Fachhochschule Westküste
Hochschule für Wirtschaft & Technik

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Ziad Ghadieh

Automatische Spracherkennung zur Robotersteuerung auf dem TI DSP D.MODULE.C6713

Fachhochschule Westküste
Fachbereich Technik

Fachhochschule Westküste - University of Applied Sciences
Faculty of Engineering

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Hamburg University of Applied Sciences
Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Ziad Ghadieh

Automatische Spracherkennung zur Robotersteuerung auf dem TI DSP D.MODULE.C6713

Masterarbeit eingereicht im Rahmen der Masterpüfung
im gemeinsamen Studiengang Mikroelektronische Systeme
am Fachbereich Technik
der Fachhochschule Westküste
und
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Ulrich Sauvagerd
Zweitgutachter : Prof. Dr.-Ing Hans-Dieter Schütte

Abgegeben am 24. April 2008

Ziad Ghadieh

Thema der Masterarbeit

Automatische Spracherkennung auf dem TI DSP D.MODULE.C6713

Stichworte

Automatische Spracherkennung, Spracheingabe, Einzelworterkennung, nichtlineare zeitliche Anpassung, dynamische Programmierung, Sprachisolierung, Merkmalextraktion.

Kurzzusammenfassung

Diese Arbeit hat zum Ziel die Entwicklung eines Systems zur automatischen Spracherkennung. Das System soll zur Robotersteuerung dienen und auf dem TI DSP D.MODULE.C6713 implementiert werden. In dieser Arbeit wird die nichtlineare zeitliche Anpassung (DTW) eingesetzt. Das System ist sprecherabhängig mit der Möglichkeit zu Sprecherunabhängig erweitert zu werden. Bei der Erkennung handelt sich um eine Einzelworterkennung. Dazu wurde ein Verfahren vorgestellt, um ein Kommando aus einem Aufnahmeintervall zu isolieren. Für die Erkennung braucht das System ein Wörterbuch, oder Referenzkommandos. Aus jedem Referenzkommando werden im Trainingsmodus Merkmale extrahiert und gespeichert. Das System besitzt zwei Modi: Das Trainings- und das Erkennungsmodus. Im Erkennungsmodus extrahiert der Algorithmus die Merkmale aus dem Kommando dann verzerrt das DTW-Verfahren die aktuellen Merkmale, um sie an die Merkmale jedes Referenzkommandos anzupassen. Dabei wird die Euklidsche Distanz berechnet. Das Referenzkommando mit der kleinsten Distanz wird als ‚erkannt' erklärt. Eine Erkennungsrate von 97,5% wurde damit erreicht.

Ziad Ghadieh

Title of the master thesis

Automatic Speech Recognition on the TI DSP D.MODULE.C6713

Keywords

Automatic speech recognition, voice input, isolated word recognition, dynamic time warping, dynamic programming, speech isolation, feature extraction.

Abstract

The Aim of this thesis is to develop an automatic speech recognition system, which will control a robot and has to be implemented on the TI DSP D.MODULE.C6713. The Dynamic Time Warping algorithm (DTW) is used. The developed system is speaker dependent but can be extended to reach the speaker independence. The system recognizes isolated words. For that purpose, an algorithm is described to isolate a command from a speech interval. In order to perform the recognition, the system has a dictionary which is a set of reference commands. The system has two modes: A training and a recognition mode. In the training mode features are extracted from each reference command and saved. In the recognition mode, features of the actual command are extracted then warped by the DTW-algorithm to be aligned to the features of all reference commands. The distance between both feature groups is calculated. The recognized command corresponds to the minimal distance. A recognition rate of 97.5% has been reached.

Vorwort

Die vorliegende Arbeit wurde an der Hochschule für angewandte Wissenschaften Hamburg vom November 2007 bis April 2008 ausgeführt.

Herr Prof. Dr.-Ing. Ulrich Sauvagerd betreute die Arbeit seitens der Hochschule für angewandte Wissenschaften Hamburg und war Erstgutachter. Für seine Unterstützung, Hilfsbereitschaft und Geduld bin ich besonders dankbar.

Herrn Prof. Dr.-Ing. Hans-Dieter Schütte möchte ich für die Erstellung des Zweitgutachtens seitens der Fachhochschule Westküste Heide herzlich danken.

Schließlich danke ich allen Freunden und Verwandten, deren Unterstützung ebenfalls zum Gelingen der Arbeit beigetragen hat.

Inhaltsverzeichnis

Begriffe und Abkürzungen

Formelzeichen

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung 1

1.1 Systeme zur automatischen Spracherkennung 2

1.2 Geschichte und Stand der Technik 4

1.3 Ziel und Vorgehensweise 5

2 Verfahren zur automatischen Spracherkennung 7

2.1 Nichtlineare Zeitliche Anpassung (DTW) 7

2.2 Hidden Markov Modelle (HMM) 10

3 Entwicklung des Systems zur automatischen Spracherkennung 14

3.1 Anforderung an das System 14

3.2 Auswahl eines Verfahrens 14

3.3 Beschreibung der Systemarchitektur 16

3.4 Beschreibung der Systemkomponenten 16

3.4.1 Verarbeitung der aufgenommenen Sprache 17

3.4.1.1 Digitalisierung 18

3.4.1.2 Vorfilterung 19

3.4.2 Isolierung der Sprachkommandos 19

3.4.2.1 Rahmenbildung zur Berechnung der Energie 20

3.4.2.2 Berechnung der Sprachenergie und der Energieschwellenwerte .. 21

3.4.2.3 Kriterien zur Ermittlung des Anfangs- und Endpunktes eines Kommandos 23

3.4.3 Extraktion der Kommandomerkmale 27

3.4.3.1 Fensterung und Normierung der Abtastwerte 28

3.4.3.2 Berechnung der Merkmale eines Kommandos 30

3.4.4 Erkennungsverfahren 32

3.4.4.1 Dynamische Programmierung und Pfadeinschränkungen 34

3.4.4.2 Entscheidungskriterium in der Erkennungsstufe 36

3.5 Das Gesamtsystem 36

4 Entwicklung der Algorithmen 38

4.1 Algorithmusschritte 38

4.1.1 Filterung der Abtastwerte 38

4.1.2 Kommandoisolierung 39

4.1.3 Berechnung der Merkmale eines Kommandos 43

4.1.4 Nichtlineare zeitliche Anpassung und Distanzberechnung 44

4.2 Simulation 46

4.2.1 MATLAB-Simulation 46

4.2.2 Simulationsergebnisse 49

5 Implementierung der Algorithmen 51

5.1 Verwendete Parameter 52

5.2 Einlesen der Abtastwerte 53

5.3 Einlesen und Speichern der Referenzmerkmale 54

5.4 Das Hauptprogramm (SpeechRecognition.c) 55

5.5 Funktionsschnittstellen 56

5.5.1 Vorfilterung (Prefilter.c) 56

5.5.2 Detektierung der Endpunkte (DetectEndPoints.c) 57

5.5.3 Berechnung der Kommandomerkmale (CalcFrameFeatures.c) 58

5.5.4 Nichtlineare zeitliche Anpassung (DTW_Recognition.c) 58

6 Testdurchführung 60

7 Zusammenfassung und Ausblick 62

8 Literaturverzeichnis 65

9 Anhang 67

9.1 Flussdiagramme 67

9.1.1 Interrupt-Service-Routine, c_int11() 67

9.1.2 Hauptprogramm - Main() 68

9.1.3 FIR_HP() 76

9.1.4 DetectEndPoints() 77

9.1.5 CalcFrameFeatures() 80

9.1.6 DTW_Recognition() 81

9.2 C-Code 83

9.2.1 SpeechRecognition.c 83

9.2.2 SpeechRecognition.h 92

9.2.3 Prefilter.c 93

9.2.4 Prefilter.h 95

9.2.5 TimeWin16.h 95

9.2.6 DetectEndPoints.c 96

9.2.7 CalcFrameFeatures.c 100

9.2.8 DTW_Recognition.c 101

9.2.9 CommandSpeaker16.h 103

Begriffe und Abkürzungen

AKF	Autokorrelationsfunktion
ASR	Automatische Spracherkennung (Automatic Speech Recognition)
D.MODULE.C6713	Digital Signal Prozessor der Firma Texas Instruments
DP	Dynamische Programmierung (Dynamic Programing)
DTW	Nichtlineare zeitliche Verzerrung (Dynamic Time Warping)
Erkennungsrate	Das Verhältnis von erkannten Wörtern zu gesamten gesprochenen Wörtern in Prozent.
Frikativ	Ein Konsonant, bei dessen Bildung ein Reibelaut erzeugt wird
HMM	Hidden Markov Modelle
Kommandomerkmale	Gewonnene Merkmale aus einer gesprochenen Äußerung
LPC	Kodierung mit linearer Vorhersage (Linear Predictive Coding)
LPCC	Linear Predictive Coding Coefficients
MFCC	Mel Frequency Cepstrum Coefficients
Phonem	Kleinste unterscheidende Einheit einer Sprache
TI	Texas Instruments
RTDX	Real Time Data Exchange, Datenaustausch in Echtzeit

