuvor trainierten und gespeicherten Prototypen geschieht mit Hilfe der Dynamischen Zeitverzerrung. Übereinstimmungs-Wahrscheinlichkeiten entscheiden, ob der gesprochene Befehl erkannt und zugeordnet oder zurückgewiesen wird.

Darstellung des Arbeitsprozesses

Auf das Thema „Spracherkennung" bin ich durch einen Anruf bei der Fahrplanauskunft der Deutschen Bahn (Tel. 0800-1507090) gekommen. Ein natürlichsprachliches Dialogsystem hat mir Auskunft gegeben mit welchem Zug und zu welchen Zeiten und ich vom Bahnhof Mainz-Kastel aus nach Darmstadt fahren kann. Dieses Gespräch mit einem „menschlichen" Computer hat mich dazu angeregt die Automatische Spracherkennung als Thema für die Jahresarbeit zu verwenden. Dabei ist mir als praktisches Anwendungsbeispiel die Steuerung eines Modellfahrzeuges über die Sprache eingefallen.

Die Informationsfindung gestaltete sich zunächst als schwierig. Die Suche in Buchhandlungen und Bibliotheken ergab keine brauchbare Informationen. Erst im Internet habe ich viele Artikel und Facharbeiten gefunden, die das Thema der Automatischen Spracherkennung behandelt haben. Das Aussortieren von brauchbaren Informationen und das Umgehen von Widersprüchen in einigen Texten hat zu Anfang einige Zeit gekostet.

Als ich eine Vorstellung von der Funktionsweise der sprachlichen Erkennung diskret gesprochener Wörter hatte, begann ich die Anforderungen an mein Programm „Vox-Mobil" aufzuschreiben. Dabei musste ich berücksichtigen, ob und wie ich die Anforderungen mit der Programmiersprache C++ bewältigen könnte. Danach hielt ich auf Papier die Struktur und den Aufbau der Benutzeroberfläche des Programms fest.
Zuerst beschäftigte ich mich mit dem Kern des Programms, der Sprachlichen Analyse. Schritt für Schritt habe ich die einzelnen Analyseverfahren implementiert und auf ihre korrekte Funktionsweise getestet. Darauf folgte die Implementierung der Worterkennung. Nach einigen Tests und Optimierungen des Analyseverfahrens fügte ich die Trainingsphase und die Hardwaresteuerung hinzu. Speziell für die Hardwaresteuerung habe ich noch ein anderes Programm

entwickelt, mit dem man die einzelnen Pins eines Ports überwachen und setzen kann. Für das letzte Unterprogramm, dem eigentlichen Hauptprogramm, kombinierte ich die Sprachliche Analyse und die Hardwaresteuerung. Da sich jetzt die hohe benötigte Zeitdauer der rechenintensiven Fourier-Transformation bemerkbar machte, implementierte ich das schnellere Verfahren Fast-Fourier-Transformation.

A Inhaltsverzeichnis

Kurzfassung

Darstellung des Arbeitsprozesses

B Abbildungsverzeichnis

[*] : Mit dem Programm „Derive" wurde die Abbildung erstellt.
[**] : Die Abbildung stammt von meinem Programm „Vox-Mobil".
[***] : Mit Hilfe von MS-Paint wurde diese Abbildung erzeugt.

C Definitionsverzeichnis

Abtastrate:

Frequenzwert, der angibt wie viele Werte pro Zeiteinheit gemessen (abgetastet) werden

Diskrete Fouriertransformation (DFT):

ein Analyseverfahren, das eine komplexe Funktion mit Sinus- und Kosinusfunktionen bestimmter Frequenzen vergleicht, auf Übereinstimmung überprüft und schließlich die Frequenzen der komplexen Funktion enthält.

Dynamische Zeitverzerrung:

ein Verfahren, welches Elemente aus zwei Listen verschiedener Länge vergleichen kann und einen Abstandswert (Grad der Unterscheidung beider Listen) als Ergebnis liefert.

Entfaltung:

Trennung zweier miteinander gefalteter Funktionen über folgende Transformationen: DFT -> Logarithmierung -> IDCT

Fenster:

eine Liste mit diskreten Samplewerten, die einen kurzen Zeitbereich des Sprachsignals enthält (z.B. 25 ms)

Filterbank:

eine Dreiecksfunktion im Frequenzbereich, definiert über die linke, rechte und über die Mittenfrequenz (Maximum = 1)

Frequenzbereich:

ein Basisraum, bei dem die Werte des Definitionsbereiches Frequenzeinheiten entsprechen bspw. die Darstellung der Amplitudenwerte entlang der Abszissenachse (Frequenzachse).

Hochpassfilterung:

höhere Frequenzwerte werden angehoben bzw. tiefere Frequenzen werden abgeschwächt.

Inverse Diskrete Kosinus Transformation(IDCT):

eine verlustbehaftete Transformation vom Frequenzraum in den Ortsraum. Es werden stark unkorrelierte (unzusammenhängende) Koeffizienten erzeugt. Im Bereich der Sprachanalyse wird der Ortsraum Quefrenzraum genannt.

Lautheit:

subjektiv empfundene Lautstärke eines Tones

Longitudinalwelle:

eine Welle, deren Schwingungsvektoren der Oszillatoren gleich der Ausbreitungsrichtung sind. Beim Schall wird die Luft in periodischen Abständen verdichtet bzw. verdünnt, es entstehen Druckgefälle.

Mel-Frequenz-Kepstral Koeffizient (MFCC):

Koeffizienten, die Merkmale der Übertragungsfunktion des Vokaltraktes beinhalten. Man erhält durch folgende Transformationen:

DFT -> Mel-Filterbänke -> Logarithmierung -> IDCT

Prototyp:

Merkmalsliste eines Wortes, das die MFC-Koeffizienten und die Lautheitswerte enthält

Quefrenz-Bereich:

ein Basisraum, bei dem die Werte des Definitionsbereiches den Quefrenzeinheiten entsprechen bspw. die Darstellung der MFC-Koeffizienten entlang der Abszissenachse (Quefrenzachse).

Sample:

ein digitaler Wert, der in Abhängigkeit vom Schalldruck über die Soundkarte berechnet wird.

Spektralbereich:

Frequenzbereich

Zeitbereich:

ein Basisraum, bei dem die Werte des Definitionsbereiches Zeiteinheiten entsprechen bspw. die Darstellung der Samplewerte entlang der Abszissenachse (Zeitachse).

1. Einleitung

In den letzten Jahren hat die automatische Spracherkennung enorm an Bedeutung gewonnen. Durch die Steigerung der Rechenleistung integrierter Schaltkreise wurde es möglich frühere Eingabefelder von technischen Geräten durch natürlich-sprachliche Mensch-Maschine-Kommunikation zu ersetzen. Bereits bekannte Einsatzgebiete sind: Diktiersysteme in der Medizin und den Rechtswissenschaften, telefonische Auskunftssysteme (z.B. Deutsche Bahn), Voice-Banking (USA), Fremdsprachen-Lernsysteme, bequeme und arbeitserleichternde Steuerung von Alltagsgeräten (Lichtschalter, Mikrowelle, Waschmaschine) und Sprechererkennung durch phonetischen Fingerabdruck (siehe Anhang, D.1 Sprechererkennung durch phonetischen Fingerabdruck).
Der weitere Fortschritt in der Grundlagenforschung wird dazu beitragen, dass die Spracherkennung zukünftig in vielen Gebieten integriert werden kann. Dazu zählen beispielsweise die Rollstuhlsteuerung für Schwerbehinderte, eine erleichternde Steuerung von Produktionsabläufen in Unternehmen und der Ersatz von Tastatur, Maus und Stift als Eingabegerät für miniaturisierte mobile Computer, z.B. Mobiltelefone.

Die Automatische Spracherkennung stellt eine Innovation dar, die vor allem dazu dient, das alltägliche Leben zu erleichtern und zu vereinfachen.

Diese Umstände und mein eigenes Interesse an dem Thema haben mich motiviert die Funktionsweise der Spracherkennung zu ergründen, um daraus ein konkretes Anwendungsbeispiel von der Theorie in die Praxis umzusetzen.

2. Einführung in die Spracherkennung

Die Interaktion zwischen Mensch und Maschine wird zumeist mittels einer Tastatur und einer Maus realisiert. Diese Art von Informationseingabe zur Verarbeitung und Steuerung von Maschinen (z.B. PCs) ist leicht von Computern umzusetzen. Die natürlichste Form der Kommunikation ist jedoch die menschliche Sprache. Systeme, die die Spracheingabe ermöglichen, werden Automatische Spracherkennungssysteme (ASR-Systeme, Automatic Speech Recognition) genannt. Diese Systeme versuchen aus dem gegebenen akustischen Signal die wahrscheinlichste Wortsequenz zu bestimmen. Man unterscheidet zwei verschiedene Arten von Spracherkennungssystemen: diskrete Worterkennung und kontinuierliche Spracherkennung. Diskret gesprochene Sprache bedeutet, dass die Wörter einzeln und getrennt voneinander ausgesprochen werden. Im Fall der kontinuierlichen Spracherkennung sind fast alle Wörter lückenlos aneinandergereiht.
In dieser Arbeit wird die diskret gesprochene Sprache verwendet, da sie zur Steuerung eines Fahrzeuges ausreicht.

2.1 Spracherzeugung

Sprachlaute sind Schallsignale, die mit Hilfe der in der Abb. 1 dargestellten Sprachorgane gebildet werden (Quelle: [ZI97]). Durch Erhöhen bzw. Vermindern des Luftdruckes in der Lunge entsteht eine Luftströmung, wodurch die Stimmbänder zum Schwingen angeregt werden. Dadurch schließt bzw. öffnet sich die Stimmritze (Glottis). Wird die Stimmritze durch die Stimmbänder verengt, entsteht eine periodische Anregung

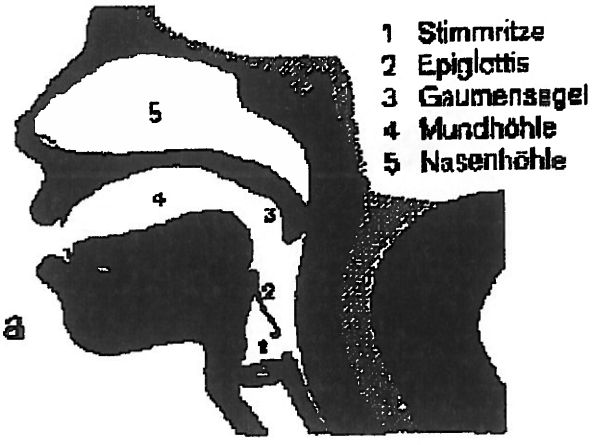

Abb. 1: Die menschlichen Sprachorgane

des nachfolgenden Resonanzraumes (Vokaltrakt), die je nach dessen Ausformung (Einengung, Erweiterung) zu einem der stimmhaften Laute z.B. der Vokale a, e ,i, o und u führt. Ist die Stimmritze jedoch weiter geöffnet, entstehen stärkere Luftturbulenzen im Vokaltrakt und es kommt zu rauschähnlichen Signalen, die zu stimmlosen Lauten wie, z.B. dem h führen. Das Sprachsignal lässt sich als „Faltung" aus dem Anregungssignal $v(t)$ der Stimmbänder mit der Impulsantwort $h(t)$ der zusammengefassten Übertragungsglieder, dem sog. Vokaltrakt, darstellen.

Mathematisch lässt sich dieser Zusammenhang folgendermaßen ausdrücken:

$$s(t) = v(t) \otimes h(t) = \int_{-\infty}^{+\infty} v(\xi) * h(t-\xi)\, d\xi$$

$h(t)$ entspricht der Impulsantwort und

$v(t)$ entspricht dem Anregungssignal

Das Modell in Abb. 2 verdeutlicht nochmals die Entstehung eines Sprachsignals.

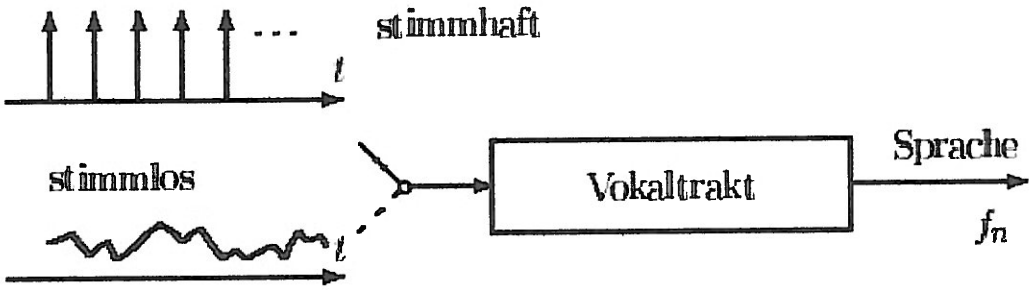

Abb. 2: Modell der Sprachgenerierung

Beispiel einer Faltung aus dem Buch [Bu03]:

Die Rechteckfunktion $f(t)$ wird mit einer anderen Rechteckfunktion $g(t)$ gefaltet:

$$u(t) = f(t) \otimes g(t) = \int_{-\infty}^{+\infty} f(\xi) * g(t-\xi) d\xi$$

$$\text{mit } f(t) = \begin{cases} 1 & -T/2 \le t \le +T/2 \\ 0 & \text{sonst} \end{cases}$$

$$\text{und } g(t) = \begin{cases} 1 & 0 \le t \le T \\ 0 & \text{sonst} \end{cases}$$

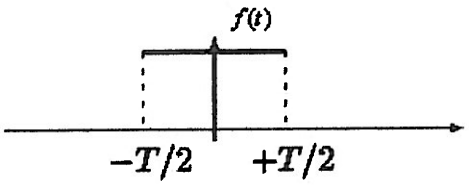

Abb. 3: Rechteckfunktion $f(t)$

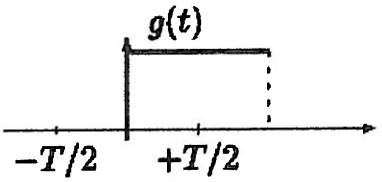

Abb. 4: Rechteckfunktion $g(t)$

Dabei wird die Funktion $g(t)$ nach rechts entlang der X-Achse verschoben und für jeden Schritt t das Integral über dem Produkt beider Rechteckfunktionen berechnet (siehe Abb. 5). An den Grenzen $t = -T/2$ und $t = 3T/2$ wird mit dem Überlapp 0 gestartet bzw. geendet. Der maximale Überlapp (100%) befindet sich bei $t = T/2$, wenn die beiden Rechtecke genau übereinander liegen.

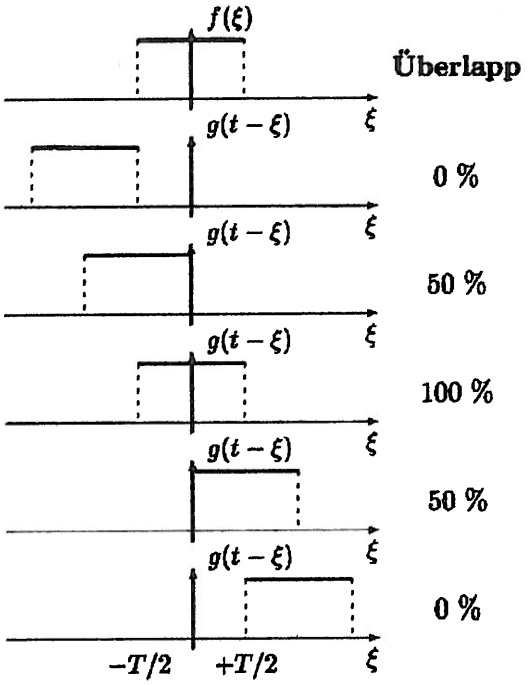

Abb. 5: Verschiebung der Funktion $g(t-\xi)$ nach rechts

Das Ergebnis der Faltung ist die Dreiecksfunktion $u(t)$, Abb. 6.

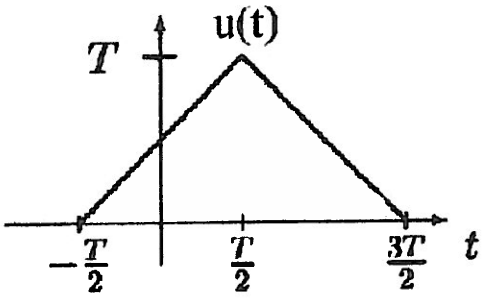

Abb. 6: Dreiecksfunktion $u(t)$

2.2 Probleme der maschinellen Spracherkennung

Variabilität:

Laute und Wörter, die ein Mensch als gleich identifizieren würde, sind nie identisch, sondern weisen immer akustische Abweichungen auf. Dies liegt zum einen an der unterschiedlichen Umgebung (Störgeräusche), den individuellen Eigenschaften des Sprechers (männlich/weiblich, alt/jung, psychische Befindlichkeit), der unterschiedlichen Sprechweise (Tempo, Betonung) und dem Abstand zum Mikrofon.

Komplexität:

Die maschinelle Spracherkennung erfordert hohe Rechenleistung und Speicherkapazität. Bei einer Abtastrate, auch Samplerate genannt, von beispielsweise 11025 Hertz sind für eine Sprachaufnahme von einer Sekunde bereits 11025 Samplewerte zu speichern und zu verarbeiten.

2.3 Merkmalsgewinnung

Da das Sprachsignal niemals für das gleiche Wort identisch ist, müssen Informationen herausgearbeitet werden, die zu einer eindeutigen Identifizierung des Sprachsignals führen. Dazu wird die Kepstrale Analyse verwendet, die als Resultat Mel-Frequenz-Kepstral Koeffizienten (MFCC, mel-frequency cepstral

coefficiants) liefert. Bei deren Berechnung versucht man auf den Einfluss des Vokaltraktes im Sprachsignal zu schließen. Dieser enthält für jeden Laut charakteristische Informationen, z.B. kann die Stimmhaftigkeit eines Vokals anhand dieser Merkmale aufgezeigt werden. Als weiteres Merkmal wird die Lautheit d.h. die Energie eines Sprachsignals verwendet. Zusammen bilden die MFC- Koeffizienten und die Energiewerte eines Sprachsignals eine Liste von Merkmalsvektoren, welche Prototyp genannt wird. Diesen Prototyp verwende ich zum Vergleich und zur Identifizierung von Sprachsignalen.

Erläuterungen zur Kepstralen Analyse [We01]:

Die Kepstrale Analyse geht von der Faltung des Anregungssignals $v(t)$ und der Impulsantwort $h(t)$ aus, die gemeinsam das Sprachsignal $s(t)$ bilden:

$$s(t) = v(t) \otimes h(t)$$

Das Ziel der Kepstralen Analyse besteht darin das Anregungssignal $v(t)$ von der Impulsantwort $h(t)$ zu trennen (Entfaltung des Sprachsignals). Für die Spracherkennung interessiert uns hier die Impulsantwort, da sie für die Ausformung von Lauten verantwortlich ist. Da die Faltung eine komplexe mathematische Funktion darstellt, wird das Sprachsignal $s(t)$ in den Spektralbereich mittels der Fourier-Analyse (FT) transformiert. Somit lässt sich der Faltungsoperator in eine Multiplikation überführen.

$$FT\{s(t)\} = FT\{v(t)\} * FT\{h(t)\}$$

Anschließend wird das Spektrum logarithmiert, um die Multiplikation in eine Addition zu überführen:

$$\log\{FT\{s(t)\}\} = \log\{FT\{v(t)\}\} + \log\{FT\{h(t)\}\}$$

Als nächster Schritt wird die Inverse Diskrete Kosinus-Transformation (IDCT) angewandt und man erhält Mel-Frequenz-Kepstral Koeffizienten. Lässt man jetzt die Koeffizienten höherer Ordnung weg, so hat das die Separierung der Anteile der Anregungsfrequenz zur Folge. Verwendet man also nur die ersten 15 Koeffizienten von beliebig vielen möglichen, dann hat man sich des Anregungssignals entledigt.