Formelzeichen

r	Erkennungsrate
ε	Fehlerrate
E_D	Deletion Error, Weglassenfehler
E_I	Insertion Error, Einfügefehler
E_S	Substitution Error, Verwechslungsfehler
S_P	Verarbeitungsgeschwindigkeit (Processing Speed)
N_F	Anzahl der Merkmale (Feature Number)
$\underline{x}_F$	Einzelner Vektor von Referenzmerkmalen
$\underline{R}_F$	Gesamte Referenzmerkmale oder Referenzmuster
I	Dauer des gesamten Referenzmusters
J	Dauer des gesamten Testmuster
$\underline{y}_F$	Einzelner Vektor von Testmerkmalen
$d(i,j)$	Lokale Distanz zwischen zwei Einzelnen Merkmalvektoren
$D(I,J)$	Gesamtdistanz zwischen zwei Mustern
$\underline{T}_F$	Gesamte Merkmale des zu erkennenden (Test-) Wortes
λ_V	HM Modell
π	Anfangswahrscheinlichkeit in einem HMM
$p(\underline{x} \mid i)$	Die Erzeugungswahrscheinlichkeit für $\underline{x}$ im Zustand in einem HMM
N	Anzahl der Abtastwerte
$WinShift$	Rahmenverschiebung
$WinWidth$	Rahmenbreite
$\underline{E}$	Energiewerte
E_{Min}	Minimalwert der Energie im sprachlosen Intervall
E_{Max}	Maximalwert der Energie im sprachlosen Intervall
Th_{Low}	Unterer Energieschwellenwert
Th_{High}	Oberer Energieschwellenwert
$MaxThDist$	Maximal zulässige Distanz bzw. Dauer zwischen den Stellen, an denen die Energieschwellenwerte überschritten werden
$MinPulseWidth$	Minimal zulässige Breite eines Energieimpulses
$MaxWordWidth$	Maximal zulässige Breite bzw. Dauer eines Kommandos
$MaxPulseDist$	Maximal zulässige Distanz bzw. Dauer

	zwischen zwei aufeinander folgenden Impulsen
$\underline{F}$	Koeffizienten des Glättungsfensters
$CoeffOrder$	Anzahl der Merkmale pro Rahmen = N_F
$\underline{C}_{R,i}$	i-ter Referenz-Merkmalvektor
$\underline{C}_{T,j}$	j-ter aktueller Merkmalvektor
N_{TF}	Anzahl der Rahmen der aktuellen Kommandomerkmale
N_{RF}	Anzahl der Rahmen der Referenzmerkmale eines Kommandos
$D_{Min}(N_{RF}-1, N_{TF}-1, P_{opt})$	Minimale akkumulierte Gesamtdistanz beim optimalen Pfad
D_{Min}	Minimale Gesamtdistanz
$\tilde{D}_{Min}$	Normierte minimale Gesamtdistanz
N_C	Anzahl der Referenzkommandos

Abbildungsverzeichnis

Abbildung 1-1 Einsatz der ASR zur Robotersteuerung 2

Abbildung 1-2 Eigenschaften von Systemen zur automatischen Spracherkennung 3

Abbildung 1-3 Die ASR soll für diesen Roboter an der HAW Hamburg entwickelt werden. Quelle: Hersteller Festo (Robotino) 6

Abbildung 1-4 Die ASR soll auf diesem digitalen Signalprozessor implementiert werden. Quelle: Hersteller Texas Instrument (D.MODULE.C6713) 6

Abbildung 2-1 Prinzip der Erkennung mit dem DTW-Verfahren 8

Abbildung 2-2 Prinzip der nichtlinearen zeitlichen Anpassung (DTW) 9

Abbildung 2-3 HMM-Erzeugungsmodelle in der Lern- und Testphase 11

Abbildung 2-4 allgemeines HMM und ein Beispielmodell für das Wort „haben“ 12

Abbildung 3-1 ASR-System auf der höchsten Ebene 16

Abbildung 3-2 ASR-Systemkomponenten 17

Abbildung 3-3 Verarbeitung der aufgenommenen Sprache 18

Abbildung 3-4 Isolierung eines Kommandos 20

Abbildung 3-5 Rahmenbildung und Berechnung der Energie 21

Abbildung 3-6 Detektierung Der Endpunkte eines Kommandos 23

Abbildung 3-7 Sonderfall [a] bei der Detektierung der Endpunkte 24

Abbildung 3-8 Sonderfall [b] bei der Detektierung der Endpunkte 25

Abbildung 3-9 Sonderfälle [c] und [d] bei der Detektierung der Endpunkte 25

Abbildung 3-10 Sonderfall [e] bei der Detektierung der Endpunkte 26

Abbildung 3-11 Sonderfall [f] bei der Detektierung der Endpunkte 27

Abbildung 3-12 Merkmalextraktion eines Kommandos 28

Abbildung 3-13 Fensterung und Normierung des isolierten Kommandos 30

Abbildung 3-14 Berechnung der AKF Koeffizienten 31

Abbildung 3-15 Erkennung eines Kommandos 32

Abbildung 3-16 Prinzip des DTW-Verfahrens 33

Abbildung 3-17 Prinzip der dynamischen Programmierung 34

Abbildung 3-18 Pfadeinschränkung bei der dynamischen Programmierung 35

Abbildung 3-19 Gesamtsystem zur automatischen Spracherkennung 37

Abbildung 4-1 Übersetzung der Endpunkte 42

Abbildung 4-2 Isolierung des Kommandos ‚Umdrehen' 48

Abbildung 4-3 Simulationsergebnis 49

Abbildung 4-4 Beispiel zum Verlauf der quasi Energie von einer Unterhaltung 50

Abbildung 5-1 Modularer Aufbau des ASR-Systems 51

Abbildung 5-2 Flussdiagram der Interrupt-Service-Routine zum Einlesen der Abtastwerte 53

Abbildung 5-3 Struktur der Referenzmerkmale 55

Abbildung 6-1 Benutzeroberfläche zum Durchführen von Trainings und Tests 60

Abbildung 9-1 Flussdiagramm ISR 67

Abbildung 9-2 Flussdiagram vom Hauptprogramm 68

Abbildung 9-3 Flussdiagramm FIR_HP() 76

Abbildung 9-4 Flussdiagramm DetectEndPoints() 77

Abbildung 9-5 Flussdiagramm CalcFrameFeatures() 80

Abbildung 9-6 Flussdiagramm DTW_Recognition() 81

Tabellenverzeichnis

Tabelle 3-1 HMM vs. DTW für Einzelworterkennung und kleinen Wortschatz **15**

Tabelle 4-1 ReferenzKommandos **47**

Tabelle 5-1 verwendete Parameter **52**

1 Einführung

Die automatische Spracherkennung (Automatic Speech Recognition, ASR) ist ein Bestandteil aller Anwendungen, bei denen ein Dialog zwischen Mensch und Maschine geführt werden soll. Das Ziel der automatischen Spracherkennung ist aus der gesprochenen Äußerung einen entsprechenden Wert maschinell zu gewinnen. Im Allgemeinen erzielt man mit der Spracheingabe die Informationsgewinnung aus Datenbanken, die Erfassung von gesprochenen Daten oder die Steuerung von Maschinen. Diese Einsatzbereiche findet man u. a.

- in der Industrie (Inventur, Qualitätskontrolle),
- im Büro (Archivabfragen),
- im Haushalt (Radiowecker, Namenswahl im Telefon),
- in Dialogsystemen (Buchungssysteme, Fahrplanauskunft),
- in der Ausbildung (Fremdsprachen, Hilfe bei Sprechstörungen),
- in der Medizin (Diktiersysteme),
- bei der Behindertenhilfe (Fahrzeugbedienung) und
- im militärischen Bereich [Fin07][ST95].

Die vorliegende Arbeit geht auf die Realisierung einer automatischen Spracherkennung ein, die zur Steuerung von Maschinen dient. Das zu entwickelnde System soll das gesprochene erkennen und entsprechende Signale ausgeben. Diesem System wird ein Gesamtsystem überlagert, das zur Steuerung eines Roboters dient (Abbildung 1-1). Die eingesetzten Algorithmen sollen auf einem digitalen Signalprozessor implementiert und getestet werden.

Um einen besseren Überblick über die automatischen Spracherkennung zu geben, wird in den nächsten Unterkapiteln auf die allgemeinen Eigenschaften der Spracherkennungssysteme, die Geschichte und den Stand der Technik

eingegangen. Darauf folgen die Zielsetzung und die Vorgehensweise bei der Lösung der Aufgabe.

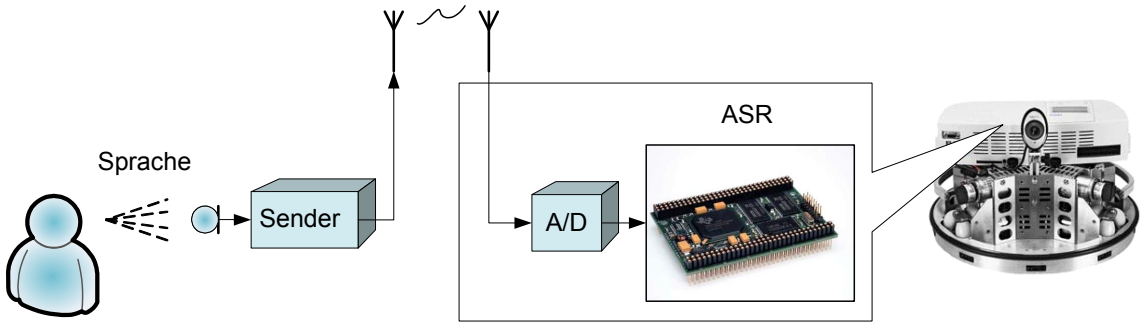


Abbildung 1-1 Einsatz der ASR zur Robotersteuerung

1.1 Systeme zur automatischen Spracherkennung

Im Allgemeinen unterscheidet man zwischen sprecherabhängigen und sprecherunabhängigen Systemen. In diesen beiden Hauptkategorien kann man jeweils zwischen folgenden Arten von Spracherkennung unterscheiden:

- Einzelworterkennung (Isolated Word Recognition). Die zugehörigen Systeme erkennen Einzelwörter oder Kommandos.
- Wortkettenerkennung (Connected Word Recognition). Hier werden Wortfolgen erkannt. Die Anzahl der zusammenhängenden Wörter ist begrenzt.
- Erkennung der kontinuierlichen Sprache (Continuous Speech Recognition). Die zugehörigen Systeme werden auch als sprachinterpretierende Systeme bezeichnet.

Alle Systeme der Spracherkennung müssen eine Lernphase durchgehen. Für sprecherabhängige Systeme bedeutet das eine Lernphase für jeden Sprecher. Für sprecherunabhängige Systeme bedeutet es eine einmalige, aber aufwendige Lernphase für alle Sprecher.

Zu den Systemeigenschaften gehören die Anzahl der zu erkennenden Wörter, die die Verarbeitungsgeschwindigkeit und die Fehlerrate.

Die Verarbeitungsgeschwindigkeit (Processing Speed, S_P) ist eine wichtige Systemeigenschaft. Die Echtzeitrealisierung hängt weit von der Verarbeitungsgeschwindigkeit ab.

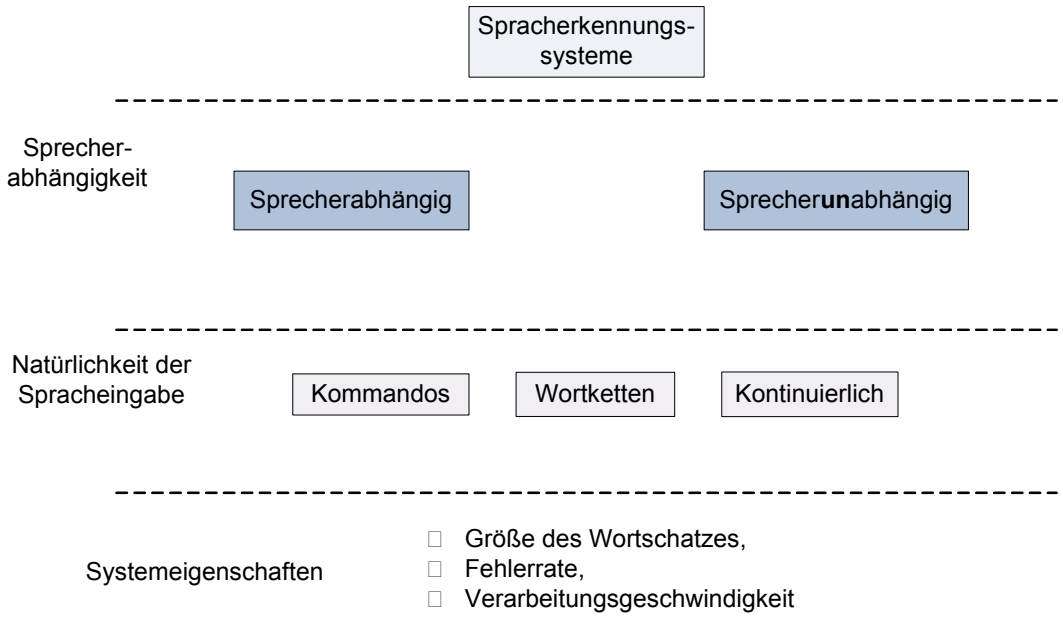


Abbildung 1-2 Eigenschaften von Systemen zur automatischen Spracherkennung

Die Fehlerrate ε setzt sich aus drei Fehlerarten zusammen:

- dem Verwechslungsfehler (Substitution Error, E_S), d.h. die falsche Erkennung von gesprochenen Wörtern,
- dem Einfügefehler (Insertion Error, E_I), d.h. das Einfügen von nicht gesprochenen Wörtern und
- dem Weglassenfehler (Deletion Error, E_D), d.h. das Weglassen von gesprochenen Wörtern.

Für die Einzelworterkennung ist nur der Verwechslungsfehler E_S relevant.

Die Fehlerrate kann man mit folgender Formel bestimmen:

$$\varepsilon = 100\% \times \left(\frac{E_S + E_I + E_D}{N_W} \right) \tag{1.1}$$

Dabei ist N_W die Anzahl der gesprochenen Wörter.

Die Erkennungsrate r ist wie folgt definiert:

$$r = 100\% - \varepsilon \tag{1.2}$$

1.2 Geschichte und Stand der Technik

Das Gebiet der automatischen Spracherkennung blickt auf eine etwa fünfzigjährige Forschungsgeschichte zurück.

In den 1960er Jahren war es möglich 10 bis 100 Einzelwörter zu erkennen. Filterbänke waren die Haupttechnik und wurden eingesetzt, um die spektralen Eigenschaften der Sprache zu gewinnen. Spektrale Muster der Einzelwörter wurden gespeichert und bei der Erkennung mit den Spektren der zu erkennenden Wörter verglichen. Damit wurden Erkennungsraten bis 90% erreicht [Sic83] [RJ04].

In den 1970er Jahren wurde die Technik der dynamischen Programmierung eingeführt. Auf dieser Technik beruht das Verfahren der nichtlinearen zeitlichen Anpassung, auch nichtlineare zeitliche Verzerrung genannt (Dynamic Time Warping, DTW), welches in der automatischen Spracherkennung eingesetzt wurde. Dazu wurden Methoden zur Beschreibung des Zeitsignals innerhalb eines Zeitfensters, wie die lineare Prädiktion (Linear Predictive Coding, LPC), eingeführt. Prototypmerkmale wurden gewonnen und gespeichert, um dann bei der Erkennung mit den Merkmalen des zu erkennenden Wortes verglichen zu werden. Mit diesen neuen Verfahren konnte man 100 bis 1000 Einzelwörter erkennen und Erkennungsraten bis 100% erreichen. Durch den Einsatz von aufwendigen Klassifikationsmethoden konnte man eine sprecherunabhängige Spracherkennung realisieren. Mit mehreren parallel laufenden DTW-Algorithmen erreichte man die Erkennung von Wortketten [Ita75][RJ04].

Ein Übergang vom dynamischen Mustervergleich und Wortprototypen zur statistischen Modellierung kam in den 1980er Jahren. Die Technik der Hidden-Markov[1]-Modelle (HMM) spielte dabei die Schlüsselrolle. Durch die Kombination von HMM und stochastischen Sprachmodellen erreichte man die Erkennung von kontinuierlicher Sprache. Damit erreichte man einen unbegrenzten Wortschatz und hohe Erkennungsraten. Die Sprachmodelle waren aber begrenzt und nicht flexibel genug [RJ04]. Auch verschiedene Versionen neuronaler Netze wurden für die Spracherkennung eingesetzt. Allerdings konnte man mit der Technik der neuronalen Netze nicht die Ergebnisse der HMM-Technik erreichen [Wik08].

Dank der steigenden Prozessorleistung im kommerziellen Bereich erschienen in den 1990er Jahren Spracherkennungssysteme, die auf dem normalen PC liefen (z.B. ViaVoice von IBM oder Sphinx von CMU). Diese Systeme unterstützten unbegrenzte Sprachmodelle und ermöglichten eine sehr gute Erkennung von einem großen Wortschatz. Seitdem wurden Fortschritte in den Semantikmodellen gemacht. Die Systeme wurden auch für die effiziente Implementierung verbessert und häufig mit TTS-Anwendungen („Text-To-Speech“) implementiert, die mehreren Eingabe- oder Steuerungsmodi unterstützen.

Aktuelle Spracherkennungssysteme, die auf einem normalen PC laufen (z.B. Dragon NaturallySpeaking Professional 9 von NUANCE), haben einen Wortschatz von über eine Million Wörter (Deutsch) und bieten die Möglichkeit, eigene Fachwörter zum Wörterbuch hinzuzufügen. Die Erkennungsgeschwindigkeit erreicht 160 Wörter pro Minute. Die Erkennungsraten erreichen 99% laut Angaben des Herstellers [Nua08].

1.3 Ziel und Vorgehensweise

Das Ziel dieser Arbeit ist ein System zu entwickeln, das eine automatische Spracherkennung zur Robotersteuerung realisiert (Abbildung 1-3). Die Realisie-

[1] Andrei Andrejewitsch Markov (1856-1922): Mathematiker, Beiträge zur Wahrscheinlichkeitstheorie und Theorie der stochastischen Prozesse.

rung soll Software-basiert sein und auf einem digitalen Signalprozessor implementiert werden (Abbildung 1-4).

Abbildung 1-3 Die ASR soll für diesen Roboter an der HAW Hamburg entwickelt werden.

Quelle: Hersteller Festo (Robotino)

Abbildung 1-4 Die ASR soll auf diesem digitalen Signalprozessor implementiert werden.

Quelle: Hersteller Texas Instrument (D.MODULE.C6713)

Zunächst werden Verfahren zur automatischen Spracherkennung vorgestellt. Danach wird das System entwickelt. Das beinhaltet die Analyse der Anforderungen an das System, die Auswahl eines Verfahrens und die Beschreibung der Systemarchitektur. Nach dem Systementwurf werden die dafür benötigten Algorithmen beschrieben und implementiert. Nach einem Systemtest folgt zum Schluss eine Zusammenfassung mit Ausblick.

2 Verfahren zur automatischen Spracherkennung

Mithilfe der Kurzzeitspektren lassen sich nur stationäre Sprachlauten, wie einzelne Vokale, klassifizieren. Sollen dagegen ganze Wörter klassifiziert werden, so muss eine bestimmte zeitliche Folge von Mustervektoren betrachtet werden, die einen gesamten Musterverlauf bildet. Hier kommen insbesondere die nichtlineare zeitliche Verzerrung für den Mustervergleich oder die Darstellung in Form der Hidden-Markov-Modelle zum Einsatz. Mit diesen Verfahren ist darüber hinaus eine einfache Erweiterung auf Ketten von Einheiten möglich, so dass prinzipiell auch Wortketten oder fließende Sprache verarbeitet werden können [Rus94].

Bei allen vorgestellten Verfahren muss eine Vorverarbeitung erfolgen. Diese Vorverarbeitung hat den Zweck, die gesprochenen Wörter zu isolieren und bestimmte Muster oder Merkmale aus den Wörtern zu gewinnen. Ferner werden die Merkmale in Einheiten geteilt, um das Problem der Variabilität in der Sprechgeschwindigkeit und der Betonung zu umgehen. Die zeitliche Länge der Wortmuster kann ja aufgrund solcher Probleme sehr stark schwanken. Die Methoden der Vorverarbeitung werden später detailliert erklärt.