Daraus ergibt sich:

Meine Vorgehensweise für die Gewinnung der Merkmalsvektoren:

(1) Preemphase: Hochpassfilterung im Zeitbereich
(2) Fensterung: das Sprachsignal wird in Fenster unterteilt
(3) DFT: Transformation in den Frequenzbereich
(4) Mel-Filterbänke: Gewichtung einzelner Frequenzbereiche
(5) Logarithmierung und Inverse Diskrete Kosinus-Transformation
(6) Hinzufügen der Lautheit zu den Merkmalen

Spricht eine Person entstehen Longitudinalschwingungen, die von einem Mikrofon in Spannungs- und Stromschwankungen umgewandelt werden. Das entstandene analoge Signal wird nun an die Soundkarte eines Rechner weitergeleitet und dort mittels eines Analog-Digitalwandlers in eine digitale Form gebracht. Die digitale Form besteht aus sog. Samplewerten. Die Abtastrate der Soundkarte legt dabei fest, wie viele Samples pro Zeiteinheit aus dem Sprachsignal berechnet werden. Übliche Abtastraten liegen zwischen 8000 Hz und 16000 Hz.

2.3.1 Preemphase

In der Preemphase wird das Sprachsignal frequenzabhängig verstärkt, d.h. die Amplitudenwerte bestimmter Frequenzen werden angehoben bzw. abgeschwächt (Quelle: [Wi00]). Dieser Schritt ist erforderlich, da die Stimmritze (Glottis) das Sprachsignal um 12dB dämpft und die Lippen es wiederum um 6dB hochpassfiltern. Daher muss ein Hochpassfilter eingesetzt werden, der die höheren Frequenzanteile um 6dB anhebt. Dafür wird ein nicht rekursiver Filter (FIR, Finite Impulse Response), der im Zeitbereich arbeitet, verwendet. Dies geschieht durch die Bildung der Ableitung des Sprachsignals unter Einfluss einer Gewichtung *a*:

$$s_{pre(n)} = s(n) - a * s(n-1)$$

Leitet man eine Wellenfunktion ab, werden die Amplituden der größeren Frequenzen verstärkt und die der kleineren Frequenzen abgeschwächt. Mit der Gewichtung *a* kann man darauf Einfluss nehmen, wie stark die Verstärkung bzw. Abschwächung abläuft.
In meinem Programm beträgt a = 0.95.

2.3.2 Fensterung und Fensterfunktionen

Bei der Fensterung (Quelle: [Bu03]) wird das Sprachsignal in gleich lange Zeitbereiche unterteilt, z.B. 25 ms. Jeder einzelne Zeitbereich besteht aus einer bestimmten Anzahl von Samplewerten und wird Fenster genannt. Es wird angenommen, dass das Sprachsignal in jedem dieser Fenster unverändert ist und somit eine Serie von Einzelbeobachtungen darstellt.

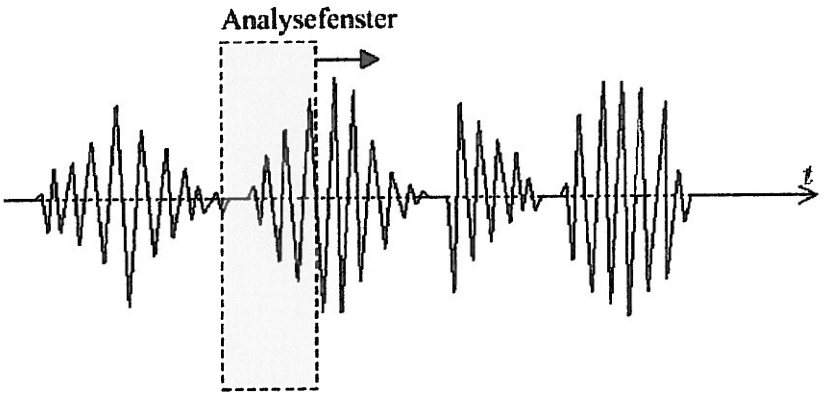

Abb. 7: Das Sprachsignal wird gefenstert

Die Einteilung in Fenster entspricht der Leistung des menschlichen Ohrs, das innerhalb kleiner Zeitintervalle keine akustischen Veränderungen wahrnehmen kann (ca. 10-20 ms). Daher kann man das akustische Signal in diesem Intervall als einen einzigen Laut auffassen. Da aber die entscheidende Äußerung auch am Anfang oder Ende eines Fensters liegen kann, kann eine Verzerrung auftreten. Dem wird vorgebeugt, indem sich die Fenster überlappen. Die Fensterung wird benötigt, da das Sprachsignal nur in

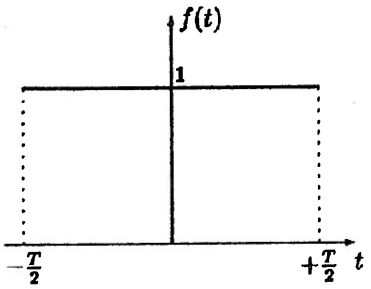

Abb. 8: Rechteckfunktion $f(t)$

kleinen Zeitbereichen als periodische Schwingung aufgefasst werden kann. Erst durch diesen Schritt lassen sich die Frequenzen berechnen. Da die oben genannten Fenster zu der Kategorie Rechteck-Fenster gehören, entstehen Sprungstellen an deren Kanten, die bei der anschließenden Transformation in den Spektralbereich sehr hohe Frequenzen bewirken und das Sprachsignal verfälschen. Aus diesem Grund werden so genannte Fensterfunktionen über jedes Fenster gelegt. Das heißt, jeder Samplewert innerhalb eines Fensters wird mit dem Funktionswert der Fensterfunktion an der entsprechenden Stelle multipliziert. Die Fensterfunktionen werden so gewählt, dass die Samplewerte an den Rändern abgeschwächt und somit Sprungstellen verhindert werden. In meinem Programm kann man zwischen einem „Rechteckfenster" (Abb. 8), einem „Hamming-Fenster" (Abb. 9) oder einem „Kaiser-Bessel-Fenster" (Abb. 10) wählen.

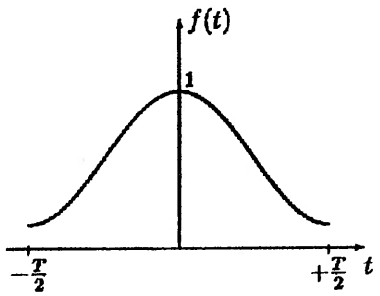

Abb. 9: Hammingfunktion

$f(t) = 0.54 - 0.46 * \cos(2\pi t/T)$

(Gaußglockenkurve)

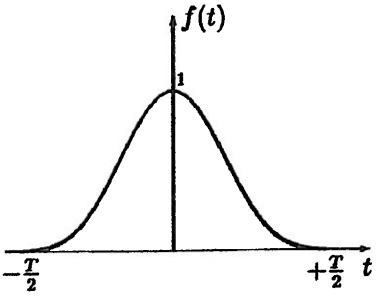

Abb. 10: Kaiser-Bessel-Funktion

2.3.3 Diskrete Fourier-Transformation (DFT)

Zu Beginn befindet sich das Sprachsignal im Zeitbereich und pro Zeiteinheit liegt ein Samplewert (digitalisierte Schwingungsamplitude) vor. Diese Darstellung ist viel zu komplex, als dass man daraus aussagekräftige Informationen ermitteln kann. Wesentlich deutlicher und charakteristischer ist die Darstellung der Sprache im Frequenzbereich.

Daher wird über jedes Fenster die DFT angewandt, um die enthaltenen Frequenzen zu berechnen. Dazu wird sich folgendes Prinzip von J.B. Fourier zu Nutze gemacht (Quelle: [Bu03]): Jede periodische Funktion $f(t)$ lässt sich als Überlagerung von Sinus- und Kosinusfunktionen geeigneter Amplituden und Frequenzen darstellen:

$$f(t) = \sum_{k=0}^{\infty} (a_k * \cos(\omega_k t) + b_k * \sin(\omega_k t)) \qquad \text{(Gl. 2.1)}$$

$$\text{mit } \omega_k = \frac{2\pi k}{T} \quad \text{und} \quad b_0 = 0$$

Die Fourierkoeffizienten a_k und b_k geben die Anteile der Kosinus- und Sinusfunktionen bestimmter Frequenzen, die Vielfache der Grundfrequenz $\omega_1 = 2\pi/T$ darstellen, an.

Damit ergibt sich folgende Vorgehensweise für die DFT: Mathematisch konstruierte Sinus- und Kosinusschwingungen mit der Frequenz $1/T$ (T = Zeitdauer eines Fensters) und mit der zweifachen, dreifachen, vierfachen usw., allgemein k-fachen Frequenz dieser Grundschwingung werden mit dem Sprachsignal verglichen, wie sehr sie ihm ähnlich sind.

Wenn man die Gleichung 2.1 nach den Fourierkoeffizienten a_k und b_k auflöst, erhält man:

$$a_k = \frac{2}{T} \int_{-T/2}^{+T/2} f(t) * \cos(\omega_k t)\, dt \qquad \text{für } k \neq 0$$

$$\text{und } a_0 = \frac{1}{T} \int_{-T/2}^{+T/2} f(t)\, dt$$

$$b_k = \frac{2}{T} \int_{-T/2}^{+T/2} f(t) * \sin(\omega_k t)\, dt \qquad \text{für alle } k$$

Die Herleitung ist im Anhang ausführlich beschrieben. Der Fourierkoeffizient a_0 gibt den Mittelwert der Funktion an. Für b_0 gilt immer: $b_0 = 0$.

Hierbei ist zu beachten, dass man für negative k auch Fourierkoeffizienten für negative Frequenzen erhält. Diese entsprechen jedoch betragsmäßig den Fourierkoeffizienten an den entsprechenden positiven Indizes. Dies ist von Bedeutung, wenn man von der Kontinuierlichen- in die Diskrete

Fouriertransformation übergeht, da die Koeffizienten der DFT an keinen negativen Frequenzen stehen. Die negativen Indizes werden an das rechte Intervallende herumgeklappt, sodass die Fourierkoeffizienten für $k > N/2$ (N = Anzahl der Samples) die gleichen Frequenzen liefern wie für $k \leq N/2$, nur spiegelverkehrt. Daher lasse ich k nur von 0 bis $N/2$ laufen und die DFT sieht somit folgendermaßen aus:

$$a_k = \frac{2}{N}\sum_{n=0}^{N-1} f_n * \cos(\frac{2\pi kn}{N}) \qquad \text{für } k = 1,2....N/2$$

$$a_0 = \frac{1}{N}\sum_{n=0}^{N-1} f_n$$

$$b_k = \frac{2}{N}\sum_{n=0}^{N-1} f_n * \sin(\frac{2\pi kn}{N}) \qquad \text{für } k = 0,1....N/2$$

Die Listen a_k und b_k geben die Anteile der geraden Funktionen (Kosinus-Anteile, da $\cos(-t) = \cos(t)$) und b_k die der ungeraden Funktionen (Sinus-Anteile, da $\sin(-t) = -\sin(t)$) für best. Frequenzen an. Hieran erkennt man schon die Analogie zu den komplexen Zahlen. Die Koeffizienten a_k kann man als den realen Anteil und b_k als den imaginären Anteil ansehen. Gemeinsam bilden sie die komplexe Zahl $F_k = a_k + ib_k$ (siehe Abb. 11). In der Polardarstellung lässt sich F_k auch durch den Betrag und die Phase (Winkel α) darstellen: $F_k = |F_k| * e^{i\alpha}$. Die Phase ist für die Frequenzbestimmung nicht relevant, da sie nur Informationen über die Verteilung der Kosinus- und Sinusanteile enthält. Daher wird das Betragsspektrum (Amplitudenspektrum) A_k berechnet:

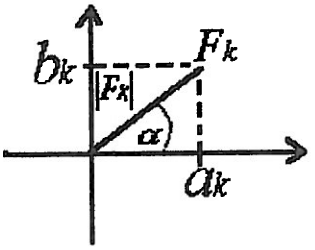

Abb. 11: komplexe Zahl F_k

$$A_k = \sqrt{(a_k)^2 + (b_k)^2} \qquad k = 1,2....,N/2$$

Die Werte des Amplitudenspektrums A_k geben an, ob sich die Frequenzen $f_k = k * f_g$ im Sprachsignal befinden. Dabei steht f_g für die Grundfrequenz, die sich aus dem Quotient der Abtastrate f_s und der Anzahl der Samplewerte N berechnen lässt, kurz: $f_g = f_s/N$. Dies entspricht der Auflösungsfrequenz, die angibt mit welcher Genauigkeit die Frequenzen des Sprachsignals erfasst werden. Erinnert man sich daran, dass man k nur von 0 nach $N/2$ laufen lässt, wird ersichtlich, dass die größte darzustellende Frequenz die Hälfte der Abtastrate beträgt. Dies entspricht dem Samplingtheorem, das besagt: **Die Abtastrate muss mindestens doppelt so hoch sein, wie die höchste darzustellende Frequenz, also** $f_s > 2 * f_{max}$. Man muss immer mehr als zwei Samples pro Periode einer Funktion nehmen, um diese vollständig darstellen zu können.

Man erkennt leicht, dass die Auflösung des Frequenzbereiches von der Anzahl der Samplewerte abhängt. Trotzdem kann man N nicht erhöhen, sonst würden die einzelnen Fenster eine zu große Zeitspanne einnehmen und das Sprachsignal wäre nicht mehr periodisch und stationär. Eine Lösung besteht darin das Sprachsignal am Ende mit vielen Nullen aufzufüllen (auch genannt: „zero-padding") und dann die Frequenzen zu bestimmen, da die hinzugefügten Nullen die Anzahl der Samplewerte erhöhen und die Frequenzen nicht beeinflusst werden. In der praktischen Umsetzung mit einem Rechner sind dem aber Grenzen gesetzt, da die DFT sehr rechenintensiv ist. Für die Berechnung der Koeffizienten sind allein N^2 Multiplikationen nötig, also ein Anstieg um die Potenz zwei in Abhängigkeit von der Anzahl N der Samplewerte.
Im Jahre 1965 haben Cooley und J.W. Tukey das DFT-Verfahren optimiert, das den Namen FFT (Fast Fourier-Transformation) trägt. Der Algorithmus, genannt „Decimation in time", basiert auf der Zerlegung des Sprachsignals in gerade und ungerade Teilmengen. Die Halbierungen werden so oft durchgeführt bis am Ende Teilmengen übrigbleiben, die aus jeweils einem Element bestehen. Daher ist es erforderlich, dass die Gesamtanzahl der Elemente auf der Basis zwei basiert:
$2^p = N$. Die Fouriertransformierte F_0 dieses einen Elementes ist gleich dem Funktionswert f_0, denn es gilt: $F_0 = f_0$. Die einzelnen Fouriertransformierten pro

Teilmenge kann man zusammenfügen, sodass sich letztendlich die Fourierkoeffizienten des Sprachsignals berechnen lassen. Mit Hilfe der FFT kann man den Rechenaufwand auf $p * N = ld(N) * N$ Multiplikationen verkleinern.
Den Quellcode des Verfahrens habe ich von [EiCo00] verwendet und ihn an meine Bedürfnisse angepasst.
Werden die oben genannten Berechnungen auf jedes Fenster angewandt, befindet sich das Sprachsignal im Spektralbereich, d.h. es gibt für jedes Fenster eine Liste mit den berechneten Amplitudenwerten A_k.

2.3.4 Mel-Filterbänke

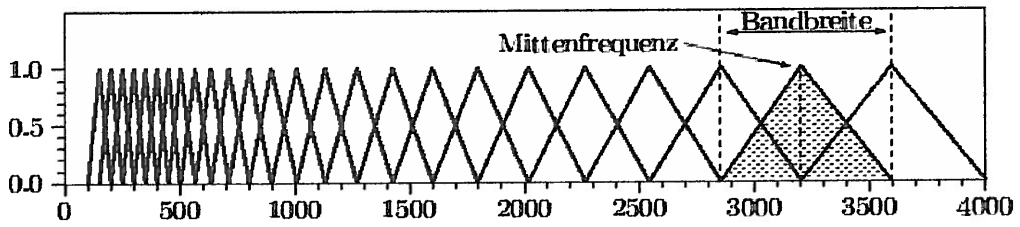

Abb. 12: überlappende Filterbänke im Frequenzbereich von 0 bis 4000 Hz

In diesem Schritt wird die Gewichtung einzelner Frequenzbereiche pro Fenster durchgeführt, da sich Laute im tiefen Frequenzspektrum (unter 1kHz) wesentlich mehr unterscheiden als im hohen Bereich (Quelle: [UEN03]). Dies ist der menschlichen Tonhöhenwahrnehmung nachempfunden, da das Ohr für niedrige Frequenzen empfindlicher ist. Realisiert wird diese Gewichtung durch Filterbänke (entsprechen den Dreiecksfunktionen), die jeweils einen bestimmten Frequenzbereich abdecken. Die Filterbänke werden so angeordnet, dass Frequenzbereiche unter 1 kHz linear abgedeckt werden. Außerdem wird dazu der größte Teil der Filterbänke verwendet. Oberhalb von 1 kHz wird eine logarithmische Aufteilung verwendet, d.h. die Bandbreite wird stetig vergrößert. Für eine optimale Gewichtung werden die Filterbänke überlappend angeordnet. Nun wird über jede Filterbank eine Dreiecksfunktion gelegt, die ihr Maximum (den Wert 1) an der Mittenfrequenz hat und zur linken sowie zur rechten Grenze auf Null abfällt. Für mein Programm wird eine entsprechende Aufteilung verwendet.

Anschließend werden das Leistungsspektrum für jede Filterbank mit der zugehörigen Dreiecksfunktion $d_m(f_k)$ multipliziert und diese Faktoren addiert. Das Leistungsspektrum berechnet man, indem man die Werte A_k des Betragsspektrums quadriert. Es ergibt sich also ein einziger Wert fb_m pro Filterbank und dieser wird als Energiewert bezeichnet:

$$fb_m = \sum_{k=1}^{N/2} dm(f_k) * A_k^2 \qquad m = 1,2,\ldots, M$$

M = Anzahl aller Filterbänke je Fenster

$d_m(f_k)$ = Dreiecksfunktionswert der m-ten Filterbank an der Frequenzstellestelle f_k

fb_m = Energiewert der Filterbank an der Stelle m

A_k^2 = Werte des Leistungsspektrums

Damit eine Dreiecksfunktion definiert werden kann, benötigt man ihre Mittenfrequenz mf_m und ihre linke und rechte Frequenzgrenze $\lg_m$ bzw. rg_m. Um nun die Funktionswerte zu erhalten, muss die Dreiecksfunktion in die vier Intervalle $(-\infty; \lg_m)$, $[\lg_m; mf_m]$, $(mf_m; rg_m]$ und $(rg_m; +\infty)$ aufgeteilt werden. Die Formel für $d_m(f_k)$ lautet dann folgendermaßen:

$$dm(fk) = \begin{cases} 1 - \dfrac{(mf_m - f_k)}{(mf_m - \lg_m)} & \lg_m \le f_k \le mf_m \\ \dfrac{(rg_m - f_k)}{(rg_m - mf_m)} & mf_m < f_k \le rg_m \\ 0 & f_k < \lg_m \text{ oder } f_k > rg_m \end{cases} \qquad \text{(Gl. 2.2)}$$

Verwendet man beispielsweise 25 Filterbänke für die Abdeckung des gesamten Frequenzbereiches so erhält man 25 Energiewerte.
Ein weiterer Vorteil der Verwendung der Filterbänke ist somit auch die Verringerung der Frequenzdaten pro Fenster von z.B. 256- auf 25 Werte.