In den weiteren Abschnitten dieses Kapitels werden die potenziellen Verfahren vorgestellt, ohne auf die Algorithmen einzugehen. In einem späteren Kapitel werden die Algorithmen für das ausgewählte Verfahren erklärt.

2.1 Nichtlineare Zeitliche Anpassung (DTW)

Die Erkennung von Spracheinheiten oder Wörtern lässt sich auf der Basis einer Abstandsklassifizierung realisieren. Das Muster des zu erkennenden Wortes wird mit den Mustern von Referenzwörtern verglichen. Dabei wird jeweils der Abstand bzw. die Differenz berechnet. Das Referenzwort mit dem kleinsten Abstand wird als Erkennungsergebnis ausgegeben.

Die Vorverarbeitung liefert für einen Analysezeitpunkt N_F Merkmale (Features), die zu einem Vektor $\underline{x}_F$ zusammengefasst werden:

$$\underline{x}_F = (x_1, x_2, \ldots, x_{N_F}) \tag{2.1}$$

Die zeitliche Folge dieser Analysepunkte bildet ein gesamtes Wortmuster. Für das Referenzwort wird dies $\underline{R}_F$ genannt. Seine zeitliche Dauer wird mit I bezeichnet:

$$\underline{R}_F = \{\underline{x}_{F,1}, \underline{x}_{F,2}, \ldots, \underline{x}_{F,I}\} \tag{2.2}$$

Die Referenzmerkmale werden in der Lernphase gespeichert, um mit den Merkmalen des Testwortes $\underline{T}_F$ verglichen zu werden. Dessen zeitliche Dauer wird mit J bezeichnet:

$$\underline{T}_F = \{\underline{y}_{F,1}, \underline{y}_{F,2}, \ldots, \underline{y}_{F,J}\} \tag{2.3}$$

Beide Merkmalgruppen $\underline{R}_F$ und $\underline{T}_F$ werden von der Vorverarbeitungsstufe geliefert.

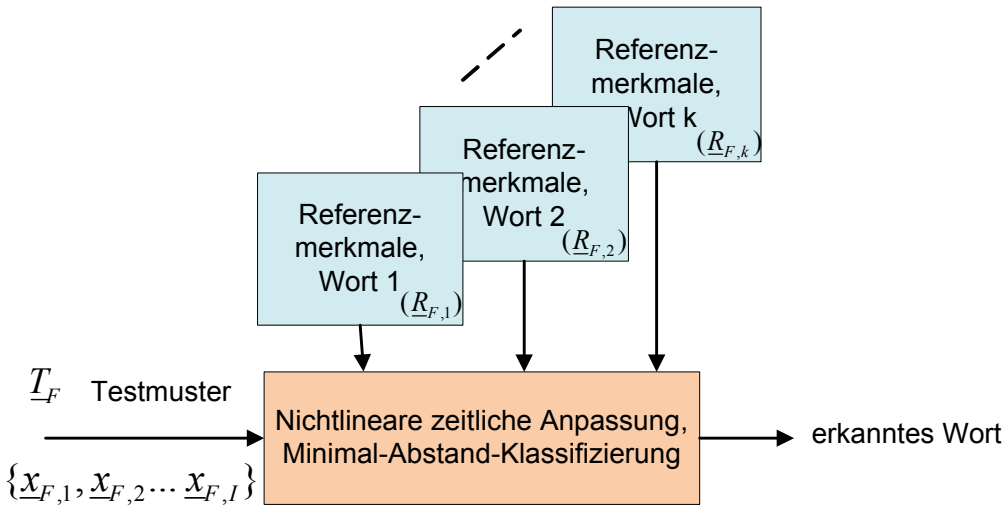


Abbildung 2-1 Prinzip der Erkennung mit dem DTW-Verfahren

Die Aufgabe des Erkennungsverfahren besteht darin, den Abstand zwischen den Merkmalgruppen $\underline{T}_F$ und $\underline{R}_F$ zu berechnen (Abbildung 2-2). Da beide Gruppen unterschiedliche Anzahl an Einzelmerkmalvektoren $\underline{x}_F$ und $\underline{y}_F$ haben, muss die eine Gruppe entweder durch Interpolation oder durch die nichtlineare zeitliche Anpassung (DTW) so groß wie die andere Gruppe gemacht werden. Die DTW-Methode wies eine bessere Erkennungsrate gegenüber der Interpolationsmethode

[Rus94]. Verschiedene Verfahren werden eingesetzt, um den Abstand zu berechnen. Darunter sind der Euklidsche quadratische Abstand, die City-Block-Distanz und die Mahalanobis-Distanz [Pla05][RL81].

Bei der Berechnung der Distanz werden die lokalen Distanzen $d(i,j)$ zwischen den Einzelmerkmalvektoren $\underline{x}_F$ und $\underline{y}_F$ berechnet. Die Gesamtdistanz $D(I,J)$ über alle Merkmale von $\underline{T}_F$ und $\underline{R}_F$ ist die Summe aller lokalen minimalen Distanzen entlang eines bestimmten Pfades. Durch Anwendung der Dynamischen Programmierung (DP) lässt sich vermeiden, dass alle Pfade berechnet werden müssen, was einen erheblichen Rechenaufwand bedeuten würde [Pla05].

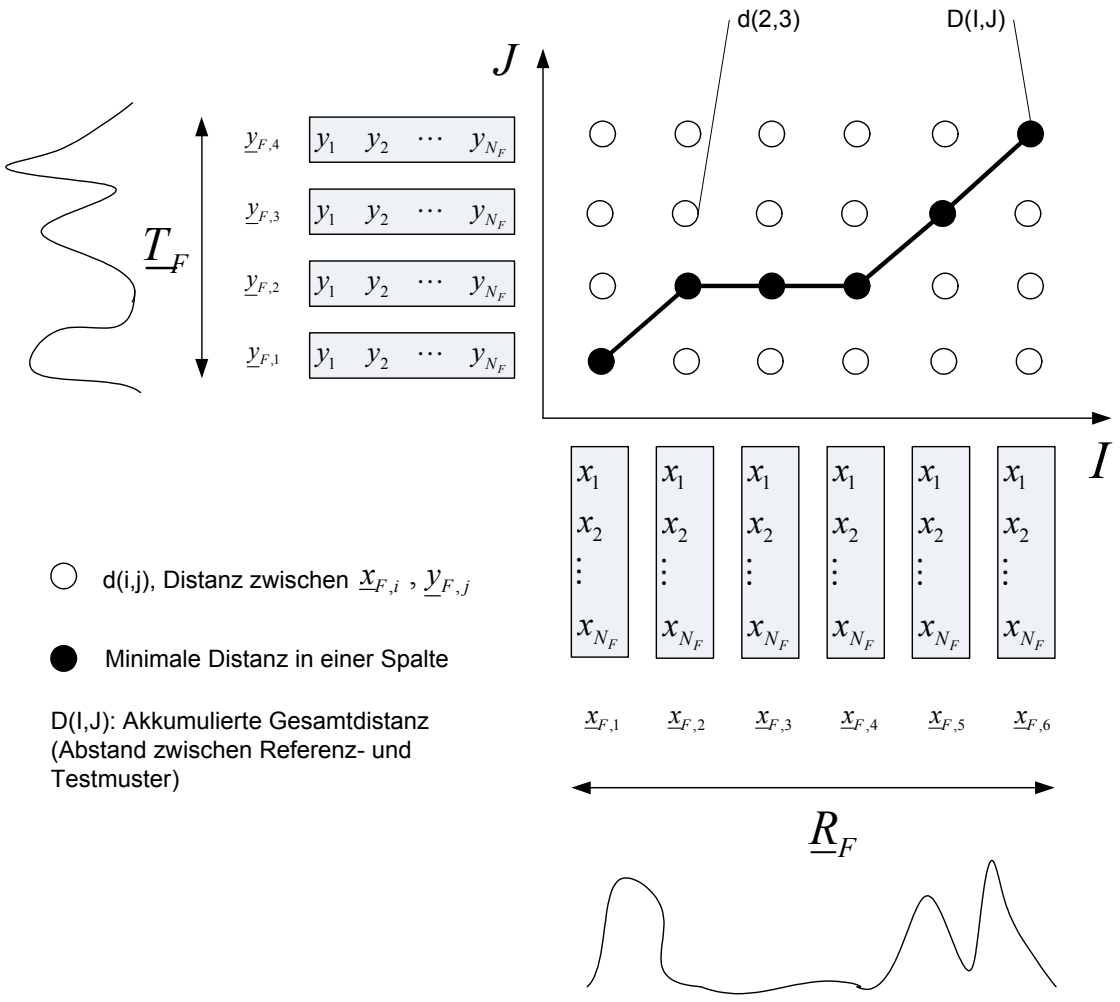


Abbildung 2-2 Prinzip der nichtlinearen zeitlichen Anpassung (DTW)

Abbildung 2-2 zeigt das Prinzip der nichtlinearen zeitlichen Anpassung anhand zweier Muster, die sich in der Dauer nur an einer Stelle unterscheiden. Das Testmuster wird mit den einzelnen Referenzmustern verglichen. Daraus folgt, dass die Rechenzeit mit der Anzahl der zu erkennenden Wörter zunimmt.

Das DTW-Verfahren lässt sich für aufbauende Entwicklungen so modifizieren, dass auch Wortketten verarbeitet werden können [Rus94]. Auch die Sprecherunabhängigkeit kann man erreichen, wenn man Clustering-Methoden anwendet [RA79]. Das Verfahren kann in Echtzeitanwendungen implementiert werden [Pla05]. Was die Erkennungsrate betrifft, man kann bei einer Einzelworterkennung hohe Erkennungsraten erreichen. So konnte L. Rabiner bei 37 Wörtern und trainierten Sprechern eine Erkennungsrate von 100% und 96,5% bei 54 Wörtern und unabhängigen klassifizierten Sprechern erreichen [RL81][LR83].