2.3.5 Logarithmierung und Inverse Diskrete Kosinus-Transformation (IDCT)

Durch eine Logarithmierung der Energiewerte eines jeden Fensters entsteht eine Kompression, wodurch die Merkmalsgewinnung weniger abhängig von absoluten Schwankungen gemacht wird, denn der Logarithmus ist wesentlich kleiner als die einzelnen Energiewerte. Dieser Schritt berücksichtigt zudem die subjektive Lautstärkeempfindung des menschlichen Gehörs. Um nun in den Quefrenz-Bereich zu gelangen, verwendet man die IDCT auf die logarithmierten Energiewerte eines jeden Fenster an. Die IDCT ist folgendermaßen definiert (Quelle: [Wi00]):

$$Y(u) = \sum_{m=1}^{M} \log_2(fb_m) * \cos(\frac{u*(m+2)*\pi}{2M}) \quad \text{für } 1 \le u \le 13$$

M = Anzahl der Filterbänke je Fenster

fb_m = Energiewert von der Filterbank an der Stelle m

$Y(u)$ = MFC-Koeffizient

Als Resultat dieser Berechnung erhält man Mel-Frequenz-Kepstral Koeffizienten („Kepstrum", Kunstwort aus „Spektrum"). Die Indizes befinden sich im „Quefrenz"-Bereich („Quefrenz", Kunstwort aus „Frequenz"). Die Koeffizienten niederer Ordnung , z.B. von 1 bis 13 repräsentieren den Einfluss des Vokaltraktes auf das Sprachsignal und werden in einer Liste gespeichert.

2.3.6 Hinzufügen der Lautheit

Die Lautheit ist ein Maß für die Energie (Intensität) eines Sprachsignals. Mit Hilfe dieser Intensitätswerte erhält man verbesserte Erkennungsraten für Wörter, die unterschiedlich laut ausgesprochen werden. Für jedes Fenster des Sprachsignals

wird die Lautheit berechnet, indem man die logarithmierten Energiewerte der Filterbänke aufsummiert (Quelle: [UEN03]):

$$L = \sum_{m=1}^{M} \log_2(fb_m)$$

M = Anzahl der Filterbänke je Fenster

fb_m = Energiewert an der Stelle m in der Filterbank

Das Hinzufügen der Lautheit zu den Merkmalsvektoren hat auch in meiner Arbeit eine verbesserte Erkennungsrate mit sich gebracht.

2.3.7 Prototyp

Die Mel-Frequenz-Kepstral Koeffizienten und die Energie des Sprachsignals eines Fensters werden in einer Liste, Prototyp genannt, zusammengefasst. Die Größe eines Prototypen hängt von der Anzahl der Fenster und somit von der Länge des Sprachsignals ab. Ein Prototyp beinhaltet die Merkmale eines Sprachsignals und wird für die Worterkennung verwendet.

2.4 Vergleich zweier Prototypen mittels der Dynamischen Zeitverzerrung

Da die Größe der Prototypen von der Länge des Sprachsignals abhängt, benötigt man ein Verfahren zum Vergleich zweier Listen von unterschiedlicher Länge. Ein mögliches Verfahren bietet die Dynamische Zeitverzerrung. Dies wird wie folgt realisiert: Man stellt sich eine Zwei-Dimensionales Feld vor. Auf der unteren Achse, der X-Achse, wird die Liste des ersten Prototypen aufgetragen. Auf der linken Achse, der Y-

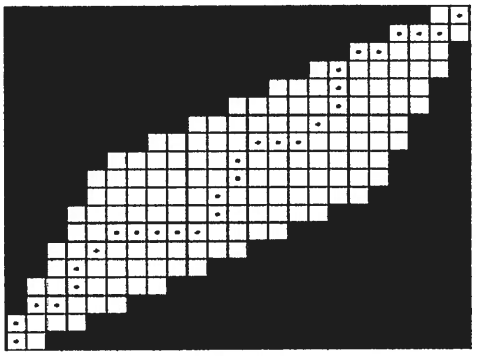

Abb. 13: Abstandsfeld – optimaler Pfad wird gesucht

Achse, geschieht dies mit der Liste des zweiten Prototypen. Nun wird in jeder Zelle der Abstand zwischen dem entsprechenden Wert auf der X-Achse und dem auf der Y-Achse berechnet (durch eine Subtraktion). Anschließend wird der Pfad gesucht, der unten links startet, oben rechts endet und in der Summe möglichst wenig Abstände in Kauf nehmen muss. Die Regeln dazu sind, dass von einer Zelle aus immer nur eine Zelle nach rechts, nach oben oder diagonal nach rechts oben gesprungen werden darf. Der Gültigkeitsbereich, den der Pfad nehmen darf, kann eingeschränkt werden, um an Effizienz zu gewinnen (siehe Abb. 13). In unserem Programm wird ein Parallelogramm mit Steigungen von ½ bzw. 2 verwendet.

2.5 Worterkennung

Mittels der Dynamischen Zeitverzerrung wird der Prototyp des zu erkennenden Wortes mit den zuvor trainierten und gespeicherten Prototypen der Befehlswörter verglichen und die Abstände berechnet. Wir haben uns dazu entschlossen, zuerst nach dem nächsten Nachbarn d.h. den Prototyp mit dem geringsten Abstand zum Prototyp des zu erkennenden Wortes zu finden. Falls jedoch der Abstand des ermittelten Prototypen eine bestimmte Grenze z.B. 6 überschreitet, wird kein Wort erkannt. Unterschreitet der Prototyp diese Grenze, wird der gemittelte Abstand der Prototypen, die zu seiner Kategorie (bestimmter Befehl) gehören, berechnet. Aus dem gemittelten Abstand wird die prozentuale Wahrscheinlichkeit berechnet, inwieweit das zu erkennende Wort dem vermuteten Befehl entspricht. Überschreitet der Prozentwert eine bestimmte Grenze, z.B. 30 % wird das Wort erkannt und dem vermuteten Befehl zugeordnet. Beide Grenzen (Einzelabstand, prozentuale Wahrscheinlichkeit) können in meinem Programm verändert und angepasst werden.

3. Softwaremäßige Umsetzung der Theorie aus Abschnitt 2 in unserem Programm

In diesem Abschnitt werden die einzelnen Stufen der Merkmalsgewinnung und die anschließende Worterkennung anhand unseres Programms graphisch dargestellt und mit Quellcode erläutert. Die Dezimalklassifikation bezieht sich auf Kapitel 2.

3.3.1 Preemphase

Die Preemphase verläuft im Hintergrund der Programmoberfläche und wird somit nicht grafisch dargestellt. Folgender Quellcode in C++ führt die Preemphase durch:

```
void CASR::Preemphase_Calculation(short *SampleBuffer,  int BufferLength,
                                                  short *PreemphaseBuffer)
{
/*
 *  Methode: Preemphase_Calculation
 *  übergebene Parameter:
                EINGABE:
                        short * SampleBuffer  = Zeiger auf den Buffer für die Samples des
                                Sprachsignals (eine Liste mit Elementen des Typs short: -32768 bis
                                32767)
                        int BufferLength        = Anzahl der besetzten Elemente im SampleBuffer
                AUSGABE:
                        short *PreemphaseBuffer = Zeiger auf den Buffer für die hochpassgefilterten
                                Samplewerte (eine Liste mit Elementen des Typs short)
*/

        float a=0.95F; //Gewichtung a
        PreemphaseBuffer[0]=SampleBuffer[0]; //das 0-te Element wird übernommen

        for(int i=1;i<BufferLength;i++)
                {
```

```
                    //Ableitung unter Einfluss der Gewichtung a
                    PreemphaseBuffer[i] = (short)((float)SampleBuffer[i] - a *
                                                (float)SampleBuffer[i-1]);
            }//Ende der For-Schleife
} //Ende der Methode Preemphase_Calculation
```

3.3.2 Fensterung und Fensterfunktionen

Die Fensterung wird im nächsten Abschnitt 3.3.3 in Verbindung mit der Diskreten-Fourier-Tranformation gezeigt. Für jede Fensterfunktion gibt es eine eigene Quellcode-Funktion. Im nachfolgenden Quellcode ist die Funktion des Hamming-Fensters, das auf das Sprachsignal angewandt wird, dargestellt:

```
void CASR::HammingWindow(short *SampleBuffer, int BufferLength)
{
/*
 * Methode: HammingWindow
 * Übergabeparameter:
        EINGABE:
                int BufferLength        =  Anzahl der besetzten Elemente im SampleBuffer
        EINGABE/AUSGABE;
                short * SampleBuffer    =  Zeiger auf den Buffer für die Samples eines einzelnen
                        Fensters (eine Liste mit Elementen des Typs short) -> call by reference
 * Funktion: Die übergebenen Samplewerte warden mit der Gaußglockenkurve gewichtet.
*/

        float fig = 2.PI / (BufferLength-1);        //PI = 3,141592654
        for(int i=0;i<BufferLength;i++) //Schleife für alle Samplewerte
                SampleBuffer[i]=(short)(SampleBuffer[i]* (0.54 - 0.46 * cos( (float)i*fig)));
}//Ende der Methode HammingWindow
```

3.3.3 Diskrete Fourier-Transformation (DFT)

Um die Vorgehensweise bei der Transformation in den Frequenzbereich anhand des Quellcodes leichter verstehen zu können, wird die folgende Methode dargestellt, die nicht den FFT- sondern den DFT-Algorithmus verwendet:

```
void CASR::DFT_Calculation(short* SampleBuffer, int SampleBufferLength, unsigned
                                    __int16 *DFTBuffer, int* DFTBufferLength)
{
/*
 * Methode:DFT_Calculation
 * Übergabeparameter:
        EINGABE:
                short * SampleBuffer          =     Zeiger auf den Buffer mit den Samplewerten
                int SampleBufferLength        =     Anzahl der besetzten Elemente des
                                                    SampleBuffers
        AUSGABE:
                unsigned __int16 *DFTBuffer   =     Zeiger auf den Buffer für die Frequenzwerte
                        (eine Liste mit positiven Zahlen: 0 bis 65536)
                int * DFTBufferLength         =     Anzahl der besetzten Elemente des
                        DFTBuffers -> call by reference
* Funktion: Die übergebenen Samplewerte werden in Fenster unterteilt, mit einer
        Fensterfunktion gewichtet  und die Frequenzwerte bestimmt. Für die Berechnung wird der
        DFT-Algorithmus verwendet.
*/

        int iWindowLength=iFrameSize; //Fenster-Größe wird gesetzt, z.B. 256 Samples
        short shBuffer[512];     //Buffer für die gewichteteten Samplewerte, Max = 512
        ZeroMemory(shBuffer, sizeof(shBuffer)); //Elemente des shBuffer werden auf Null
                                                    //gesetzt
        float fDiv=(float)(2.0/(float)iWindowLength); //Divisor
        float fDFTReal=0.0;                         //realer Anteil (Kosinusanteil)
        float fDFTImag=0.0;                         //imaginärer Anteil (Sinusanteil)

        int iSteps=((SampleBufferLength / (iWindowLength/2))-1); //Anzahl der Fenster
        float fig=2*PI/iWindowLength; //Einsparung der Berechnungen: 2pi / 256

        int iBegin = 0; //Beginn des j-ten Fensters im SampleBuffer
```

```
        int iPos = 0; //Position im DFTBuffer
        int k=0; int i=0; int j=0; //Inkrementiervariablen
        while(j<iSteps) //Schleife für jedes Fenster
        {
                iBegin=j * iWindowLength/2;  //Anfangsposition des j-ten Fensters

                //Kopieren der Samples pro Fenster in den shBuffer
                memcpy(shBuffer, &SampleBuffer[iBegin], (sizeof(short)*iWindowLength));

                // Auswahl der zu verwendenden Fensterfunktion //
                if(iWindowFunction==1)
                        HammingWindow(shBuffer, iWindowLength); //Hammingfunktion
                else if(iWindowFunction!=0)
                        Kaiser_BesselWindow(shBuffer, iWindowLength); //Kaiser-Bessel-Funktion

                for(k=0;k<(iWindowLength/2);k++) //For-Schleife
                {
                        for(i=0;i<iWindowLength;i++) //Berechnung der Anteile
                        {
                                fDFTReal  += ( (float) shBuffer[i] * cos(k*i*fig) ); //Real-Anteil
                                fDFTImag += ( (float) shBuffer[i] * sin(k*i*fig) ); //Imaginär-Anteil
                        }
                        //Betragsspektrum wird aus dem Real- und Imaginäranteil berechnet //
                        DFTBuffer[k+iPos] = (short) (( fDiv * sqrt( fDFTImag*fDFTImag +
                                                fDFTReal*fDFTReal )) + 0.5 );
                        fDFTReal = 0.0;
                        fDFTImag = 0.0;
                }
                iPos += (iWindowLength/2); //Position im DFTBuffer wird hochgesetzt
                j++;
        }
        *DFTBufferLength = iPos; //Die Länge des DFT-Buffers wird gespeichert
} Ende der Methode DFT_Calculation
```

Die Resultate der DFT sowie die Fensterung des Sprachsignals können mit meinem Programm grafisch dargestellt werden. Als Beispiel dazu dient das über ein Mikrofon eingegebene Wort "Halt":

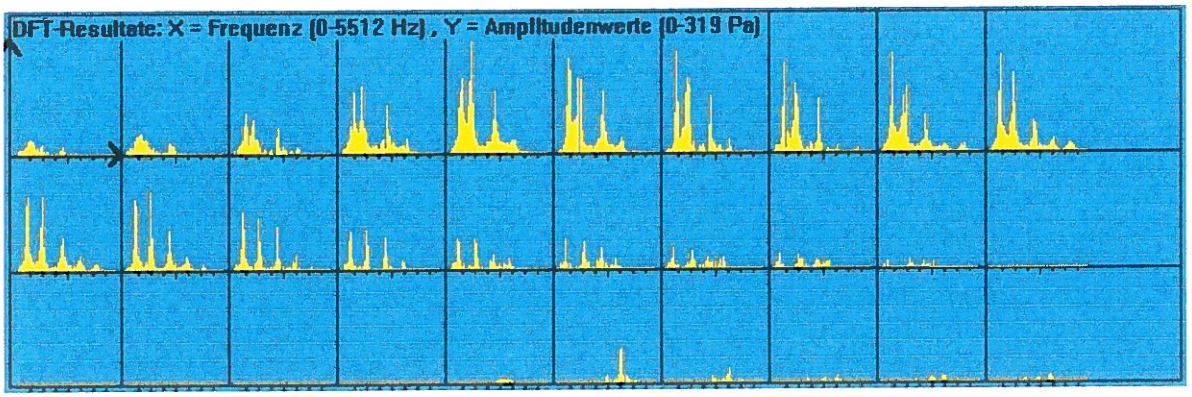

Abb. 14: DFT- Resultate für insgesamt 30 Fenster. Jedes Fenster hat eine eigene X- und Y-Achse. Als Fensterfunktion wurde das Kaiser-Bessel-Fenster verwendet. Die höchsten Amplitudenwerte befinden sich bei dem Vokal "a".

In der Abb. 14 ist das Wort „Halt", das mit der Abtastrate 11025 Hz aufgenommen wurde, im Spektralbereich dargestellt. Für die DFT ist eine Fensterlänge von 256 Samples verwendet worden. Es ist leicht zu erkennen, dass der Vokal „a" drei Formanten besitzt, dessen Amplitudenwerte im Vergleich zu den anderen Lauten groß sind. Der erste Peak liegt bei ca. 602 Hz und stellt die Grundfrequenz f_0 dar. Die weiteren drei Peaks sind ganzzahlige Vielfache dieser Grundschwingungen (Oberschwingungen): 1290 Hz (2 x f_0), 2451 Hz (4 x f_0), 3531 Hz (6 x f_0).

Die Ungenauigkeiten bei den Vielfachen resultieren aus der begrenzten Auflösungsfrequenz von $f_g = \frac{11025Hz}{256} = 34{,}07Hz$.

3.3.4 Mel-Filterbänke

Die Anwendung der Filterbänke auf das Betragsspektrum geschieht mit der Methode „EnergyValue_Calculation", die als Hilfsmethode „GetOneValueOfFilterbank" verwendet. Diese Hilfsmethode berechnet das Produkt

aus einem übergebenem Amplitudenwert mit dem passenden Funktionswert einer Dreicksfunktion und gibt das Ergebnis zurück. Die Dreiecksfunktion wird durch die Übergabe von der linken und rechten Frequenzgrenze sowie der Mittenfrequenz definiert.

```
float CASR:: GetOneValueOfFilterbank(int iLeftFrequence, int iMiddleFrequence, int iRightFrequence,
                                     int iAmplitude, int iFrequence)
{
/*
 * Methode:GetOneValueOfFilterbank
 * Übergabeparameter:
        EINGABE:
                int iLeftFrequence    = linke Frequenzgrenze (linke Grenze der Dreieckesfunktion)
                int iMiddleFrequence = mittlere Frequenz (Spitze der Dreieckesfunktion)
                int iRightFrequence = rechte Frequenzgrenze (rechte Grenze der Dreieckesfunktion)
                int iAmplitude      = die Amplitude, die zu iFrequence gehört
                int iFrequence      = die Frequenz, die zur Amplitude gehört
        AUSGABE:
                Rückgabewert der Funktion = das Produkt als Typ float
 * Funktion: Mithilfe der übergebenen Parameter wird das Produkt der Amplitude mit
         dem zugehörigen Wert der Dreiecksfunktion berechnet und dann zurückgegeben.
*/
        //Prüfe ob sich die Frequenz innerhalb der Grenzen befindet
        if(iFrequence >= iLeftFrequence && iFrequence <= iRightFrequence)
        {
                float iValue;
                int x=0;
                if(iFrequence <= iMiddleFrequence) //linke Dreieckshälfte
                {
                        //linkes Frequenzband der Dreiecksfunktion
                        x = iMiddleFrequence - iLeftFrequence;

                        //Berechne Dreieckswert
                        iValue = 1 - (((float)1/x) * (iMiddleFrequence - iFrequence));

                        //Multiplikation des Dreieckswertes mit der Amplitude
                        iValue *= iAmplitude;
                }
```

```
            else            //rechte Dreieckshälfte
            {
                    //rechtes Frequenzband der Dreiecksfunktion
                    x = iRightFrequence - iMiddleFrequence;

                    //Berechne Dreieckswert
                    iValue = (((float)1/x) * (iRightFrequence - iFrequence));

                    //Multiplikation des Dreieckswertes mit der Amplitude
                    iValue *= iAmplitude;
            }
            return iValue; //das berechnete Produkt wird zurückgegeben
        }
        else return 0; //die übergebene Frequenz befindet sich außerhalb der Dreiecksfunktion
                    //daher wird Null zurückgegeben
} //Ende der Methode
```