2.2 Hidden Markov Modelle (HMM)

Das Problem der Variabilität der Sprache lässt sich durch eine statistische Beschreibung der Variabilität in Form von stochastischen Modellen lösen. In der Lernphase wird für jede Klasse ein Modell λ_V für die Erzeugung der Musterverläufe dieser Klasse aufgebaut. In der Testphase wird eine Klassifikation durchgeführt (Abbildung 2-3). Die Wahrscheinlichkeit, dass dieses Muster vom Modell (Erzeugungswahrscheinlichkeit) erzeugt werden kann, wird berechnet. Diese Berechnung wird für jede Klasse durchgeführt. Die Klasse mit der höchsten Wahrscheinlichkeit gilt als erkannt (Maximum-Likelihood-Prinzip).

Für jedes zu erkennendes Wort wird ein „left-to-right-model“ λ_V erstellt, in dem jedes Phonem[2] einem Zustand entspricht. Dieses Modell erlaubt keine Übergänge zu Vorgängerzuständen und modelliert damit den Zeitaspekt. Abbildung 2-4 zeigt ein Beispiel für ein Modell mit 5 Zuständen und ein Modellierungsbeispiel für das

[2] kleinste unterscheidende Einheiten einer Sprache. Deutsch hat etwa 35 Phoneme.

Wort „haben“. Da das aktuelle Durchlaufen der Zustände von außen unsichtbar („hidden“) ist, wird dieses Modell als Hidden-Markov-Modell (HMM) bezeichnet.

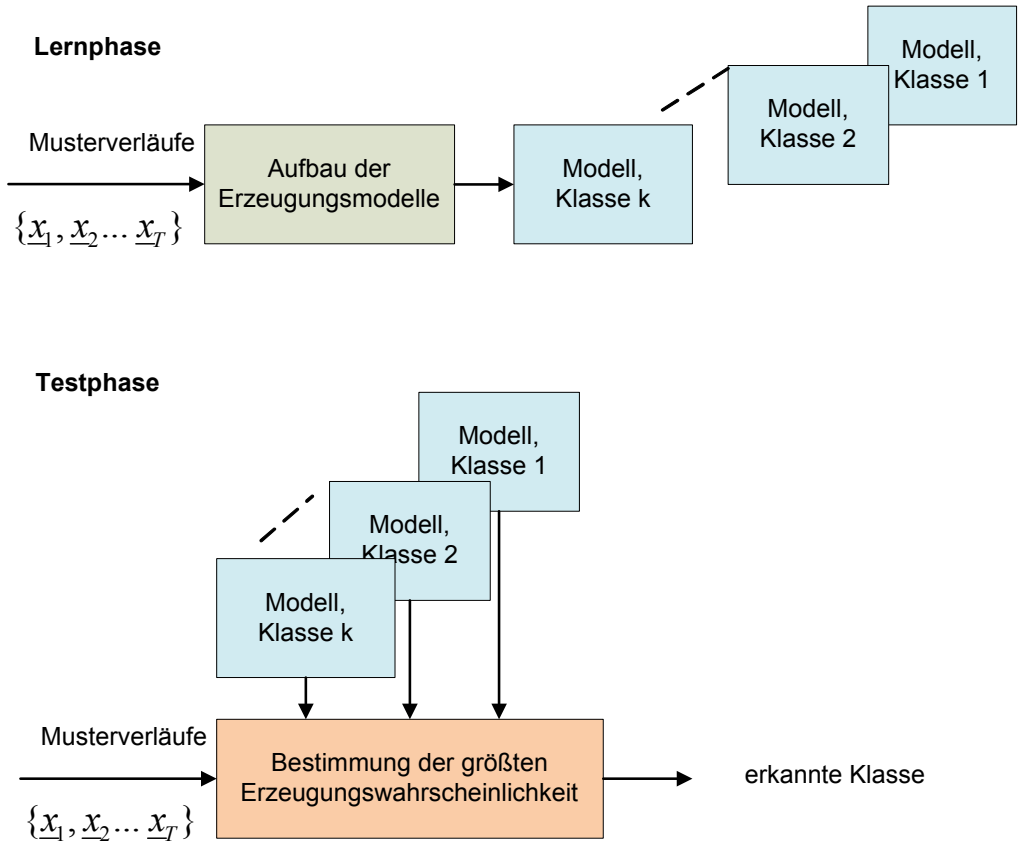


Abbildung 2-3 HMM-Erzeugungsmodelle in der Lern- und Testphase

Im Modell steht p(x|i) für die Erzeugungswahrscheinlichkeit von x im Zustand i und a_{ij} für die Übergangswahrscheinlichkeit vom Zustand i zum Zustand j.

Ein HMM ist vollständig bestimmt durch folgende Modelparameter:

- Die Anfangswahrscheinlichkeit π.
- Die Übergangsmatrix A (a_{11}, a_{12}…).
- Die Erzeugungswahrscheinlichkeiten p(x|i) in den Zuständen S.

Die Modellparameter müssen in einer Trainingsphase bestimmt werden. Dazu wird der Baum-Welch-Algorithmus eingesetzt.

Bei der Erkennung tritt das Problem des Rechenaufwands auf. Die Anzahl der möglichen Kombinationen von Zuständen (Reihenfolge der Phoneme) ist äußerst hoch. Hier wird der Viterbi-Algorithmus eingesetzt. Dieser Algorithmus verwendet das Prinzip der dynamischen Programmierung. Es wird jeweils der maximale Wahrscheinlichkeitswert weiterberechnet. In der Praxis erlangte der Viterbi-Algorithmus sehr große Bedeutung. Die Algorithmen werden in [Rab89] beschrieben.

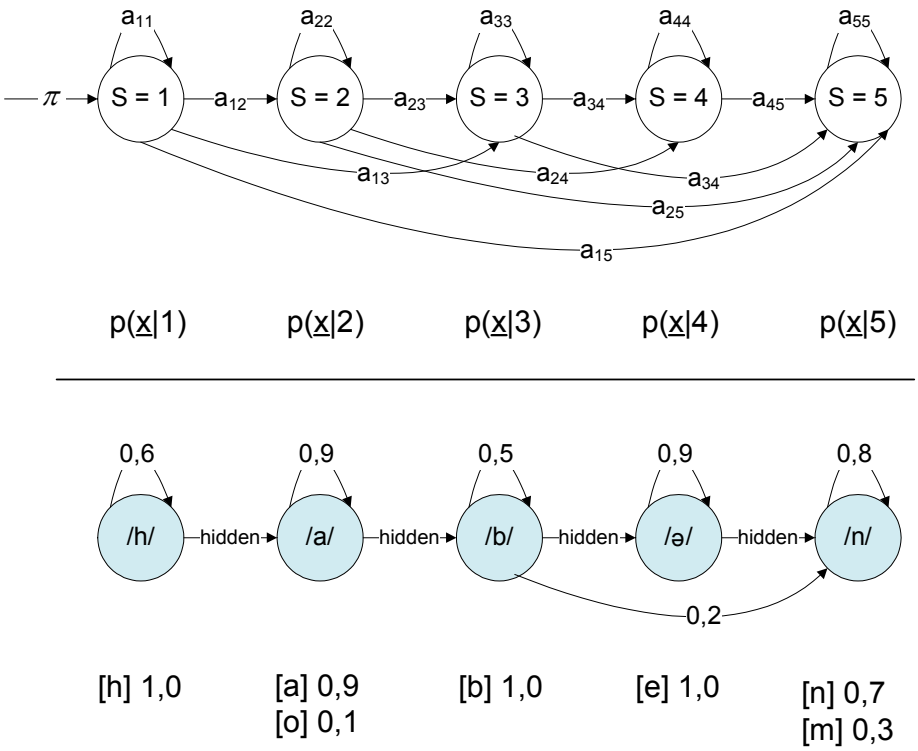


Abbildung 2-4 allgemeines HMM und ein Beispielmodell für das Wort „haben“

Auch das HMM-Verfahren zur automatischen Spracherkennung lässt sich so modifizieren, dass Wortketten und fließende Sprache verarbeitet werden können Dazu werden weitere Verarbeitungsschritte, wie das Herausfiltern von ungültigen Wörtern gemäß einem Lexikon aller gültigen Wörter, durchgeführt. Auch Syntaxfehler und falsche Zusammensetzungen werden mit einer Syntaktik-Semantik-Analyse herausgefiltert [RJ04]. Berücksichtigt man bei der Trainingsphase alle möglichen vorkommenden Phoneme in einem Wort, so kann man eine Sprecherunabhängigkeit erreichen. In der Regel sind 500 Trainings

erforderlich bei einem Telefonkanal und einem niedrigen Rauschpegel. L. Rabiner erreichte bei 11 Wörtern (0…9 und O) eine Erkennungsrate von 98% [RJ04].

Das nächste Kapitel geht auf die Entwicklung des Systems zur automatischen Spracherkennung ein.

3 Entwicklung des Systems zur automatischen Sprach-erkennung

Bei der Entwicklung des Systems zur automatischen Spracherkennung müssen zuerst die Anforderungen analysiert werden. Erst dann kann man die Systemarchitektur definieren. Zu den analysierten Anforderungen kann man ein Verfahren entwickeln oder auswählen.

3.1 Anforderung an das System

In Kapitel 1.3 wurde das Ziel dieser Arbeit vorgestellt. Aus diesem Ziel kann man folgende Anforderungen an das System erkennen:

Das System soll einen Roboter steuern. Was die Natürlichkeit der Sprache (Kommandos, Wortketten oder kontinuierliche Sprache) betrifft, werden in der Regel Kommandos als Spracheingabe bei Systemen zur Steuerung von Maschinen verwendet (z.B. Fernbedienung zur Steuerung von Maschinen) [Sic83].

Das System soll auf dem digitalen Signalprozessor TI D.MODULE.C6713 in Echtzeit laufen. Das auszuwählende Verfahren soll diese Anforderung erfüllen. Die Verarbeitungsgeschwindigkeit soll daher genügend klein bleiben.