Die Methode „EnergyValue_Calculation" berechnet für jedes Fenster die Energiwerte der Filterbänke:

```
void CASR::EnergyValue_Calculation(unsigned __int16 *DFTBuffer, int DFTLength, unsigned
                                    __int32 *EnergyValueBuffer, int *EVBufferLength)
{
/*
 * Methode:EnergyValue_Calculation
 * Übergabeparameter:
        EINGABE:
            unsigned __int16 * DFTBuffer   = Zeiger auf den Buffer mit den Frequenzwerten
                (eine Liste mit positiven Zahlen: 0 bis 65536)
            DFTLength           =       Anzahl der besetzten Elemente des DFTLength
        AUSGABE:
            EnergyValueBuffer   =       Zeiger auf den Buffer für die Energiewerte
            EVBufferLength      =       Anzahl der besetzten Elemente des
                                        EnergyValueBuffer
 * Funktion: Die Energiewerte pro Fenster werden über Filterbänke ermittelt
    und abgespeichert.
*/
        CRecording *pRec=new CRecording; //Instanz von CRecording wird erstellt
```

```
//Auflösungsrate: Samplerate / Fensterlänge
float fVisRate = (float)(pRec->iSampleRate/iFrameSize);
delete pRec;    //Instanz von CRecording wird gelöscht

int iWindowLength=iFrameSize/2; //halbe Fenstergröße

int iPos=0;     //Index der benötigten Amplitudenwerte für ein Frequenzband
int i=0; int e=0; int w=0;  //Inkrementiervariablen: w für die Fenster, e für die
                            //Mittenfrequenzen, i für die Amplitudenwerte pro Fenster

int iNumOfWind = DFTLength / iWindowLength;     //Anzahl der Fenster
int iNumOfMiFre = shNumberOfMiddleFrequences; //Anzahl der Mittenfrequenzen z.B. 26

int iFrequence=0;               //Frequenzwert
float fFilterbankValue=0;       //Energiewert einer Filterbank
int iStart;                     //Anfangswert für die Schleife

for(w=0;w<iNumOfWind;w++) //Schleife für jedes Fenster
{
        iPos=w*iWindowLength;    //Anfangsposition: Fensteranfang

        for(e=0;e<iNumOfMiFre;e++) //Schleife für jede Mittenfrequenz
        {
                fFilterbankValue =0; //Energiewert wird auf Null gesetzt

                //Anfangswert für die nachfolgende Schleife
                iStart = (shMiddleFrequences[e]/fVisRate) - 1;

                for(i=iStart;i<iWindowLength;i++) //Schleife für die Amplitudenwerte
                {
                        iFrequence = fVisRate *i;       //Frequenzwert an der Stelle i
                        if(iFrequence>shMiddleFrequences[e+2])
                                break;   //Verlasse die Schleife

                        //Methode GetOneValueOfFilterbank wird aufgerufen und liefert das
                        //Produkt aus dem Amplitudenwert mit dem entsprechenden Wert
                        //der Dreiecksfunktion
                        fFilterbankValue += GetOneValueOfFilterbank
                                        (shMiddleFrequences[e], shMiddleFrequences[e+1],
```

```
                                        shMiddleFrequences[e+2], (DFTBuffer[i+iPos] *
                                        DFTBuffer[i+iPos]) , iFrequence);
                    }

                    //Energiewerte werden im Buffer abgespeichert
                    EnergyValueBuffer[e+w*iNumOfMiFre] = fFilterbankValue;
                }

        }

        //Speichern der Anzahl der Filterbänke aller Fenster
        *EVBufferLength=iNumOfWind*iNumOfMiFre;
} //Ende der Methode EnergyValue_Calculation
```

Das Wort "Halt" aus Abschnitt 3.3.3 wird jetzt nach der Berechnung der Filterbänke in folgender Weise dargestellt:

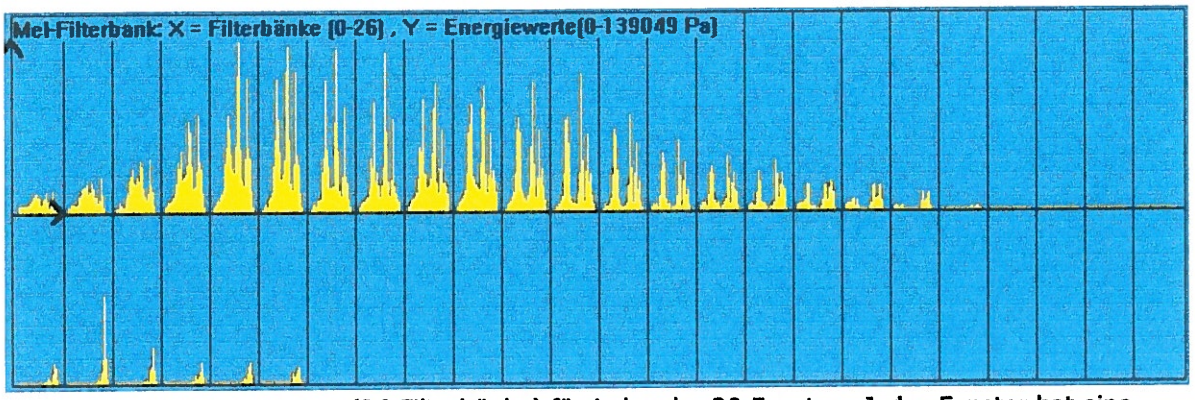

Abb. 15: 26 Energiewerte (26 Filterbänke) für jedes der 30 Fenster. Jedes Fenster hat eine eigene X- und Y-Achse. Die höchsten Ausschläge sind bei dem Vokal "a" und dem Konsonanten "t" zu erkennen.

3.3.5 Logarithmierung und Inverse Kosinus-Transformation (IDCT)

Die Energiewerte der Filterbänke werden mit der Methode „Logarithmus_Calculation" logarithmiert:

```
void CASR::Logarithmus_Calculation(unsigned __int32 *EnergyValueBuffer, int
                                   BufferLength, short *LogValueBuffer, int *LogBufferLength)
{
/*
 * Methode: Logarithmus_Calculation
 * übergebene Parameter:
        EINGABE:
                unsigned __int32 * EnergyValueBuffer = Zeiger auf den Buffer der Energiewerte der
                        Filterbänke (Liste mit positiven Zahlen von 0 bis 4294967296)
                int BufferLength = die Anzahl der besetzten Elemente des EnergyValueBuffer
        AUSGABE:
                short * LogValueBuffer= Zeiger auf den Buffer mit den logarithmierten
                        Energiewerten (eine Listen mit Zahlen des Typs short)
                int *LogBufferLength = die Anzahl der besetzten Elemente des LogValueBuffers
                        -> call by reference
 * Funktion: Aus jedem Energiewert wird der Logarithmus gebildet und dann im
        LogValueBuffer abgespeichert.
*/
        for(int i=0;i<BufferLength;i++) //Schleife für die Energiewert-Berechnung
        {
                if (EnergyValueBuffer[i]<1) //Falls der Energiewert kleiner 1 ist
                        LogValueBuffer[i]=1;
                else
                        //Der Logarithmus zur Basis 2 wird gebildet
                        LogValueBuffer[i]=(short)log(EnergyValueBuffer[i]);
        }

        *LogBufferLength=BufferLength; //Speichern der Länge des LogBufferLength
} //Ende der Methode Logarithmus_Calculation
```

Die Energiewerte aus Abschnitt 3.3.4 können nach der Logarithmierung in folgender graphischer Weise dargestellt werden:

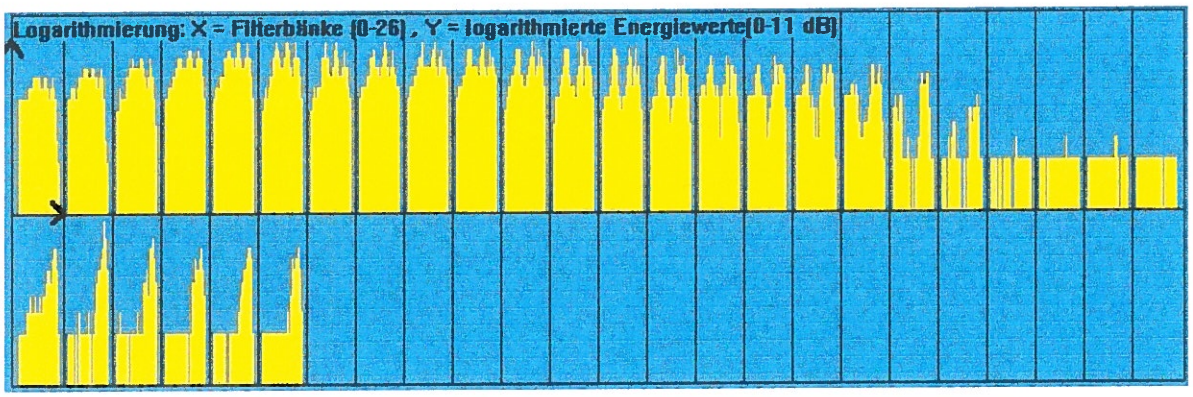

Abb. 16: 26 logarithmierte Energiewerte für jedes der 30 Fenster. Jedes Fenster hat eine eigene X- und Y-Achse.

Letztendlich werden die Mel-Frequenz-Kepstral Koeffizienten mittels der IDCT berchnet:

```
void CASR::MFCC_Calculation(short *LogValueBuffer, int LogBufferLength, short
                                *MFCCBuffer, int *MFCCBufferLength)
{
/*
 * Methode: MFCC_Calculation
  * übergebene Parameter:
        EINGABE:
                short * LogValueBuffer = Zeiger auf den Buffer mit den logarithmierten
                        Energiewerten der Filterbänke (eine Liste mit Zahlen des Typs short)
                int LogBufferLength = die Anzahl der besetzten Elemente des LogValueBuffer
        AUSGABE:
                short * MFCCBuffer  = Zeiger auf den Buffer mit den  MFC-Koeffizienten (eine
                        Liste mit Zahlen des Typs short)
                int * MFCCBufferLength = die Anzahl der besetzten Elemente des MFCCBuffer
                        -> call by reference
  * Funktion: Aus den Energiewerten jedes Fensters wird eine bestimmte Anzahl
         von MFCC-Koeffizienten berechnet und abgespeichert.
*/

      int iNumMiddleFre = shNumberOfMiddleFrequences; //Anzahl der Mittenfrequenzen
      int iNumMFCC   = shNumberOfMFCC; //Anzahl der verwendeten IDCT-Koeffizienten
      float fMFCC=0.0;
```

```
int iSteps=LogBufferLength /iNumMiddleFre; //Anzahl der Fenster mit den
                                                    //logarithmierten Energiewerten
int w=0; int k=0; //Inkrementiervariablen
int iInd=0; //Index
float fig=(float) PI/(2*iNumMiddleFre); //Einsparung von Mehrfachberechnungen

for(w=0;w<iSteps;w++) //Schleife für jedes Fenster
{
    for(k=1;k<=iNumMFCC;k++) //Schleife für die Anzahl der MFC-Koeffizienten
    {
        iInd=w*iNumMiddleFre;
        for(int i=1;i<=iNumMiddleFre;i++) //Schleife für die Anzahl der Mittenfrequenzen
        {
            // Kosinustransformation //
            fMFCC += (float) (LogValueBuffer[iInd]* (float) cos(( (float)
                        k*((float)i+(float)2)*fig) ));
            iInd++;
        }
        iInd = w*iNumMFCC + k-1; //Index wird für den MFCCBuffer neu gesetzt
        MFCCBuffer[iInd] = (short)fMFCC; //Speichern der MFC-Koeffizienten
        fMFCC=0.0;
    }
}
*MFCCBufferLength=iInd+1; //die Länge des MFCCBuffer wird gespeichert
} //Ende der Methode MFCC_Calculation
```

Die Mel-Frequenz-Kepstral Koeffizienten, die sich aus der Anwendung der IDCT auf die log. Eneriewerte ergeben, werden folgendermaßen visualisiert:

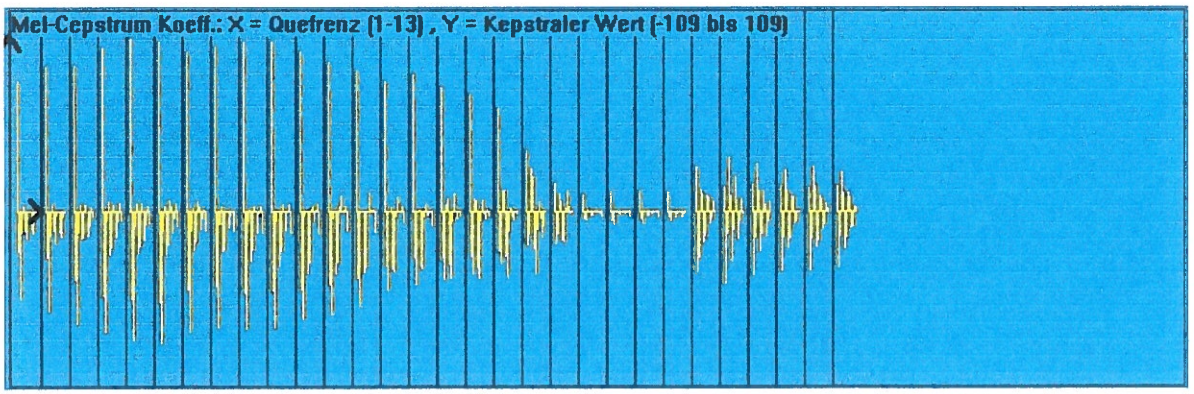

Abb. 17: 13 Mel-Kepstrum Koeffizienten für jedes der 30 Fenster. Jedes Fenster hat eine eigene X- und Y-Achse.

Die gewonnenen Koeffizienten enthalten Charakteristiken des Vokaltraktes und werden für die Spracherkennung als Merkmale eingesetzt. Die einzelnen Peaks am Anfang eines Fensters, bspw. beim Vokal „a" des Wortes „Halt", sind ein Maß für die Stimmhaftigkeit dieses Lautes.

3.3.6 Hinzufügen der Lautheit

Die Berechnung der Energie des Sprachsignals eines jeden Fensters habe ich in meinem Programm folgendermaßen implementiert:

```
void CASR::Energy_Calculation(unsigned __int32 *FilterbankBuffer,  int
                                        BufferLength,  int *Energy)
{
/*
 * Methode: Energy_Calculation
 * Übergabeparameter:
        EINGABE:
                unsigned __int32 * FilterbankBuffer   =  Zeiger auf den Buffer mit den
                        Energiewerten (Liste mit positiven Zahlen von 0 bis 4294967296)
                int BufferLength =  Anzahl der besetzten Elemente des FilterbankBuffer
        AUSGABE:
                int * Energy     =  Zeiger auf den Energiewert eines Fensters -> call by reference
*/

        double logar=0.0;
        for(int i=0;i<BufferLength;i++) //Schleife
        {
                logar+=(double)log10(FilterbankBuffer[i]); //Logarithmus zur Basis 10
        }
        *Energy=(int)logar; //Die Energie des Sprachsignals wird gespeichert
}//Ende der Methode Energy_Calculation
```

3.4 Vergleich zweier Prototypen mittels der Dynamischen Zeitverzerrung und anschließende Worterkennung

Nachdem ein Wort über ein Mikrofon eingegeben wurde, wird daraus ein Prototyp berechnet, der mit den vorhandenen, trainierten Prototypen verglichen wird. Die Resultate der Vergleiche sowie das erkannte Wort wird mit einem Dialog angezeigt. Als Beispiel dient wieder das Wort "Halt" aus Abschnitt 3.3.3:

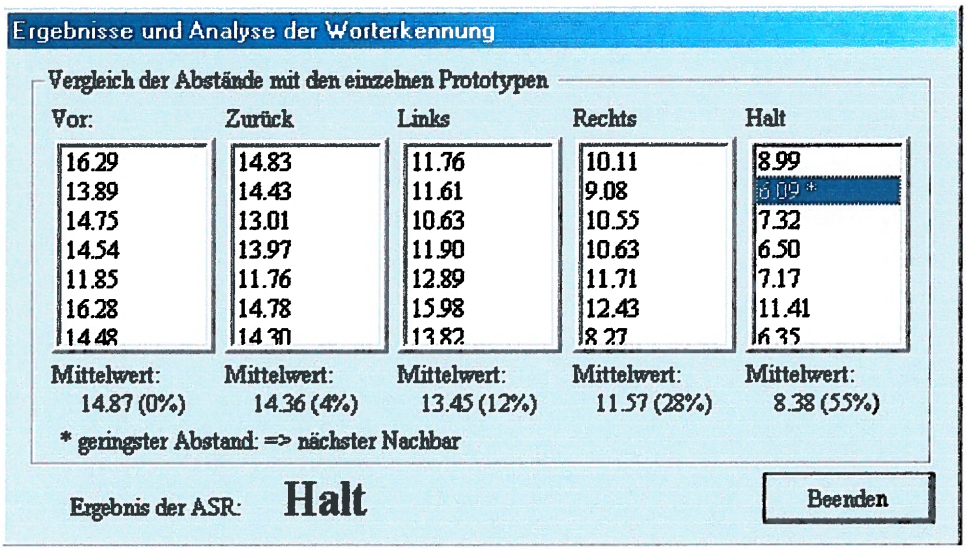

Abb. 18: Dialog mit den Ergebnissen der Dynamischen Zeitverzerrung und der Worterkennung.

Der geringste Einzelabstand des Wortes „Halt" wird in einem Listenfeld mit einem blauen Farbhintergrund hervorgehoben (Abb. 18). Der Mittelwert beinhaltet den Abstand und die prozentuale Wahrscheinlichkeit der Übereinstimmung.

4. Gewinnung der Merkmale anhand eines konkreten Zahlenbeispiels

Auf die Funktion $f(t)$, dargestellt in der Abb. 19, werden die Transformationen aus den Abschnitten 2.3.2 bis 2.3.7 angewandt. Die Preemphase aus Abschnitt 2.3.1 entfällt, da die Funktion zuvor nicht tiefpassgefiltert wurde. Die Dezimalklassifikation bezieht sich auf Kapitel 2.

$$f(t)=\begin{cases}2000*[\sin(689*2\pi t)+\sin(1378*2\pi t)+\sin(3445*2\pi t-40)+\sin(4134*2\pi t+3)] & 0\le t\le 2.9*10^{-3}\\ 0 & \text{sonst}\end{cases}$$

(Gl.4.1)

Die kontinuierliche Funktion $f(t)$ (siehe Gl. 4.1) wird mit einer Frequenzrate von 11025 Hz abgetastet, d.h. alle $9.07*10^{-5}$ s ein Samplewert. Es ergeben sich 32 diskrete Werte der Funktion f_n:

n	0	1	2	3	4	5	6	7
f_n	-1207	-82	7391	1195	383	3920	-2676	-1250

n	8	9	10	11	12	13	14	15
f_n	1772	-288	568	-767	-948	-3549	-5283	823

n	16	17	18	19	20	21	22	23
f_n	-1207	-82	7391	1195	383	3920	-2676	-1250

n	24	25	26	27	28	29	30	31
f_n	1772	-288	568	-767	-948	-3549	-5283	823

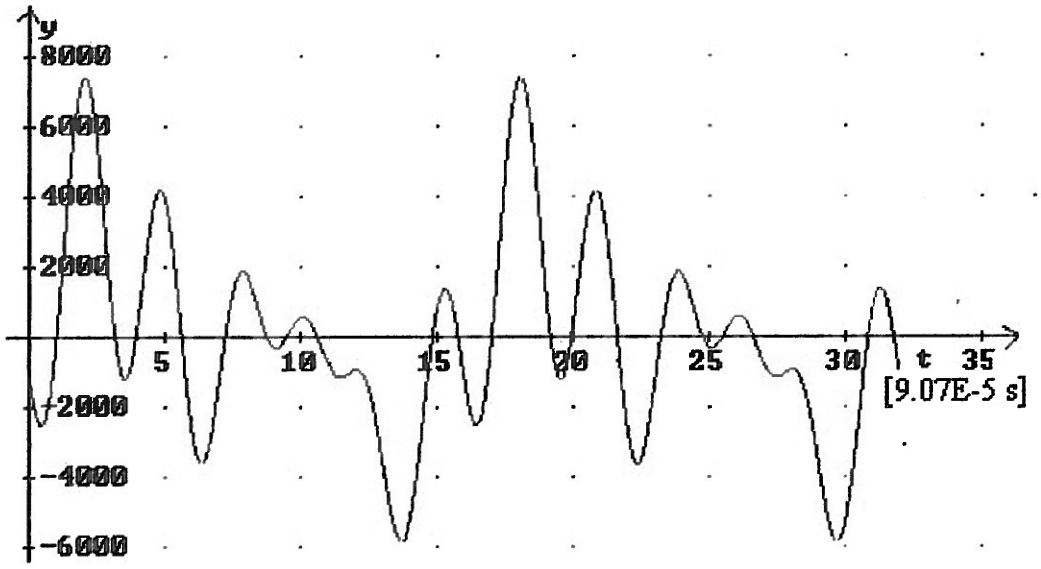

Abb. 19: Darstellung der kontinuierlichen Funktion $f(t)$, die $2.902*10^{-3}$ s lang dauert.

4.3.2 Fensterung und Fensterfunktion

Die Funktion f_n hat 32 Samplewerte bzw. dauert 2.9 ms. Ein Fenster der Länge 32 würde bei der nachfolgenden DFT eine Auflösungsfrequenz von 344 Hz ($f_g = \frac{11025\,Hz}{32} = 344\,Hz$) bewirken. Damit ließen sich keine genauen Frequenzen ermitteln. Daher mache ich mir das „Zero-Padding" zu Nutze, indem ich den 32 Samplewerten 224 Nullen hinzufüge. Zusammen ergibt dies ein Fenster der Länge 256 und eine Auflösungsfrequenz von 43 Hz ($f_g = \frac{11025\,Hz}{256} = 43\,Hz$). Diese Asuflösung ist ausreichend für die Freqenzbestimmung über dir DFT im nächsten Abschnitt.

Als Fensterfunktion habe ich die Hammingfunktion verwendet, da sie einen Kompromiss zwischen dem Rechteck- und dem Kaiser-Bessel-Fenster, bezüglich des Rechenaufwandes, darstellt.

Zuerst wird die Hamming-Fensterfunktion auf die Funktion f_n angewendet, indem man die Funktionswerte an den Stellen n miteinander multipliziert:

$$fh_n = f_n * f_{ham}(n) \qquad \text{mit} \quad f_{ham}(n) = 0.54 - 0.46 * \cos(\frac{2\pi n}{255})$$

$$\text{und} \quad n = 0,1,2,...,255$$

n	0	1	2	3	4	5	6	7
f_n	-1207	-82	7391	1195	383	3920	-2676	-1250
$f_{ham}(n)$	0.0801	0.0805	0.08125	0.0822	0.0834	0.08501	0.0868	0.0889
fh_n	-96.68	-6.601	600.51	98.229	31.942	333.23	-232.2	-111.12

n	8	9	10	11	12	13	14	15
f_n	1772	-288	568	-767	-948	-3549	-5283	823
$f_{ham}(n)$	0.0912	0.0938	0.0967	0.0999	0.1033	0.1070	0.1110	0.1152
fh_n	161.6	-27.01	54.925	-76.62	-97.92	-379.74	-586.4	94.8

n	16	17	18	19	20	21	22	23
f_n	-1207	-82	7391	1195	383	3920	-2676	-1250
$f_{ham}(n)$	0.1197	0.1245	0.1294	0.1347	0.1402	0.1459	0.1519	0.1581
fh_n	-144.4	-10.2	956.39	160.96	53.69	571.92	-406.4	-197.6

n	24	25	26	27	28	29	30	31
f_n	1772	-288	568	-767	-948	-3549	-5283	823
$f_{ham}(n)$	0.1645	0.1712	0.1780	0.1852	0.1925	0.2	0.2077	0.2157
fh_n	291.49	-49.3	101.1	-142	-182.4	-709.8	-1097	177.52

n	32	33	34	...	...	...	254	255
f_n	0	0	0	0	0	0	0	0
$f_{ham}(n)$	0.2238	0.2322	0.2407	...	...	...	0.08013	0.08

fh_n	0	0	0	0	0	0	0	0

Das Ergebnis der Gewichtung wird in der Abb. 20 graphisch dargestellt. Damit der Unterschied zur vorherigen Abb. 19 besser sichtbar wird, wurden die Punkte miteinander verbunden.

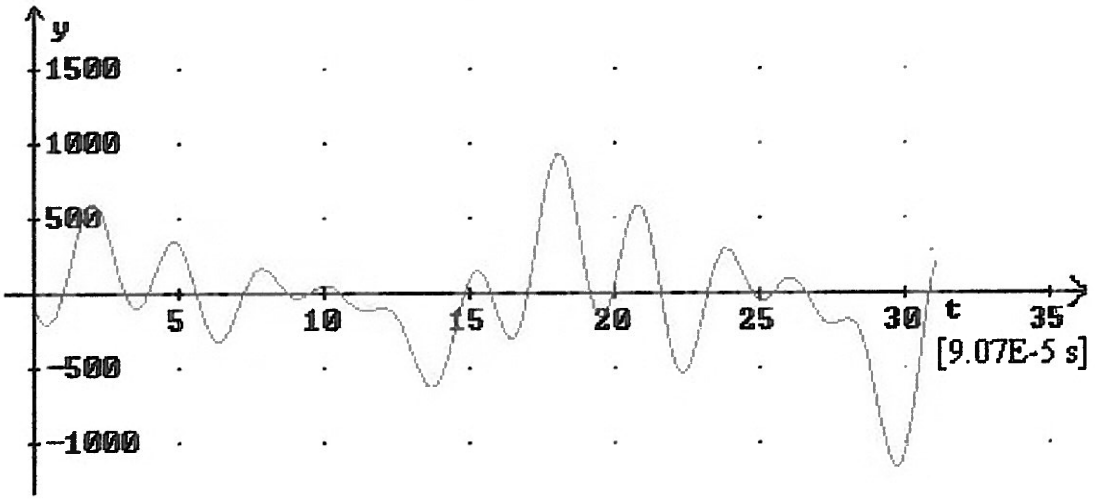

Abb. 20: Darstellung der Funktion fh_n, d.h. der mit dem Hammingfenster gewichteten Funktion f_n

4.3.3 Diskrete Fourier-Transformation (DFT)

Zuerst werden die Sinus- und Kosinusanteile berechnet, anschließend folgt die Bildung der Amplitudenwerte. Dabei ist zu beachten, dass man in die Formel für *ak* bzw. bk, aufgrund des Zero-Paddings, die Anzahl der Samplewerte auf 32 setzt und nicht auf 256.

$$a_k = \frac{2}{32} * \sum_{n=0}^{31} fh_n * fc_k(n)) \qquad fc_k(n) = \cos(\frac{2\pi kn}{256})$$

k = 1,2,3,.....,128

$$b_k = \frac{2}{32} * \sum_{n=0}^{31} fh_n * fs_k(n)) \qquad fs_k(n) = \sin(\frac{2\pi kn}{256})$$

k = 1,2,3,.....,128

Folgende Tabelle zeigt die Berechnungen, die getätigt werden müssen um bspw. die Amplitudenwerte für k=1 und k=33 zu erhalten.

		k = 1				k = 33			
n	fh_n	$fc_1(n)$	$fh_n * fc_1(n)$	$fs_1(n)$	$fh_n * fs_1(n)$	$Fc_{33}(n)$	$fh_n * fc_{33}(n)$	$Fs_{33}(n)$	$fh_n * fs_{33}(n)$
0	-96.68	1	-96.6	0	0	1	-96.68	0	0
1	-6.601	0.999	-6.59	0.0245	0.161	0.6895	-4.551	0.724	-4.779
2	600.51	0.998	599.3	0.0490	29.42	-0.049	-29.42	0.998	599.3
3	98.229	0.997	97.93	0.0735	7.219	-0.757	-74.35	0.653	64.14
4	31.942	0.995	31.78	0.0980	3.130	-0.995	-31.78	-0.098	-3.13
5	333.23	0.992	330.5	0.1224	40.78	-0.615	-204.9	-0.788	-262.5
6	-232.2	0.989	-229	0.1467	-34.06	0.146	-33.9	-0.989	229.6
7	-111.1	0.985	-109	0.1709	-18.98	0.8175	-90.82	-0.575	63.88
8	161.6	0.980	158.3	0.1950	31.51	0.9807	158.4	0.195	31.5
9	-27.01	0.975	-26.3	0.2191	-5.91	0.5349	-14.44	0.844	-22.79
10	54.925	0.970	53.27	0.2429	13.34	-0.242	-13.29	0.970	53.27
11	-76.62	0.963	-73.7	0.2667	-20.43	-0.870	66.65	0.492	-37.6
12	-97.92	0.956	-93.6	0.2902	-28.4	-0.956	93.61	-0.290	28.3
13	-379.74	0.949	-360	0.3136	-119	-0.449	170.5	-0.893	339.1
14	-586.4	0.941	-551	0.3368	-197.4	0.3368	-197.4	-0.941	551.8
15	94.8	0.932	88.35	0.3598	34.1	0.9142	86.66	-0.405	-38.39
16	-144.4	0.923	-133	0.3826	-55.2	0.9238	-133.3	0.382	-55.16
17	-10.2	0.914	-9.32	0.4052	-4.13	0.3598	-3.66	0.932	-9.5
18	956.39	0.903	863.6	0.4275	408.8	-0.427	-408.3	0.903	863.62
19	160.96	0.893	143.7	0.4496	72.36	-0.949	-152.7	0.313	50.38
20	53.69	0.881	47.3	0.4713	25.3	-0.881	-47.3	-0.471	-25.28
21	571.92	0.870	497.5	0.4928	281.8	-0.266	-152.1	-0.963	-550.7
22	-406.4	0.857	-348	0.5141	-208	0.5141	-208.9	-0.857	348.2

23	-197.6	0.844	-166	0.5349	-105	0.9757	-192.7	-0.219	43.27
24	291.49	0.831	242.2	0.5555	161.9	0.8314	242.3	0.555	161.7
25	-49.3	0.817	-40.2	0.5758	-28.3	0.1709	-8.4	0.985	-48.56
26	101.1	0.803	81.18	0.5956	60.2	-0.595	-60.1	0.803	81.18
27	-142	0.788	-111	0.6152	-87.3	-0.992	140.8	0.122	-17.32
28	-182.4	0.773	-140	0.6343	-115.6	-0.773	140.9	-0.634	115.6
29	-709.8	0.757	-537	0.6531	-463.5	-0.073	51.8	-0.997	707.6
30	-1097	0.740	-811	0.6715	-736.6	0.6715	-736.6	-0.740	811.78
31	177.52	0.724	128.5	0.6895	122.4	0.9996	177.4	-0.024	-4.26
		a_1 = 1/16 * -334.7 = -20 b_1 = 1/16 * -730 = -45				a_{33} = 1/16 * -1566.5 = -97 b_{33} = 1/16 * 4064.2 = 254			

Führt man die Tabelle für die restlichen Werte für k (k = 1,2,...,128) fort, erhält man jeweils die Sinus- und Kosinusanteile. Das Amplitudenspektrum berechnet sich wie folgt:

$$A_k = \sqrt{(a_k)^2 + (b_k)^2} \qquad \text{mit } k = 1,2,...,128$$

Folgende Tabelle umfasst alle Ergebnisse:

k	a_k	b_k	A_k	k	a_k	b_k	A_k	k	a_k	b_k	A_k	k	a_k	b_k	A_k
1	-20	-45	49	33	-97	254	271	65	38	32	49	97	120	-206	238
2	28	-81	85	34	-192	174	259	66	5	71	71	98	186	-129	226
3	87	-60	63	35	-232	61	239	67	-46	78	90	99	201	-31	203
4	116	-3	116	36	-201	-47	206	68	-90	47	101	100	161	54	169
5	97	59	113	37	-119	-114	164	69	-101	-7	101	101	87	99	87
6	39	92	99	38	-23	-118	120	70	-71	-55	89	102	10	92	92
7	-28	75	80	39	46	-66	80	71	-13	-71	72	103	-40	47	61
8	-67	11	67	40	64	8	64	72	44	-40	59	104	-48	-11	49
9	-51	-68	85	41	29	69	74	73	70	26	74	105	-19	-54	57
10	19	-123	124	42	-31	88	93	74	45	102	111	106	26	-63	68
11	119	-121	169	43	-85	61	104	75	-26	151	153	107	62	-40	78
12	205	-53	211	44	-104	6	104	76	-120	150	192	108	72	0	72

13	236	59	243	45	-81	-45	92	77	-201	93	221	109	52	34	62
14	196	174	262	46	-31	-66	72	78	-238	0	238	110	16	45	47
15	96	247	265	47	18	-48	51	79	-220	-98	240	111	-15	30	33
16	-25	252	253	48	41	-3	41	80	-157	-168	229	112	-26	0	26
17	-124	191	227	49	27	43	50	81	-73	-194	207	113	-14	-29	32
18	-168	93	192	50	-13	66	67	82	2	-176	176	114	13	-39	41
19	-149	0	149	51	-58	54	79	83	49	-130	138	115	39	-28	48
20	-89	-55	104	52	-82	16	83	84	65	-79	102	116	48	2	48
21	-22	-61	64	53	-72	-28	77	85	58	-38	69	117	37	22	43
22	18	-30	34	54	-34	-53	62	86	42	-13	43	118	13	32	34
23	22	6	22	55	9	-45	45	87	27	2	27	119	-10	23	25
24	0	23	23	56	35	-9	36	88	13	16	20	120	-20	0	20
25	-22	6	22	57	29	35	36	89	-7	31	31	121	-11	-22	24
26	-18	-30	34	58	-4	63	36	90	-40	40	56	122	10	-31	32
27	23	-60	64	59	-49	59	76	91	-85	31	90	123	32	-22	38
28	89	-54	104	60	-79	25	82	92	-129	-5	129	124	41	0	41
29	148	1	148	61	-77	-19	79	93	-150	-71	165	125	32	20	37
30	165	94	189	62	-43	-50	65	94	-132	-149	199	126	11	29	31
31	122	191	226	63	2	-50	50	95	-69	-213	223	127	-10	20	22
32	24	251	252	64	36	-16	39	96	24	-237	238	128	-18	0	18

Auffällig große Amplitudenwerte befinden sich an den Stellen k = 15, 33, 79 und 96/97 (zugehörige A_k -Werte sind gelb hinterlegt). Deren zugehörige Frequenzen lassen sich über $f_k = k * f_g$ berechnen, mit der Auflösungsfrequenz

$$f_g = \frac{11025\ Hz}{256} = 43\ Hz\ .$$

k	15	33	79	96/97
f_k [Hz]	645	1419	3397	4128/4171

Mit meinem Programm können die informatisch berechneten Frequenzen dargestellt werden. Damit mehrere Fenster dargestellt werden können, wurde die Funktion f_n hintereinander gereiht (Abb. 21). In der Grafik erkennt man deutlich die Amplitudenmaxima an den vier Frequenzen, aus denen die Funktion f_n zusammengesetzt ist. Diese Frequenzen sind auch für die Erzeugung der kontinuierlichen Funktion $f(t)$ (siehe Gl. 4.1) verwendet worden, d.h. die Frequenzbestimmung über die DFT ist erfolgreich gewesen. Die vorhandenen Abweichungen sind mit der begrenzten Auflösungsfrequenz und mit Rundungsfehlern zu begründen.

Auffällig bei der Abbildung 21 ist, dass neben den vier höchsten Amplitudenmaxima weitere kleinere spektrale Komponenten vorhanden sind, obwohl diese nicht in der Funktion $f(t)$ vorkommen. Die Ursache dafür sind die wenigen diskreten Werte, die die Funktion beschreiben. Bei nur 32 Samplewerten kann es vorkommen, dass zufälligerweise Kurven mehrerer Frequenzen, die nicht in der Funktion vorhanden sind, auf die diskreten Werte „passen" und übereinstimmen.

Mithilfe der a_k bzw. b_k Werte aus der Tabelle kann man Aussagen über die Kosinus- bzw. Sinusähnlichkeit der Einzelkurve machen, z.B. ist der b_k – gegenüber dem a_k –Wert bei der Frequenz 645 Hz (k=15) dominant, woraus man schließen, das es sich um eine Sinuskurve handelt. Bei 1419 Hz handelt es sich ebenfalls um eine Sinuskurve und bei den letzten beiden Frequenzen um Kosinuskurven (aufgrund der Verschiebung um -40 bzw. + 3, siehe Gl.4.1).

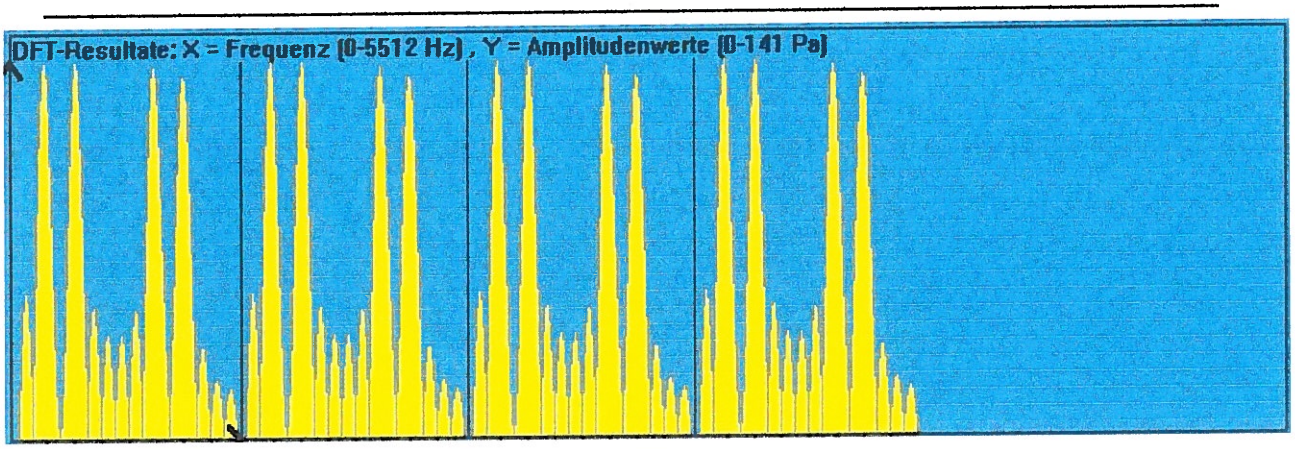

Abb. 21: Die Resultate der Frequenzberechnungen sind erkennbar. Man sieht vier hohe spektrale Gipfel, die untereinander fast gleich hohe Amplitudenwerte besitzen.

4.3.4 Gewichtung über Mel-Filterbänke

In diesem Abschnitt werden die Amplitudenwerte A_k mit Hilfe von Filterbänken gewichtet. Um den Frequenzbereich von 5525 Hz abzudecken, verwende ich 26 Filterbänke mit den folgenden Mittenfrequenzen (= 26 + 2, aufgrund der zwei Begrenzungen links und rechts):

Nr.	0	1	2	3	4	5	6	7	8	9
[Hz]	100	150	200	250	300	350	400	450	500	575

Nr.	10	11	12	13	14	15	16	17	18	19
[Hz]	650	725	800	875	950	1025	1125	1250	1400	1600

Nr.	20	21	22	23	24	25	26	27
[Hz]	1850	2150	2525	2925	3525	4200	5025	5525

(Tab. 4.1)

Anhand der Tabelle 4.1 erkennt man die lineare Abdeckung von 100 Hz bis 500 Hz in 50 Hz-Schritten, die Gleichaufteilung von 500 Hz bis 1025 Hz in 75 Hz-Schritten und die steigende Aufteilung von 1025 Hz bis 5525 Hz.