Ferner soll selbstverständlich eine möglichst hohe Erkennungsrate erreicht werden.

3.2 Auswahl eines Verfahrens

Das auszuwählende Verfahren soll die oben genannten Anforderungen erfüllen. Es soll gleichzeitig auf dem DSP implementierbar und im Rahmen dieser Arbeit realisierbar sein. In Kapitel 2 wurden zwei Verfahren zur automatischen Spracherkennung kurz vorgestellt, nämlich die nichtlineare zeitliche Verzerrung (DTW) und die Hidden-Markov-Modelle (HMM).

Da man mit Kommandos und einen kleinen Wortschatz einen Roboter steuern kann, genügt die Einzelworterkennung der Anforderung der Steuerung.

Hinsichtlich der Erkennungsrate bei der Einzelworterkennung weist das DTW-Verfahren eine leicht bessere Erkennungsrate als das HMM-Verfahren.

Das HMM-Verfahren ist vorteilhafter hinsichtlich der Sprecherunabhängigkeit. Allerdings ist ein aufwendiger Lernprozess damit verbunden. Was den Speicherbedarf betrifft, ist das HMM-Verfahren nur bei großem Wortschatz besser. HMM-Systeme sind daher oft in Desktopanwendungen zu finden (z.B. Nuance). Einige stehen auch als Open-Source für Desktopanwendungen zur Verfügung (z.B. HTK von Universität Cambridge).

Der Realisierungs- und Implementierungsaufwand ist beim DTW-Verfahren kleiner. Dieses Verfahren wird in mobilen Geräten eingesetzt, wo Ressourcen und Rechenkapazitäten klein sind.

Das Merkmalextrahieren in der Vorverarbeitungsstufe ist das gleiche für beide Verfahren. Der größte Teil der Arbeit wird somit realisiert, denn die Algorithmen der Vorverarbeitung haben einen größeren Aufwand als die Algorithmen der Erkennung.

Aus diesen Gründen wird das DTW-Verfahren ausgewählt und wird zusammen mit der Vorverarbeitung in den nächsten Kapiteln erklärt.

Eigenschaften	HMM	DTW
Echtzeitfähig	✓	✓
Hohe Erkennungsrate		✓
Kleine Ressourcen und Rechenkapazität		✓
Sprecherunabhängigkeit	✓	

Tabelle 3-1 HMM vs. DTW für Einzelworterkennung und kleinen Wortschatz

3.3 Beschreibung der Systemarchitektur

Da alle Systeme zur automatischen Spracherkennung eine Lernphase (Training) durchgehen müssen, kann man das System auf höchster Ebene in zwei Phasen teilen:

- Die Lern- oder Trainingsphase. In dieser Phase wird das System trainiert. Die daraus gewonnenen Informationen werden gespeichert. Die Lernphase wird für jeden Sprecher durchgeführt werden.
- Die Test- oder Erkennungsphase. In dieser Phase wird die Sprache erkannt und ein entsprechendes Signal ausgegeben. Dabei werden die aus der Trainingsphase gewonnenen Informationen verwendet.

Abbildung 3-1 zeigt das System zur automatischen Spracherkennung auf höchster Ebene.

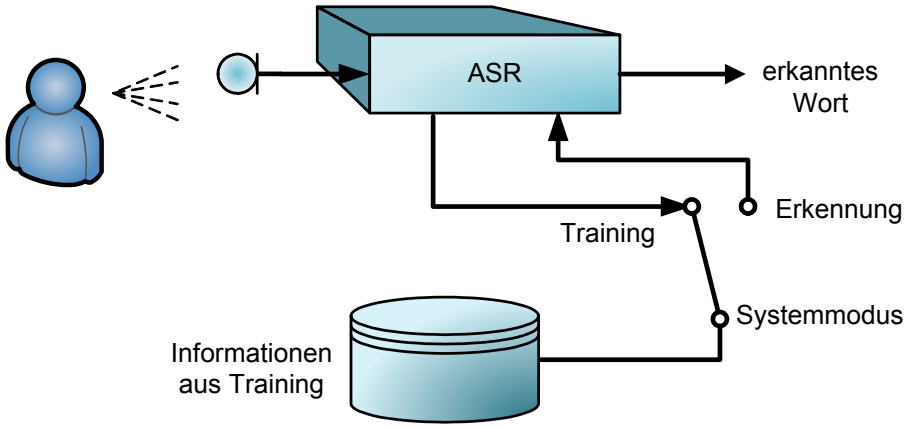


Abbildung 3-1 ASR-System auf der höchsten Ebene

3.4 Beschreibung der Systemkomponenten

Das ASR-System besteht im Prinzip aus einer Vorverarbeitungsstufe und einer Erkennungsstufe. Die Vorverarbeitungsstufe besteht aus einer Sprachverarbeitungsstufe, einer Stufe zur Isolierung der Kommandos und einer Stufe zur Merkmalextraktion (Abbildung 3-2).

In der Sprachverarbeitungsstufe wird die Sprache mit einem Analog-Digital-Umsetzer digitalisiert und dann einer Filterung unterzogen.

In einem weiteren Teil der Vorverarbeitungsstufe werden die gesprochenen Kommandos isoliert, um bei der späteren Analyse Rechenzeit und Speicherbedarf zu reduzieren. Der Anfangs- und Endpunkt jedes Kommandos werden detektiert. Dieses Verfahren ist sehr bedeutend. Fehler in der Isolierung der Kommandos führen zum Informationsverlust, was die Erkennung schwerwiegend beeinträchtigen kann.

Aus dem isolierten Kommando werden Merkmale extrahiert. Diese werden in der Trainingsphase als Referenz gespeichert. In der Erkennungsphase werden die extrahierten Merkmale an die Erkennungsstufe weitergegeben.

Zuletzt kommt eine Erkennungsstufe, die die aktuellen Kommandomerkmale mit allen Referenzmerkmalen vergleicht und daraus ermittelt, welches Kommando gesprochen wurde.

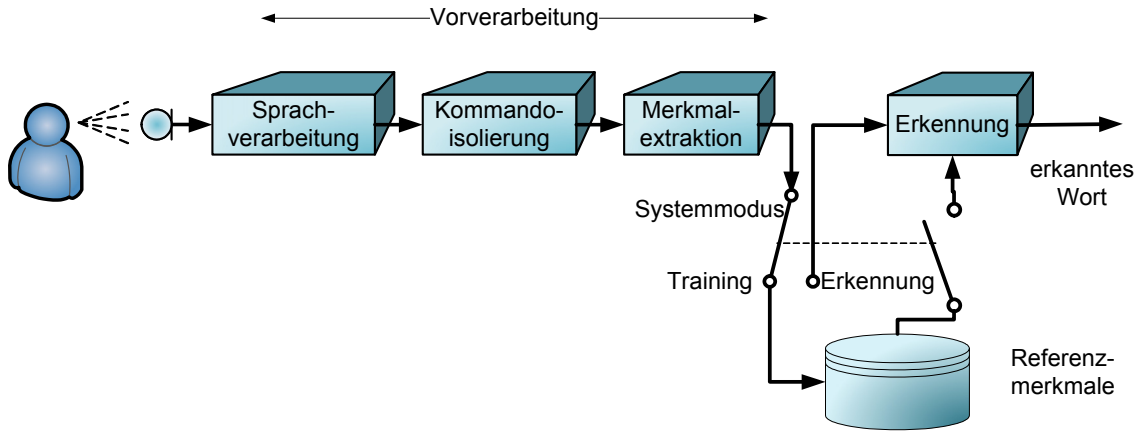


Abbildung 3-2 ASR-Systemkomponenten

3.4.1 Verarbeitung der aufgenommenen Sprache

Die gesprochenen Kommandos werden zuerst digitalisiert. Eine Vorfilterung muss durchgeführt werden, um eine einwandfreie Kommandoisolierung zu gewährleisten. Abbildung 3-3 zeigt die beiden Stufen der Sprachverarbeitung.

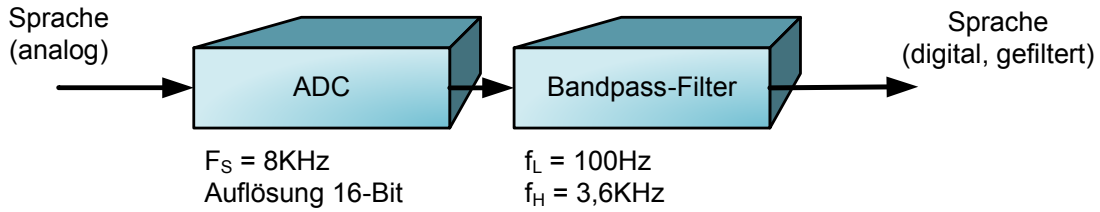


Abbildung 3-3 Verarbeitung der aufgenommenen Sprache

3.4.1.1 Digitalisierung

Die Digitalisierung wird durch D.MODULE.PCM3003, den Analog-Digital-Umsetzer der Firma Texas Instruments, realisiert. Der PCM3003 hat folgende technische Daten:

- Auflösung 16 Bit.
- Abtastfrequenz 8 bis 48 kHz.
- 4 oder 8 Audiokanäle.
- Single-ended, 1,8 Vpp.
- Delta-Sigma-Modulation.

Bei der Artikulation von Frikativen[3] werden Frequenzanteile im Ultraschallbereich erzeugt. Allerdings sind keine nötigen Informationen für die Spracherkennung darin enthalten. Die meisten nötigen Informationen, um eine menschliche Stimme zu erkennen, liegen im Spektrum unter 4 kHz. Das ist aus dem Telefon-Kanal bekannt. Die ausgewählte Abtastfrequenz beträgt dabei gemäß Nyquist-Theorem mindestens 8 kHz.

[3] Ein Frikativ ist ein Konsonant, bei dessen Bildung ein Reibelaut erzeugt wird. Frikative können stimmhaft oder stimmlos sein.

3.4.1.2 Vorfilterung

Die digitalisierte Sprache wird einer Bandpass-Filterung mit folgenden Grenzfrequenzen unterzogen: f_L = 100 Hz und f_H = 3,6 kHz. Nach der Filterung ist die Qualität der Sprache vergleichbar mit der Qualität eines Telefon-Kanals (3,4 kHz). Die Filterung ist sehr wichtig für die nächste Stufe, die Kommandoisolierung, weil dadurch unnötige niedrige Frequenzanteile (z.B. 60 Hz-„hm“) herausgefiltert werden.

3.4.2 Isolierung der Sprachkommandos

Eine weitere Stufe der Vorverarbeitung ist die Kommandoisolierung. Bei der Einzelworterkennung geht man davon aus, dass in einem Aufnahmeintervall ein Sprachkommando existiert. Im Laufe der Aufnahme können Umgebungsrauschen oder einfach eine Stille vor und nach dem Kommando vorkommen.

Eine der Aufgaben der Isolierung von Sprachkommandos besteht darin, der Anfangs- und Endpunkt eines Kommandos zu detektieren. Die Rechenzeit wird dadurch minimiert. Falls kein Kommando im Aufnahmeintervall existiert, kann ein entsprechender Fehler ausgegeben werden.

Das Hauptproblem bei der Kommandoisolierung bilden das Umgebungsrauschen und Geräusche des Sprechers (Atmen, Mundgeräusche, etc.). Die Extraktion eines Kommandos in einer Umgebung mit niedrigem Signal-Rausch-Abstand ist keine triviale Aufgabe. Diese Arbeit wird auf das Problem des Umgebungsrauschens allerdings nicht weiter eingehen. Mit der Verwendung eines Rauschen unterdrückenden „close-talking“-Mikrofons kann man die Einflüsse des Umgebungsrauschens minimieren [RS75]. Durch den Einsatz des Bandpass-Filters in der vorigen Stufe wird das störende 60 Hz-„hm“ eliminiert. Diese beiden genannten Maßnahmen sind sehr bedeutend für die einwandfreie Funktion der Kommandoisolierung.

Das in dieser Arbeit eingesetzte Verfahren zur Kommandoisolierung basiert auf dem von Rabiner und Sambur vorgestellten Verfahren [RS75]. Es handelt sich um ein Energie-basiertes Verfahren. Abbildung 3-4 stellt das Verfahren dar.

Zuerst werden neben dem Energieverlauf bestimmte Schwellenwerte berechnet, dann werden Kriterien bestimmt, um den Anfangs- und Endpunkt des Kommandos zu ermitteln. Diese Schritte werden in den nächsten Unterkapiteln erklärt.

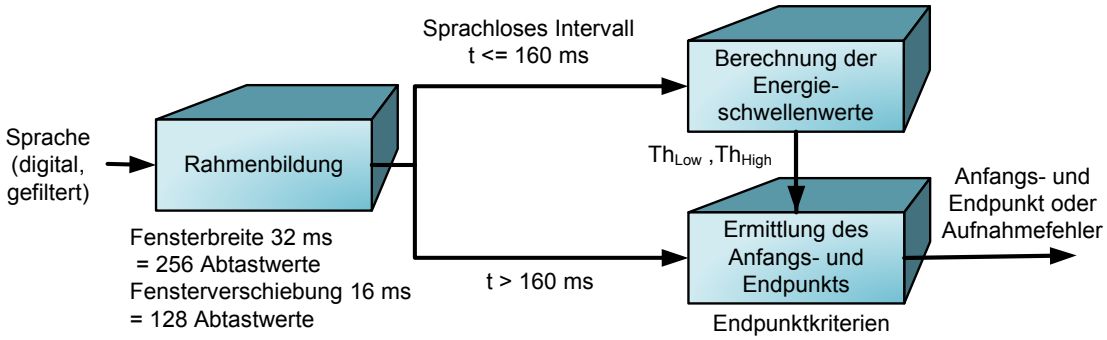


Abbildung 3-4 Isolierung eines Kommandos

3.4.2.1 Rahmenbildung zur Berechnung der Energie

Um den Anfangs- und Endpunkt zu detektieren, wird ein über die Abtastwerte gleitender Rahmen eingesetzt. Die Energie des Rahmens wird berechnet, dann gleitet der Rahmen weiter, d.h. er wird um eine bestimmte Anzahl von Abtastwerten verschoben.

Verschiedene Werte für die Rahmenbreite und -verschiebung können eingesetzt werden. Die Rahmenbreite muss genug klein sein, um abrupte Änderung im Verlauf zu detektieren. Anderseits soll die Rahmenbreite möglichst groß gewählt werden, damit Rechenzeit gespart werden kann. Ein guter Kompromiss für die Rahmenbreite ist 32 ms. Das entspricht 256 Abtastwerten bei der 8kHz-Abtastfrequenz. Die Rahmenverschiebung wird so gewählt, dass sich zwei aufeinander folgende Rahmen überlappen. Dies bewirkt einen glättenden Effekt auf den Energieverlauf und eignet sich besonders gut für die Ermittlung der Endpunkte. Eine Rahmenverschiebung von 50% wird ausgewählt (Abbildung 3-5). Das bedeutet auch eine Überlappung von 50% und entspricht 16 ms oder 128 Abtastwerten. Dies hat den Vorteil, dass die Rahmenbreite ein Vielfaches der Rahmenverschiebung ist.

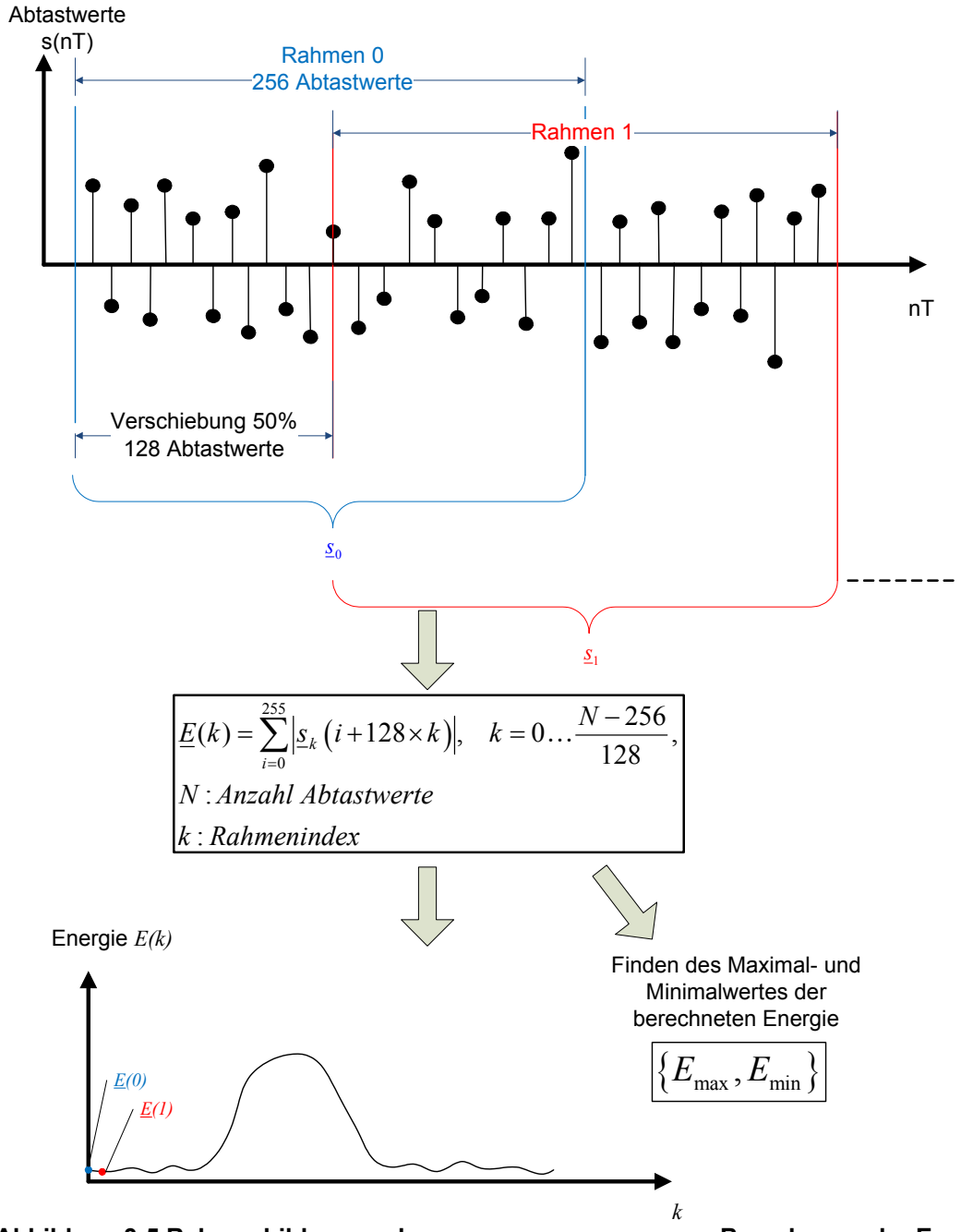


Abbildung 3-5 Rahmenbildung und Berechnung der Energie

3.4.2.2 Berechnung der Sprachenergie und der Energieschwellenwerte

Für eine Zahlenfolge ist die Energie als Summe der einzelnen Quadratwerte definiert. Stattdessen wird hier eine quasi Energie berechnet, die aus der Summe der Beträge berechnet wird.

$$\underline{E}(k) = \sum_{i=0}^{WinWidth} \left| \underline{s}_k (i + WinShift \times k) \right|, \quad k = 0 \ldots \frac{N - WinWidth}{WinShift} \tag{3.1}$$

Dabei sind $\underline{s}_k$ die Abtastwerte des k-ten Rahmens, N die gesamte Anzahl der Abtastwerte, $WinShift$ die Rahmenverschiebung (128 Abtastwerte) und $WinWidth$ (256 Abtastwerte) die Rahmenbreite.