Über folgende Formel erhält man die Energiewerte einer Filterbank:

$$fb_m = \sum_{k=1}^{128} dm(f_k) * A_k^2 \qquad m = 1,2,\ldots, 128$$

Um zu demonstrieren wie man die Energiewerte berechnet, beschreibe ich nun ausführlich die Gewichtung der Amplitudenwerte mit Hilfe der 18. Filterbank (siehe Tab.4.1). Die Dreiecksfunktion ist definiert über die Mittenfrequenz mit 1400 Hz (Spitze des Dreiecks: Wert ist 1) und über die linke sowie die rechte Frequenzgrenze mit 1250 Hz bzw. 1600 Hz. Alle anderen Frequenzen außerhalb der Spanne 1250-1600 Hz nehmen den Wert Null an. Folgende Formel gilt für die Funktionswerte der 18. Dreiecksfunktion:

$$d_{18}(fk) = \begin{cases} 1 - \dfrac{(1400 - f_k)}{150} & 1250 \le f_k \le 1400 \\ \dfrac{(1600 - f_k)}{200} & 1400 < f_k \le 1600 \\ 0 & f_k < 1250 \text{ oder } f_k > 1600 \end{cases} \qquad \text{(vgl. Gl. 2.4)}$$

Die Frequenzwerte f_k lassen sich über $f_k = k * f_g$ (f_g = 43 Hz) berechnen. In der Tabelle 4.1 sind die Einzelwerte und das Ergebnis, der Energiewert der 18. Filterbank, dargestellt.

f_k [Hz]	43	86	...	1290	1333	1376
A_k^2	2401	7225	...	35721	51076	63504
$d_{18}(f_k)$	0	0	0	0.266	0.55	0.84
$A_k^2 * d_{18}(f_k)$	0	0	0	9501	28091	53343

f_k [Hz]	1419	1462	1505	1548	1591	...
A_k^2	73441	67081	57121	42436	26896	...
$d_{18}(f_k)$	0.905	0.69	0.475	0.26	0.045	0
$A_k^2 * d_{18}(f_k)$	66464	46285	27132	11033	1210	0
fb_m = 243059						

Die Energiewerte für die restlichen Filterbänke sind in der nächsten Tabelle zusammengefasst.

m	1	2	3	4	5	6	7	8	9
fb_m	11616	**17483**	15046	13793	12076	15067	29291	112452	132086

m	10	11	12	13	14	15	16	17	18
fb_m	**135252**	101601	80940	6386	1653	2653	25990	124744	**243059**

m	19	20	21	22	23	24	25	26
fb_m	181640	57048	43034	57356	275707	**460050**	336025	126110

Aus der Tabelle ist ablesbar, dass es vier Spitzen (Gipfel) gibt (fett hervorgehoben). Mein Programm zeigt die Filterbänke in folgender Abb. graphisch an:

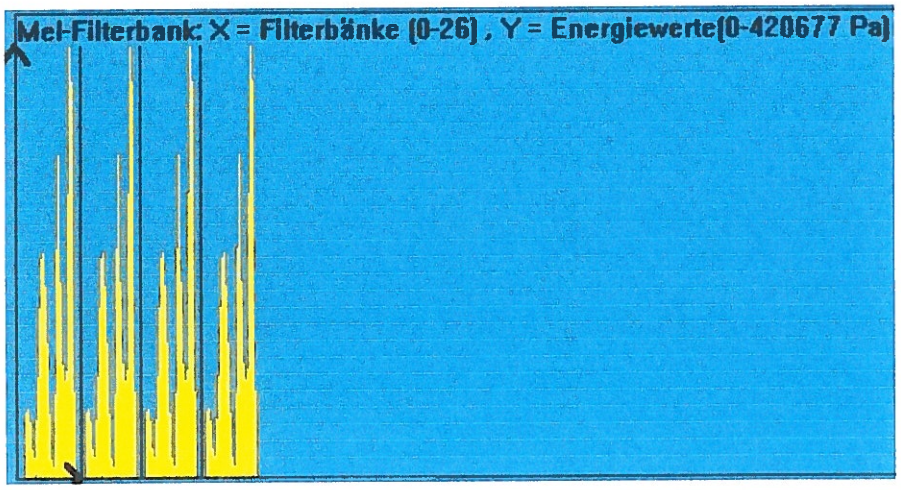

Abb. 22: Darstellung der Energiewerte der 26 Filterbänke mit meinem Programm. Es sind vier Gipfel zu erkennen, die die höchsten spektralen Intensitäten beinhalten.

Die Grafik der Abb. 22 spiegelt die Tonhöhenwahrnehmung des Frequenzbereiches von 0 bis 5512 Hz durch das menschliche Gehör wieder, d.h. unser Gehör fasst alle spektralen Intensitäten A_k (k=0,1,...128) in 26

Energiewerte, die jeweils für einen bestimmten Frequenzbereich (Breite der zugehörigen Dreiechsfunktion) stehen, zusammen.

4.3.5 Logarithmierung und IDCT

Nun werden die 26 Filterbankwerte logarithmiert (zur Basis 2) und anschließend der Inversen Kosinus-Transformation unterzogen, die 13 MFC-Koeffizienten erzeugt:

$$Y(u) = \sum_{m=1}^{26} \log_2(fb_m) * co_u(m)) \quad \text{für } 1 \le u \le 13$$

$$\text{mit } co_u(m) = \cos(\frac{u*(m+2)*\pi}{52})$$

Eine detaillierte Berechnung des MFC-Koeffizienten für $u = 4$ beschreibt die nachfolgende Tabelle:

m	1	2	3	4	5	6	7	8	9	10
$log_2(fb_m)$	13.5	14.09	13.87	13.75	13.55	13.87	14.83	16.77	17.01	17.04
$co_4(m)$	0.748	0.568	0.354	0.12	-0.12	-0.35	-0.56	-0.74	-0.88	-0.97
$log_2(fb_m) * co_4(m)$	10.09	8.003	4.909	1.65	-1.62	-4.85	-8.30	-12.4	-14.9	-16.5

m	11	12	13	14	15	16	17	18	19	20
$log_2(fb_m)$	16.63	16.3	12.64	10.69	11.37	14.66	16.92	17.89	17.47	15.79
$co_4(m)$	-1	-0.97	-0.88	-0.74	-0.56	-0.35	-0.12	0.120	0.354	0.568
$log_2(fb_m) * co_4(m)$	-16.6	-15.8	-11.1	-7.91	-6.36	-5.13	-2.03	2.146	6.184	8.968

m	21	22	23	24	25	26

$log_2(fb_m)$	15.39	15.8	18.07	18.81	18.35	16.94
$co_4(m)$	0.748	0.885	0.970	1	0.970	0.885
$log_2(fb_m) * co_4(m)$	11.51	13.98	17.52	18.81	17.79	14.99
$Y(4) = 13.05$						

Die weiteren Ergebnisse für $1 \le u \le 13$ sehen wie folgt aus:

u	1	2	3	4	5	6	7	8	9	10	11	12	13
$Y(u)$	204	-93	-115	12	2	-83	-55	14	-3	-37	-10	-2	-45

In meinem Programm werden die informatisch berechneten MFC-Koeffizienten angezeigt:

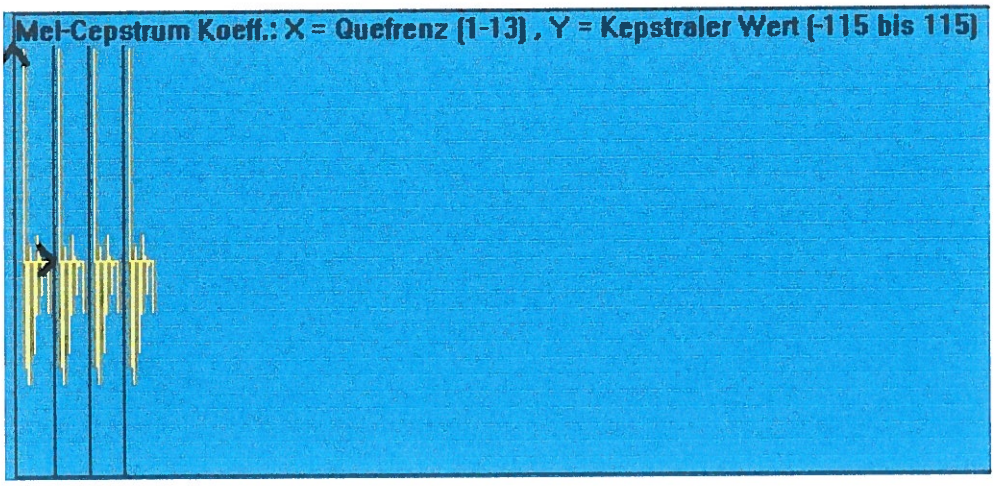

Abb. 23: Darstellung der 13 MFC-Koeffizienten.

Mit den MFC-Koeffizienten hat man nun eine reduzierte Anzahl von Merkmalen, die für die komplexe Funktion f_n charakteristisch sind.

4.3.6 Berechnung der Lautheit

Die Lautheit ist ein Maß für die subjektive Lautstärkeempfindung. Man kann sie näherungsweise berechnen, indem man die logarithmierten Energiewerte der Filterbänke aufsummiert:

$$L = \sum_{m=1}^{26} \log_2(fb_m)$$

m	1	2	3	4	5	6	7	8	9	10	11	12	13
$log_2(fb_m)$	13.5	14.1	13.8	13.7	13.5	13.8	14.8	16.7	17	17	16.6	16.3	12.6

m	14	15	16	17	18	19	20	21	22	23	24	25	26
$log_2(fb_m)$	10.6	11.3	14.6	16.9	17.8	17.4	15.7	15.3	15.8	18	18.8	18.3	16.9
L = 400													

4.3.7 Prototyp

Der Prototyp beinhaltet die Merkmale der Funktion f(t), also die 13 MFC-Koeffizienten und den Lautheitswert. Diese werden in einer Liste gespeichert.

5. Programm

Für die Erstellung des Programms „Vox-Mobil" habe ich die Programmiersprache C++ verwendet. Die Vorteile dieser Sprache sind die Möglichkeit zur Objekt Orientierten Programmierung (OOP) und die Schnelligkeit bei der Ausführung von Anwendungen. Zudem habe ich einige Kenntnisse in dieser Sprache, da ich mich seit ca. drei Jahren mit C++ beschäftige und Programme schreibe. „Vox-Mobil" habe ich mit der Entwicklungsumgebung Visual C++ 6 von Microsoft erstellt. Das

Projekt umfasst 22 Quellcode-Dateien (11 Klassen, bestehend aus je 2 Dateien), die zusammen 335 KByte Speicher einnehmen (Stand 17.4.04).
Die Anwendung „Vox-Mobil" besteht aus einer Hauptseite und vier Unterprogrammen, die nachfolgend erklärt werden.

5.1 Hauptseite

Die Hauptseite enthält vier Schaltflächen, über die man zu den entsprechenden Unterprogrammen gelangen kann.

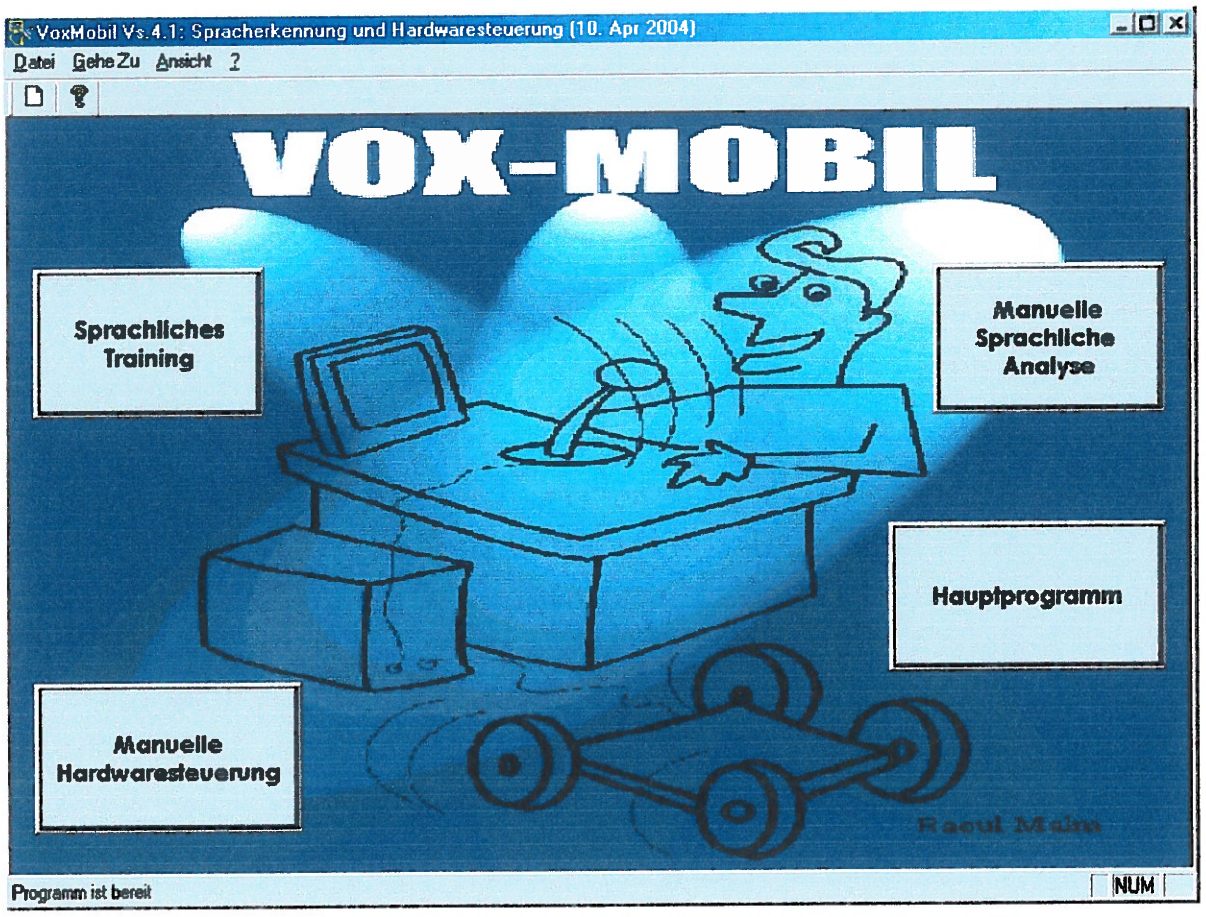

Abb. 24: Hauptseite des Programms „Vox-Mobil"

5.2 Manuelle Sprachliche Analyse

In diesem Unterprogramm besteht die Möglichkeit ein Sprachsignal zu analysieren und die Resultate pro Analyseschritt grafisch anzuzeigen. Das Sprachsignal kann dabei wahlweise entweder von einer Wave-Datei geladen oder über ein Mikrofon eingegeben werden. Daraufhin wird eine Grafik mit den Samplewerten angezeigt. Als Beispiel wird das Wort „Halt" verwendet, das nach den Einstellungen von der Abb. 13 im Unterpunkt 4.2.1 folgendermaßen angezeigt wird:

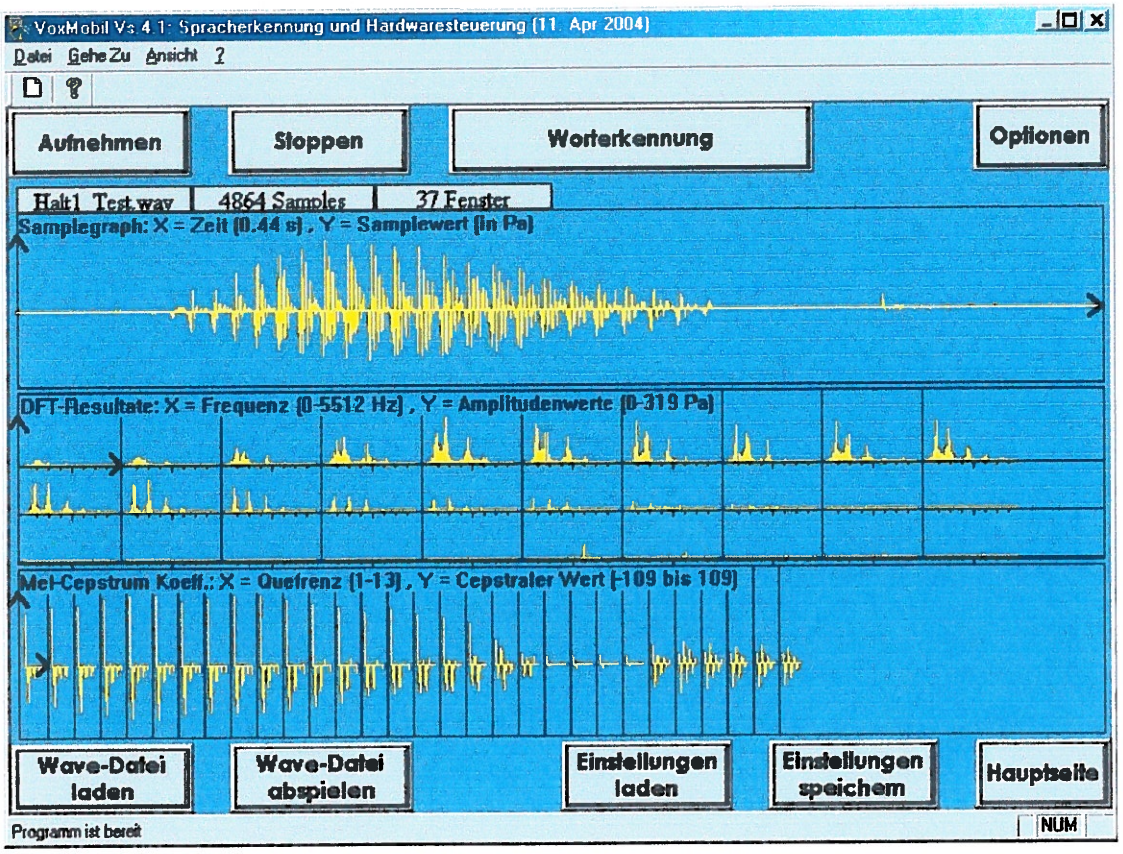

Abb. 25: Manuelle Sprachliche Analyse: Das Wort „Halt" wird im Zeit-, im Frequenz- und im Kepstralbereich visualisiert.

5.2.1 Optionen

Drückt der Benutzer die Schaltfläche Optionen öffnet sich ein Dialog, der Einstellungsmöglichkeiten bietet. Für die Aufnahme von Sprachsignalen über ein Mikrofon kann die Abtastrate zwischen drei Werten gewählt werden: 11025 Hz, 12800 Hz und 22025 Hz. Mithilfe der Angabe des Ruhepegels kann auf die Hintergrundgeräusche reagiert werden.
Die Fenstergröße kann auf 256 oder 512 Samples eingestellt werden. Die Zeitspanne eines Fensters berechnet sich folgendermaßen: $\Delta t = \frac{\text{Anzahl der Samples}}{\text{Abtastrate}}$. Im Bereich der MFCC- Berechnungen kann zudem die Anzahl der verwendeten Koeffizienten pro Fenster festgelegt werden.
Für die Fourier-Transformation kann zwischen dem Rechteck-, dem Hamming oder dem Kaiser-Bessel-Fenster gewählt werden. Mit dieser Auswahlmöglichkeit kann der Einfluss von Fensterfunktionen auf die Frequenzbestimmen aufgezeigt werden.
In dem hervorgehobenem Bereich „Grafische Darstellung" besteht die Möglichkeit Grafiken für die Frequenzen, die Mel-Filterbänke, die Logarithmen oder die MFC-Koeffizienten anzuzeigen. Der Bereich „Worterkennung", bietet die Anpassung der beiden Akzeptanzkriterien für die Abstandsberechnung.
Möchte der Benutzer ein aufgenommenes Sprachsignal in einer Wave-Datei sichern, so kann er dies und den Namen der Datei im Bereich „Weitere Einstellungen" angeben.

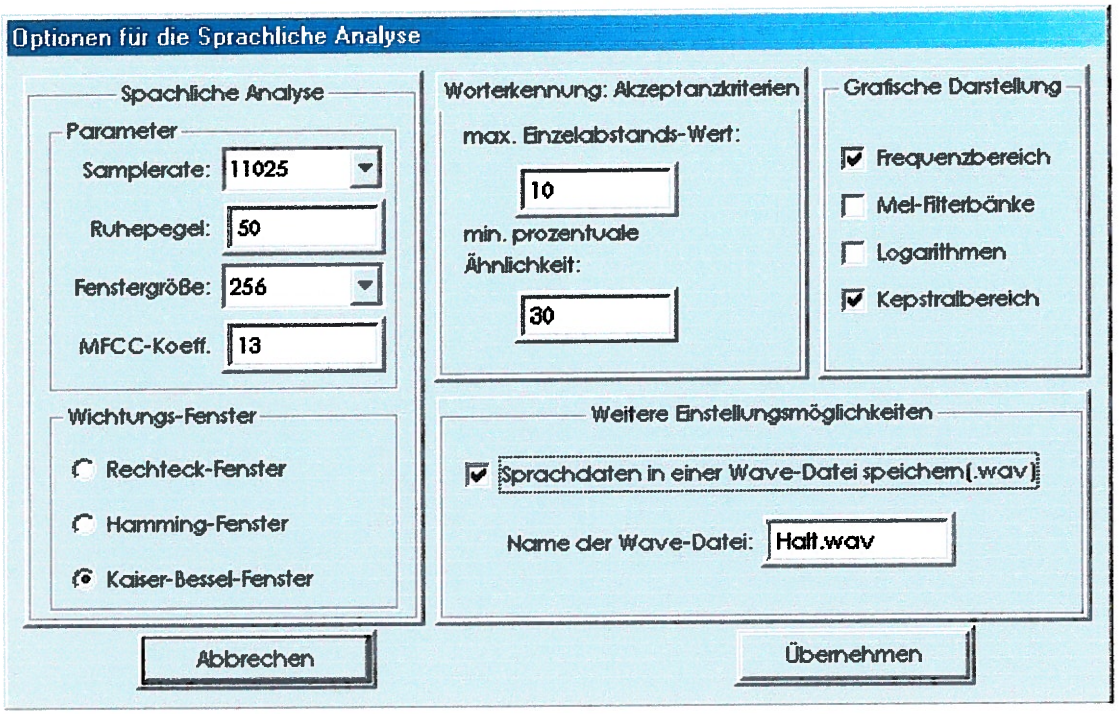

Abb. 26: Einstellungsmöglichkeiten für die Sprachliche Analyse

5.2.2 Worterkennung

Wird die Schaltfläche „Worterkennung" gedrückt, erscheint ein Dialogfenster, das die einzelnen Ergebnisse der Abstandsberechnung auflistet und dem gesprochenem Wort einen Befehl entweder zuweist oder das Wort nicht erkennt. Beispielsweise wird in der Abb. 10 des Unterpunktes 3.4 die Resultate für das Wort „Halt" angezeigt. Die Einzelabstände zu den gespeicherten Prototypen, die zu einer der fünf Kategorien gehören, sind in den entsprechenden Listenfeldern zu finden. Der geringste Abstand ist mit einem Sternchen versehen. Unter den Listenfeldern befinden sich die gemittelten Abstände aller Prototypen pro Kategorie zu dem gesprochenem Wort. In der Klammer neben den gemittelten Abständen steht jeweils ein prozentualer Wert, der die Wahrscheinlichkeit angibt inwieweit der Steuerbefehl dem gesprochenem Wort, im Vergleich zu den anderen möglichen Befehlen, entspricht. Letztendlich wird das erkannte Wort angezeigt.

5.3 Sprachliches Training

Die Seite „Sprachliches Training" ermöglicht das Anlegen und Entfernen von Prototypen für die Steuerbefehle. Über die Schaltflächen „Aufnehmen" kann man ein Wort über ein Mikrofon hineinsprechen. Falls sich das Wort in einer Wave-Datei befindet oder ein schon vorhandener Prototyp auf der Festplatte existiert, können diese über die Schaltfläche „Laden" hinzugefügt werden. Bevor der Prototyp eines Sprachsignals automatisch hinzugefügt wird, erscheint ein kleines Dialogfenster, das den gemittelten Abstand zu den Prototypen aus der entsprechenden Kategorie enthält und den Benutzer um eine Bestätigung bittet. Damit sich der Anwender das zuvor hineingesprochene Wort näher analysieren kann, wird es im Zeit- und im Kepstralen Bereich visualisiert. In den Abbildungen 14 und 15 wird das Wort „Zurück" hinzugefügt.

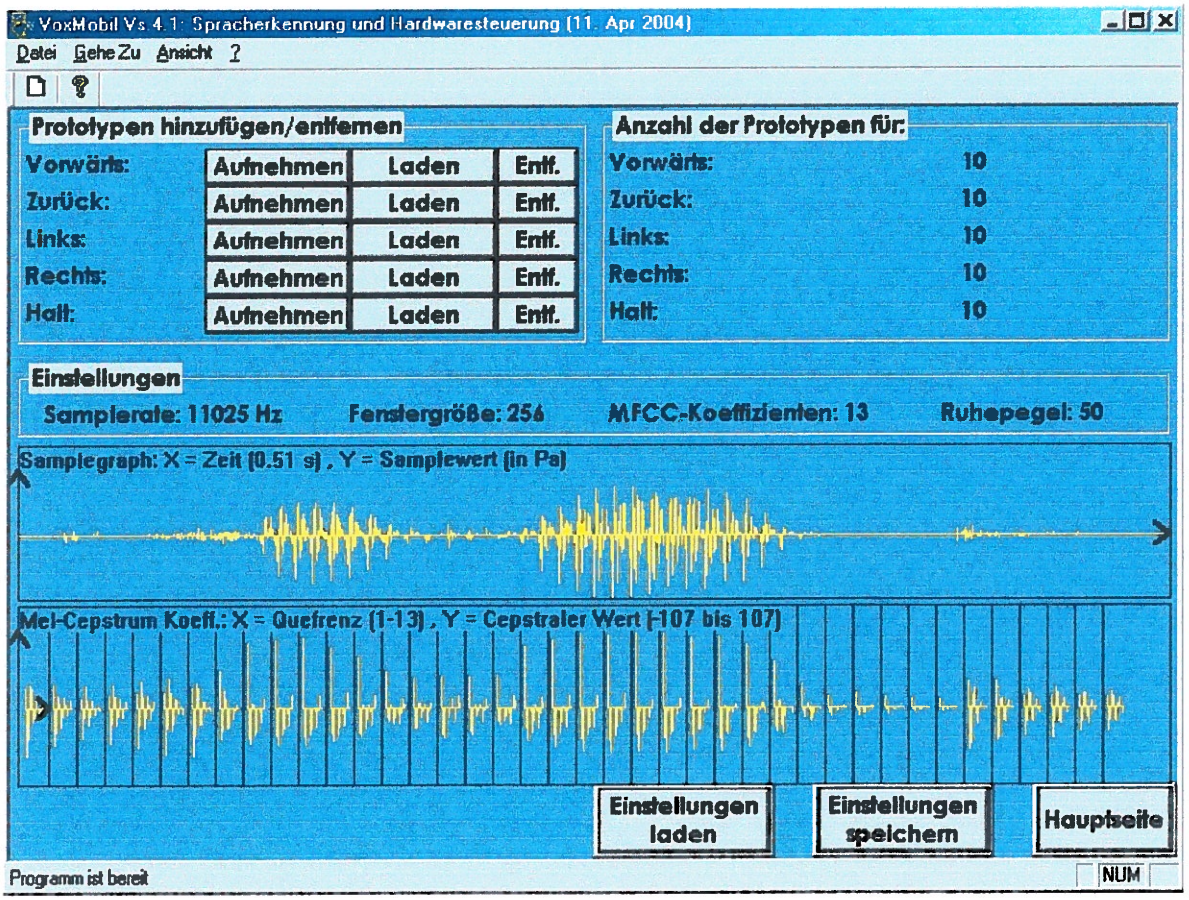

Abb. 27: Sprachliches Training: Die zwei Graphiken stellen das gesprochene Wort „Zurück" im Zeit- und im Kepstralbereich dar.

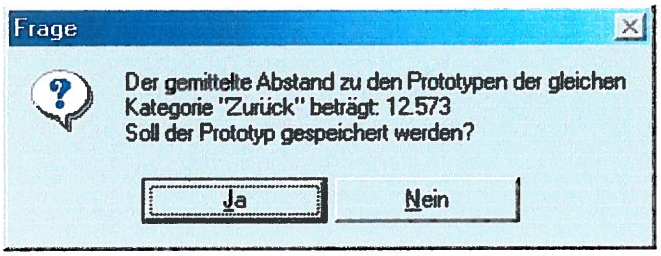

Abb. 28: Dialogfenster mit einer Frage

5.4 Manuelle Hardwaresteuerung

In diesem Unterfenster können die Unipolaren Schrittmotoren angesteuert werden. In dem Bereich „Teste Motoren" kann man die einzelnen Motoren separat voneinander drehen lassen. Über „Einstellungen" kann die Basisadresse des Druckerports und die Geschwindigkeit des Fahrzeuges, indem man die Zeitdauer zwischen zwei Impulsen verändert, angepasst werden. Das Statusfeld gibt an wie und mit welcher „Geschwindigkeit" sich die einzelnen Motoren und das Fahrzeug bewegen. In der Abb. ist der Status für eine Vorwärts-Rechts-Bewegung dargestellt. Mittels des Steuerkreuzes kann der Anwender das Fahrzeug lenken.

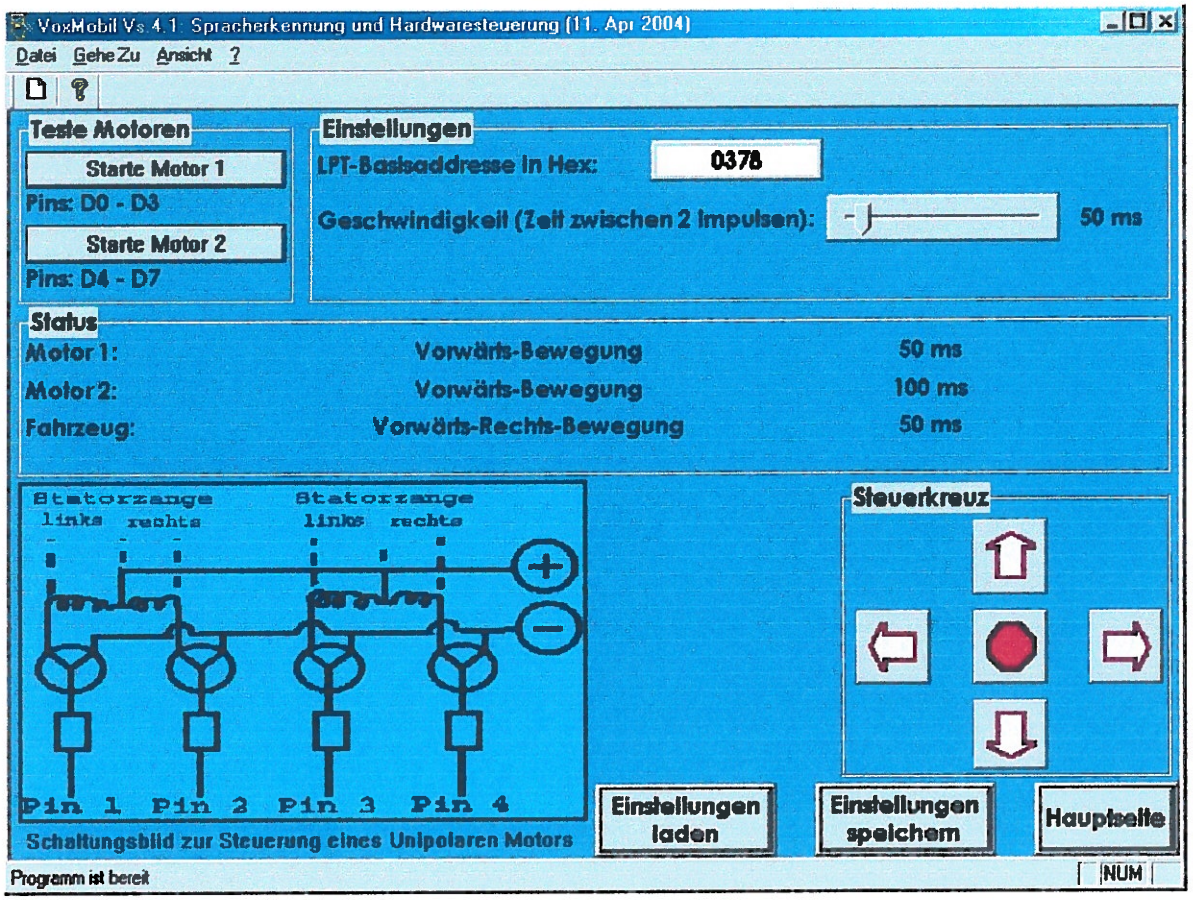

Abb. 29: Manuelle Hardwaresteuerung: Das Fahrzeug bewegt sich momentan vorwärts

5.5 Hauptprogramm

Das Hauptprogramm verbindet die Funktionalität der Spracherkennung und der Fahrzeugsteuerung. Während über ein Mikrofon aufgenommen wird, läuft ein Samplegraph mit. Aus dem gesamten Sprachfluss werden zuerst die Grenzen eines möglichen Wortes gesetzt (senkrechte schwarze Striche) und dann wird das Sprachsignal nach den Kriterien der Spracherkennung analysiert. Das Ergebnis wird angezeigt und dementsprechend das Fahrzeug bewegt. In der Abb. 17 hat der Anwender das Wort „Vor" in ein Mikrofon gesprochen und das Fahrzeug bewegt sich vorwärts.

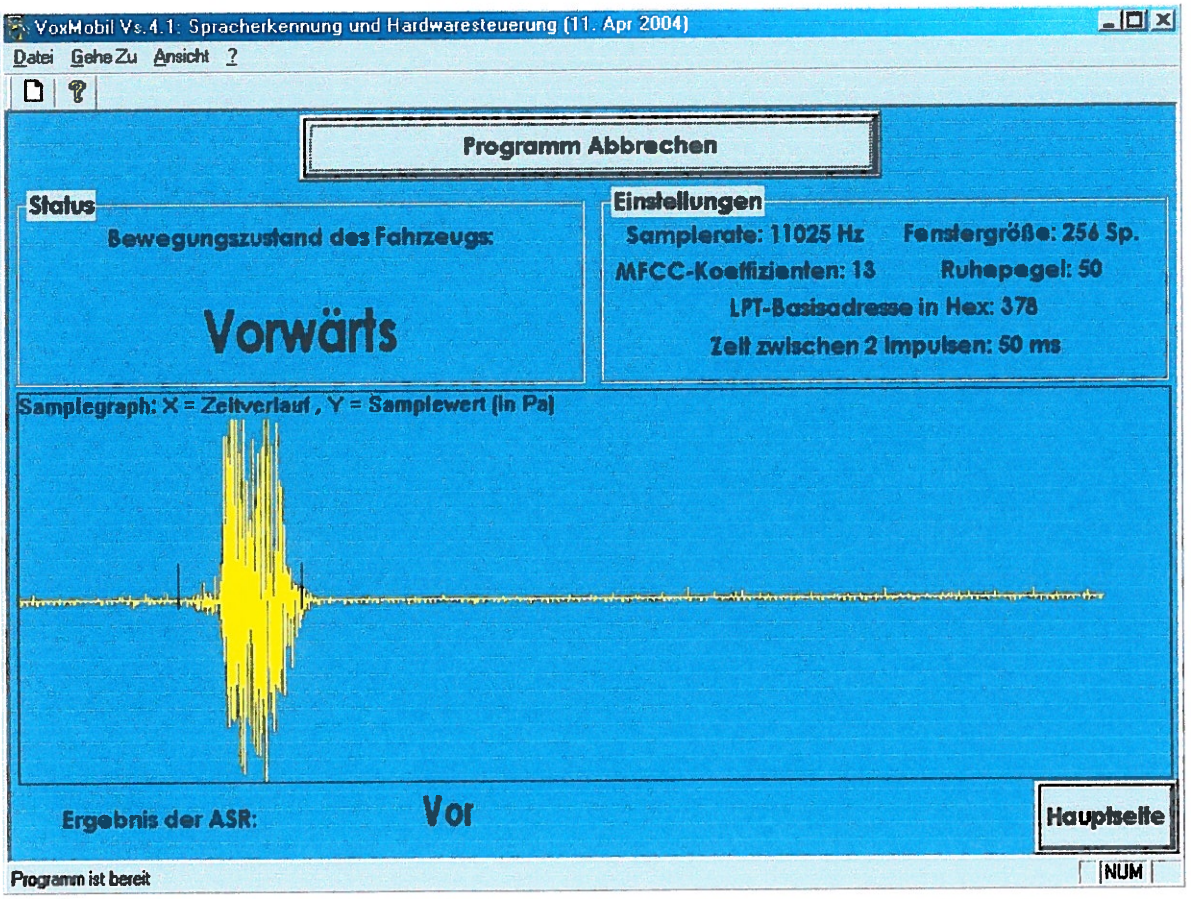

Abb. 30: Hauptprogramm: Das Wort „Vor" wurde erkannt und das Fahrzeug bewegt sich vorwärts.

6. Technik und Fahrzeug

Um mein Fahrzeug anzutreiben werden zwei unipolare Schrittmotoren verwendet, die links und rechts an der hinteren Achse des Chassis angebracht sind. Mit Hilfe von acht Transistoren, die zur Verstärkung des Laststroms dienen, wird die Steuerung realisiert.

6.1 Vom Mikrofon zum LPT-Port

Wenn die Ergebnisse der automatischen Spracherkennung einen Befehl erkannt und zugeordnet haben, wird dieser von unserem Programm ausgeführt, indem die acht Pins der parallelen Schnittstelle (Druckerport) in der richtigen Reihenfolge aktiviert bzw. deaktiviert werden. Dazu dient beispielsweise folgender Quellcode für die Vorwärts-Bewegung:

```
void CHardware::ProgramToGoForwards()
{
/*
 * Methode: ProgramToGoForwards
 * Funktion: Es werden alle Pins des Druckerports angesteuert, sodass sich
        das Fahrzeug im Vollschrittmodus bewegt
 * Fehler: Dieser Quellcode ist nicht unter Windows 2000, NT und XP lauffähig, da diese
        Betriebssysteme den direkten Zugriff auf die Ports verhindern.
*/

        int i=4; //lokale Variable i
        _outp(uiPortAddress+2, 0);    //Basisaddresse +2 => Stromrichtung = Output
        _outp(0x02F8+4, 2);           //serieller Port: blaues Vorderlicht wird aktiviert

        //Die while-Schleife wird nur unterbrochen, wenn ein Unterbrechungs-
        //Ereignis von dem Hauptthread aktiviert wurde.
        while(WaitForSingleObject(eStartInterrupt, 0)!=WAIT_OBJECT_0)
        {
                if(i>=(INT_MAX-1)) //verhindert einen Speicherüberlauf
```

```
                i=4;
            //die Liste iForwardSteps enthält die einzelnen Schritte//
            _outp(uiPortAddress, iForwardSteps[i%4]);
            Sleep(iDelayOfMotors); //eine kurze Zeitspanne wird gewartet
            i++;
        } //Ende while-Schleife

        _outp(0x02F8+4, 0);  //blaues Vorderlicht wird wieder deaktiviert
} //Ende der Methode ProgramToGoForwards
```

6.2 Vom LPT-Port zum Schrittmotor

Die Pins des LPT-Ports sind mit unserer Schaltung über ein Kabel verbunden , bestehend aus acht NPN-Transistoren. Jeder einzelne Transistor kann durchgeschalten werden, wodurch ein Laststrom (max. 50mA) durch die Schrittmotoren fließt (Impulsgebung). Durch eine verschieden schnelle Impulsgebung für die einzelnen Schrittmotoren, ist das Fahrzeug auch in der Lage Kurven zu fahren.

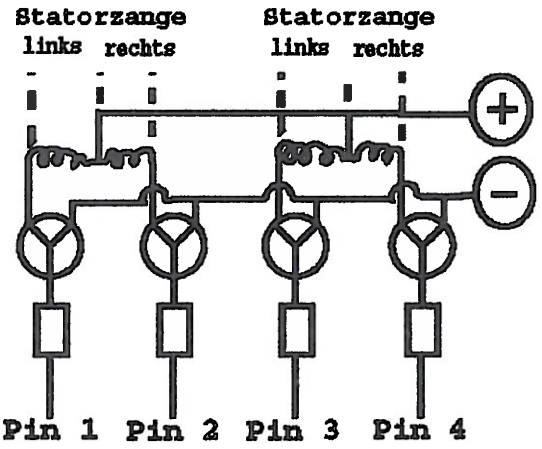

Abb. 31: Schaltung zur Steuerung eines Unipolaren Schrittmotors

Abb. 32: verwendeter Unipolarer Schrittmotor

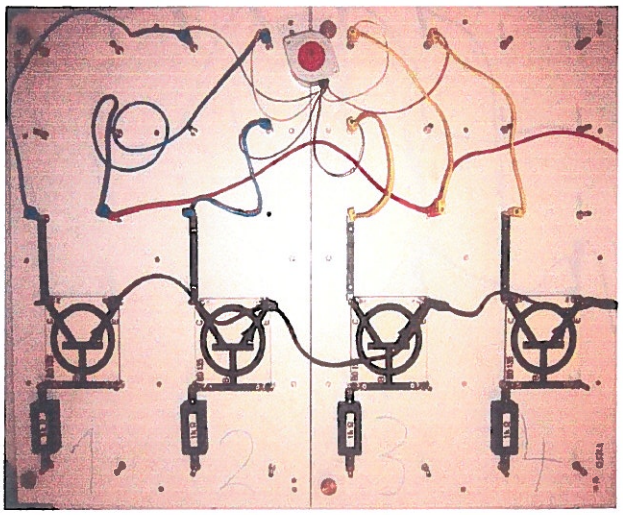

Abb. 33: Schaltungsbrett zur Steuerung eines Unipolaren Schrittmotors

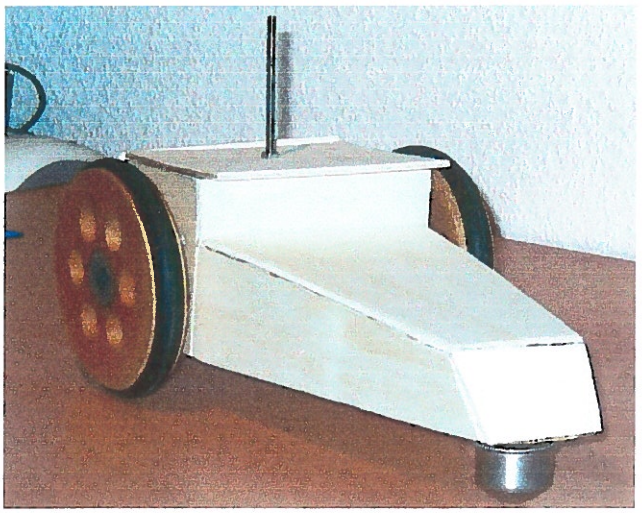

Abb. 34: selbstgebautes Fahrzeug

7. Verbesserungsmöglichkeiten

Verbesserungen können im Bereich der Filterung erreicht werden. Beispielsweise lassen sich Filter für die Glättung einfügen oder um Hintergrundgeräusche aus dem Sprachsignal zu entfernen. Geräuschsfilter können aber nicht ohne weiteres verwendet werden, da sie immer bestimmte Frequenzen unterdrücken. Daher sind diese Filter nur nützlich, wenn die Spracherkennung an Orten durchgeführt wird, die gleichbleibende Geräusche aufweisen, z.B. der Motorlärm im Inneren eines fahrenden Pkws. Aus dem Grund der Mobilität habe ich keine Geräuschsfilterung in meinem Programm implementiert.

Neben der Merkmalsgewinnung mittels des Mel-Kepstrum Verfahrens gibt es noch die Lineare Prädiktion (Lineare Vorhersage). Dabei versuch man die Übertragungsfunktion des Vokaltraktes vorherzusagen (zu modellieren). Es wird versucht einen Samplewert aus den vorangegangenen Samplewerten

darzustellen. Somit erhält man Einstellungsparameter (Prädiktionskoeffizienten), die als Merkmale verwendet werden.

8. Zusammenfassung der Ergebnisse

Mein Programm „Vox-Mobil" ermöglicht ein praktisches Anwendungsbeispiel für eine Mensch-Maschine Kommunikation mittels der Sprache. Es kann einzelne Befehlswörter erkennen und daraufhin ein Modellfahrzeug steuern.

Die Erkennungsraten des Programms hängen hauptsächlich vom Sprecher und von den Hintergrundgeräuschen ab. In einer Umgebung, in der es nicht allzu laut ist, liegen die Erkennungsraten für einen auf das Programm antrainierten Benutzer bei ca. 80 %. Diese Prozentzahl unterliegt Schwankungen in Bezug zur mehr oder minder deutlichen Sprechweise des Anwenders. Versucht nun ein anderer Benutzer das Programm auszuführen, ohne eigene Prototypen angelegt zu haben, sinkt die Erkennungsrate deutlich. Dies ist mit der unterschiedlichen Betonung, Sprechweise und der unterschiedlichen Grundfrequenz eines jeden Anwenders zu begründen. Das Programm ist daher nicht sprecherunabhängig. Eine weitere Problematik stellen die Hintergrundgeräusche dar. Nach einigen Tests hat sich herausgestellt, dass die „Verunreinigung" des Sprachsignals durch Hintergrundgeräusche kein Problem darstellt, sondern die dadurch resultierende falsche Wortgrenzenerkennung. Die richtige Wortgrenzensetzung durch das Programm ist von wichtiger Bedeutung, da unterschiedliche Anfangs- und Endgrenzen zu Fehlern bei dem Vergleich zweier Prototypen mittels der Dynamischen Zeitverzerrung führen. Daher habe ich in meinem Programm eine Funktion implementiert, die das zur Analyse freigegebene Sprachsignal nochmals auf richtige Wortgrenzensetzung untersucht. Dadurch konnte die Erkennungsrate auch bei lauten Hintergrundgeräuschen gesteigert werden.

D Anhang

D.1 Sprechererkennung durch phonetischen Fingerabdruck

Der Artikel "Phonetischer Fingerabdruck" aus dem Focus 2/2004 zeigt die Möglichkeit auf, mittels der Stimmenanalyse Video- und Tonbandaufnahmen als authentisch zu identifizieren. Des Weiteren ist es machbar Aussagen über die körperliche Verfassung eines Menschen zu treffen, über Hintergrundgeräusche auf den Aufnahmeort zu schließen oder eine Manipulation des Materials festzustellen. Beispielsweise ist festgestellt worden, dass die Tonbandaufnahmen Osama bin Ladens seit dem 11. September 2001 authentisch waren. Der Ort der Auzeichnung einer der Tonbandaufnahmen kann sogar auf eine bestimmte Region der Welt eingeschränkt werden, da im Hintergrund das Gezwitscher eines bestimmten Vogels herausgefiltert wurde.

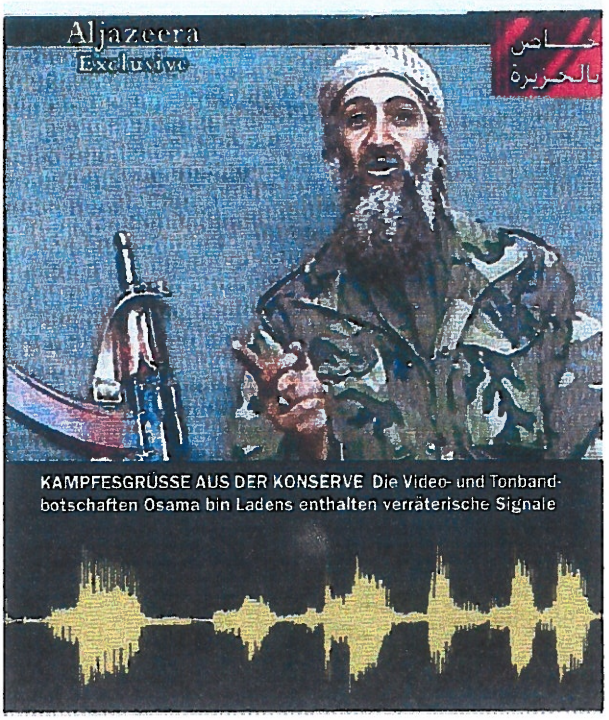

Abb. 22: oben: Foto aus einer Videobandaufnahme mit Osama bin Laden, Unten: Sprachsignal in Samplewerten

D.2 Herleitung der Fourierkoeffizienten a_k und b_k

geg: Jede komplexe periodische Funktion lässt sich folgendermaßen darstellen:

$$f(t) = \sum_{k=0}^{\infty} (a_k * \cos(\omega_k t) + b_k * \sin(\omega_k t))$$

ges: die spektralen Komponenten a_k und b_k

Lsg: Die Herleitungen für a_k und b_k sind prinzipiell identisch. Daher werden die Schritte nur für die Fourierkoeffizienten a_k aufgezeigt. Um b_k zu berechnen muss im ersten Schritt mit $\sin(\omega_k t)$ multipliziert werden und dann ebenfalls von $-T/2$ bis $+T/2$ über t integriert werden. Nun aber zur Herleitung der a_k - Koeffizienten, die aus vier Schritten besteht.

1.) beidseitige Multiplikation mit $\cos(\omega_k t)$:

$$f(t) * \cos(\omega_k t) = \sum_{k=0}^{\infty} (a_k * \cos^2(\omega_k t)) + \sum_{k=0}^{\infty} (b_k * \sin(\omega_k t) * \cos(\omega_k t))$$

2.) Integration mit $\int_{-T/2}^{+T/2}$ über t :

$$\int_{-T/2}^{+T/2} f(t) * \cos(\omega_k t)\, dt = \int_{-T/2}^{+T/2} \sum_{k=0}^{\infty} (a_k * \cos^2(\omega_k t))\, dt + \int_{-T/2}^{+T/2} \sum_{k=0}^{\infty} (b_k * \sin(\omega_k t) * \cos(\omega_k t))\, dt$$

Das folgende Integral ergibt immer Null:

$$\int_{-T/2}^{+T/2} \sin(\frac{2\pi k t}{T}) * \cos(\frac{2\pi k t}{T}) = 0$$

Für den Beweis verwendet man die trigonometrische Identität

$$\sin a * \cos b = \frac{1}{2}[\sin(a+b) + \sin(a-b)]$$

und rechnet dann das Integral aus. Somit lässt sich schreiben:

$$\int_{-T/2}^{+T/2} \sum_{k=0}^{\infty} (b_k * \sin(\omega_k t) * \cos(\omega_k t))\, dt$$

$$= \sum_{k=0}^{\infty} \int_{-T/2}^{+T/2} (b_k * \sin(\omega_k t) * \cos(\omega_k t))\, dt = 0$$

Unter dieser Berücksichtigung lässt sich schreiben:

$$\int_{-T/2}^{+T/2} f(t) * \cos(\omega_k t)\, dt = \int_{-T/2}^{+T/2} \sum_{k=0}^{\infty} (a_k * \cos^2(\omega_k t))\, dt$$

$$= \sum_{k=0}^{\infty} a_k * \int_{-T/2}^{+T/2} (\cos^2(\omega_k t))\, dt$$

3.) Löse das Integral $\int_{-T/2}^{+T/2} \cos^2(\omega_k t)\, dt$.

Dabei wird folgende Fallunterscheidung gemacht:

Integralberechnung für

Fall 1: $k = 1,2,3.....\infty$

Fall 2: $k = 0$

Fall 1: $k = 1,2,3.....\infty$

Die allgemeine Form der Partiellen Integration lautet:

$$\int_a^b u(x) * v'(x)\, dx = \left[u(x) * v(x)\right]_a^b - \int_a^b u'(x) * v(x)\, dx$$

Diese Regel wird nun auf $\int_{-T/2}^{+T/2}\cos^2(\omega_k t)\,dt$ angewandt und man erhält:

$$\int_{-T/2}^{+T/2}\cos^2(\omega_k t)\,dt = \left[\frac{1}{\omega_k} * \cos(\omega_k t) * \sin(\omega_k t)\right]_{-T/2}^{+T/2} + \int_{-T/2}^{+T/2}\sin^2(\omega_k t)\,dt$$

, mit $v' = \cos(\omega_k t) \qquad v = 1/\omega_k * \sin(\omega_k t)$

$$u = \cos(\omega_k t) \qquad u' = -\omega_k * \sin(\omega_k)$$

Nach diesem Schritt wird auch ersichtlich, warum die oben genannte Fallunterscheidung von Nöten ist.

Da $\sin(\pm\frac{\omega_k T}{2}) = \sin(\pm\frac{2\pi k T}{2T}) = \sin(\pm\pi k) = 0$ für $k = 0,1,2....\infty$ ist, gilt:

$$\left[\frac{1}{\omega_k} * \cos(\omega_k t) * \sin(\omega_k t)\right]_{-T/2}^{+T/2} = 0$$

Somit lässt sich insgesamt schreiben:

$$\int_{-T/2}^{+T/2}\cos^2(\omega_k t)\,dt = \int_{-T/2}^{+T/2}\sin^2(\omega_k t)\,dt$$

Nach dem Winkelpythagoras $\cos^2 + \sin^2 = 1$ folgt für $\int_{-T/2}^{+T/2}\sin^2(\omega_k t)\,dt$:

$$\int_{-T/2}^{+T/2}\sin^2(\omega_k t)\,dt = \int_{-T/2}^{+T/2}(1-\cos^2(\omega_k t))\,dt$$

$$= \int_{-T/2}^{+T/2}1\,dt - \int_{-T/2}^{+T/2}\cos^2(\omega_k t)\,dt = T - \int_{-T/2}^{+T/2}\cos^2(\omega_k t)\,dt$$

Dadurch erhält man:

$$\int_{-T/2}^{+T/2}\cos^2(\omega_k t)\,dt = T - \int_{-T/2}^{+T/2}\cos^2(\omega_k t)\,dt$$

$$2 * \int_{-T/2}^{+T/2} \cos^2(\omega_k t)\, dt = T$$

$$\Rightarrow \underline{\int_{-T/2}^{+T/2} \cos^2(\omega_k t)\, dt = \frac{T}{2}} \quad \text{für } k = 1,2,3.....\infty$$

Fall 2: Falls $k = 0$ gilt:

$$\int_{-T/2}^{+T/2} \cos^2(\omega_0 t)\, dt = \int_{-T/2}^{+T/2} \cos^2\left(\frac{2\pi * 0 * t}{T}\right) dt$$

$$\underline{= \int_{-T/2}^{+T/2} 1\, dt = T} \quad \text{für } k = 0$$

4.) Die Rückeinsetzung von $\frac{T}{2}$ bzw. T in die Gleichungen:

nach **Fall 1**: $k = 1,2,3.....\infty$

$$\int_{-T/2}^{+T/2} f(t) * \cos(\omega_k t)\, dt = \sum_{k=1}^{\infty} a_k * \frac{T}{2}$$

$$\sum_{k=1}^{\infty} a_k = \frac{2}{T} \int_{-T/2}^{+T/2} f(t) * \cos(\omega_k t)\, dt$$

$$\Rightarrow \underline{a_k = \frac{2}{T} \int_{-T/2}^{+T/2} f(t) * \cos(\omega_k t)\, dt} \quad \text{für } k = 1,2,3....\infty$$

nach **Fall 2**: $k = 0$

$$\int_{-T/2}^{+T/2} f(t) * \cos(\omega_0 t)\, dt = a_0 * T$$

Wegen $\cos(\omega_0 t) = \cos(0) = 1$ lässt sich schreiben:

$$\int_{-T/2}^{+T/2} f(t)\, dt = a_0 * T$$

$$=> \quad a_0 = \frac{1}{T} \int_{-T/2}^{+T/2} f(t)\, dt$$

a_0 entspricht somit dem Mittelwert der Funktion $f(t)$.

E Literaturverzeichnis

[Ba97] Bager, Jo: Sprache als Forschungsobjekt und Eingabemedium; In c`t Seite 284 bis 294; Heft 4, 1997

[Bu03] Butz, Tilman: Fouriertransformation für Fußgänger, 3.Auflage; B.G. Teubner Stuttgart; 2003

[Ch98] Chapman, Davis: Visual C++ 6 in 21 Tagen; Markt&Technik München, 1998

[Eh93] Ehlers, Frank: Hörhilfe-Spracherkennung aus Wav-Dateien; In c`t Seite 186 bis 194; Heft 11, 1993

[EiCo00] Eisenberg, Gunnar; Coll, Yorck Hernandez: Programmierprojekt; TU-Berlin, 2000

[He99] Fritz, Hermann: Wissenswertes zur Schallanalyse; 2003; Link:http://www.unet.univie.ac.at/~a7425519/Skripten/Schallanalyse.html

[Ja02] Jahn, Michael: Grundlagen-Voicescanner;HTW Dresden, 2002 Link: http://www.informatik.htw-dresden.de/~iwe/Belege/Jahn/

[Lo99] Louis, Dirk: Jetzt lerne ich Visual C++ 6; Markt&Technik München, 1999

[Moe97] Möbius, Bernd: Signalformen und Sprachsignalanalyse; IMS, Universität Stuttgart;1997

[Ni03] Niebergall, Patrick: Einführung in die automatische Spracherkennung; Referat;Westfälische Wilhems-Universität Münster, 2003

[RuRo00] Ruckdäschel, Anja A.; Rockinger, Stephan M.: Spracherkennung Phonetik/Phonologie; Seminararbeit;Universität Regensburg, 2000

[1-Sch99] Schweiger, A.: Messtechnik und Datenverarbeitung; Skript zur Vorlesung

[2-Sch99] Schwarz, Sven: Spracherkennung einzelner Befehlsworte zur Steuerung eines Roboters; Universität Kaiserslautern, 1999

[To01] Toptsis, Ioannes: Geräuschunterdrückung für Spracherkennungssysteme im Fahrzeug; Diplomarbeit; Universität Bielefeld, 2001

[UEN03] Klassifikation von Mustern; Universität Erlangen-Nürnberg, 2003
Link: http://www5.informatik.uni-erlangen.de/ MEDIA/nm/ klassifikation-von-mustern/m03_6.pdf

[Wa91] Walther, Dominik: Spracherkennung mit Hilfe Neuronaler Netze; Jugend forscht 1991

[Wi00] Willett, Daniel: Beiträge zur statistischen Modellierung und effizienten Dekodierung in der automatischen Spracherkennung;Dissertation;Gerhard-Mercator-Universität, 2000

[Wo00] Wons, Holger: Studienarbeit zur Kosinus und Fourier Transformation; Universität Mannheim, 2000

[Zi97] Zinke, Joachim: Verfahren zur Sprechererkennung; FH Giessen-Friedberg, 1997
Link: http://monet.fh-friedberg.de/fachbereiche/e2/telekom-labor/zinke/ digiaudi/diss/node4.htm#SECTION00211000000000000000

[We01] Wendemuth, Andreas; 3. Merkmalsextraktion; Woche2.pdf; Uni-Marburg, 2001

F Erklärung über die selbstständige Anfertigung der Arbeit

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Mainz-Kostheim, 14.07.2003 **Unterschrift:**