Diese Form hat den Vorteil, dass die Änderungen der Signalamplitude besser als mit der Quadratform reflektiert werden. Ferner kann die Berechnung der Beträge mit weniger Aufwand durchgeführt werden.

Der berechnete Energieverlauf hat die Form eines Impulses. Je nach dem, was für ein Kommando gesprochen wird, können ein oder mehrere Impulse vorkommen. Während der Stille weist der Verlauf schwache Energiewerte auf. Dahingegen treten große Energiewerte während der Sprache auf. Somit können Energieschwellenwerte definiert werden, um die Endpunkte der Sprache zu detektieren.

Zur Berechnung der Energieschwellenwerte Th_{Low} und Th_{High} wird vorausgesetzt, dass zunächst ein sprachloses Intervall im zeitlichen Ablauf des aufgenommenen Gesamtintervalls als erstes vorkommt. Die Breite des Intervalls soll mindestens 100ms sein. Sie wurde auf 160ms festgelegt und ist somit ein Vielfaches von der Rahmenbreite. In diesem Intervall werden der Maximal- und Minimalwert der Energie (E_{Max}, E_{Min}) ermittelt, so dass am Ende des Intervalls die Energieschwellenwerte folgendermaßen berechnet werden:

$$I_1 = 0,05 \times \left(E_{Max} - E_{Min} \right) + E_{Min} \tag{3.2}$$

$$I_2 = 4 \times E_{Min} \tag{3.3}$$

$$Th_{Low} = Min\left(I_1, I_2 \right) \tag{3.4}$$

$$Th_{High} = 4 \times Th_{Low} \tag{3.5}$$

Mit den Schwellenwerten können Kriterien definiert werden, um den Anfangs- und Endpunkt des Kommandos zu detektieren.

3.4.2.3 Kriterien zur Ermittlung des Anfangs- und Endpunktes eines Kommandos

Abbildung 3-6 zeigt den normalen Energieverlauf eines Sprachkommandos. Ein Anfangspunkt wird erkannt, wenn die Energie zum ersten Mal die beiden Schwellenwerte überschreitet. Ein Endpunkt wird erkannt, wenn die beiden Schwellenwerte zum letzten Mal unterschritten werden.

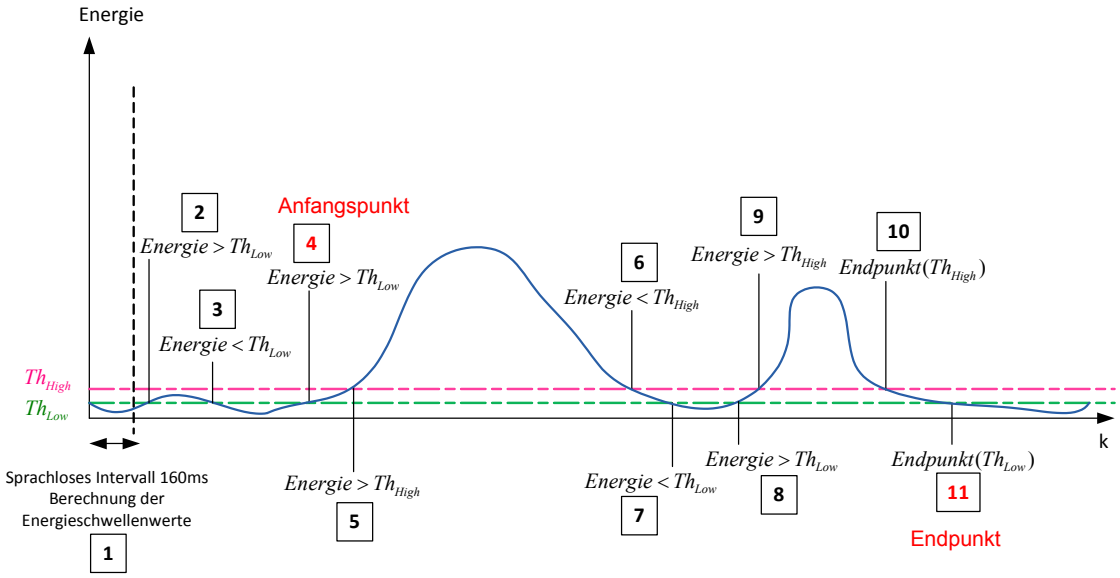


Abbildung 3-6 Detektierung Der Endpunkte eines Kommandos

Im Folgenden wird ein Beispielszenario zur Detektierung der Endpunkte vorgestellt, das alle Kriterien enthält. Die folgende Nummerierung ([1]…[11]) entspricht den markierten Stellen in der Abbildung.

[1] Sprachloses Intervall (160 ms). In diesem Intervall werden die Schwellenwerte Th_{Low} und Th_{High} ermittelt.

[2] Die Energie überschreitet den untere Schwellenwert Th_{Low}. Diese Stelle wird gespeichert. Wenn die Energie später Th_{High} überschreitet, dann wird diese Stelle als Anfangspunkt genommen.

[3] Th_{Low} wird unterschritten, bevor Th_{High} überschritten wird. Daher wird die gespeicherte Stelle aus [2] verworfen.

[4] Th_{Low} wird wieder überschritten. Diese Stelle wird wieder gespeichert.

[5] Die Energie überschreitet beide Schwellenwerte. Jetzt wird die Stelle aus [4] als tatsächlicher Anfangspunkt übernommen. Der zeitliche Abschnitt zwischen den Stellen [4] und [5] könnte größer als nötig sein. Um Rechenzeit zu sparen, wird dieser Abschnitt oder Distanz mit einem Maximalwert $MaxThDist$ verglichen und ggf. auf $MaxThDist$ reduziert werden. $MaxThDist$ wurde auf 150ms festgelegt. Dieser Wert garantiert, dass eventuelle Frikative am Anfang des Kommandos übernommen werden, denn ein Frikativ nimmt 30ms bis 50ms in Anspruch [RS75]. Dieser Sonderfall ([a]) ist in Abbildung 3-7 dargestellt.

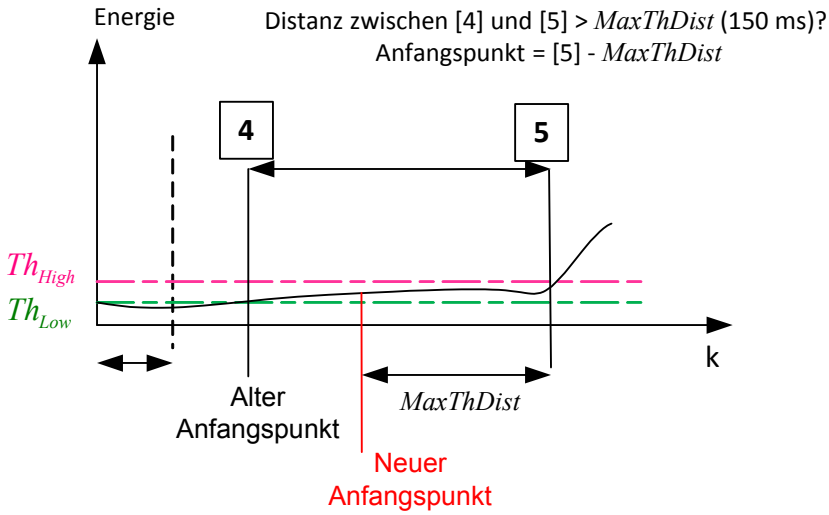


Abbildung 3-7 Sonderfall [a] bei der Detektierung der Endpunkte

[6] Th_{High} wird unterschritten. Diese Stelle wird temporär als Endpunkt übernommen, denn Th_{Low} soll unterschritten werden, damit ein Endpunkt erkennt wird. Ähnlich wie im Sonderfall [a] wird im Sonderfall [b] die Distanz zwischen den Stellen überprüft, an denen die beiden Schwellenwerte unterschritten wurden. Das gilt auch für den Fall, dass das Kommando endet, bevor Th_{Low} unterschritten wird. Abbildung 3-8 zeigt den Sonderfall [b].

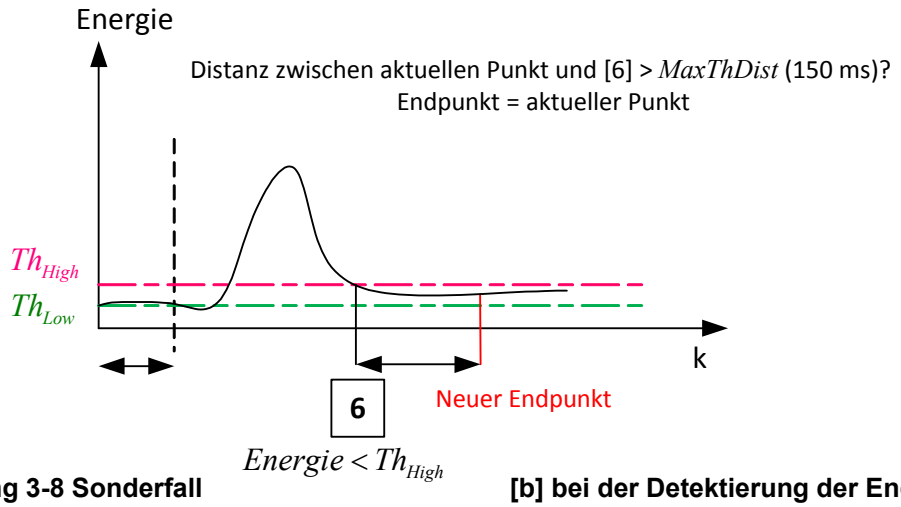


Abbildung 3-8 Sonderfall [b] bei der Detektierung der Endpunkte

[7] Th_{Low} wird unterschritten. Diese Stelle wird als Endpunkt erkannt, wenn die Bedingung vom Sonderfall [b] nicht erfüllt ist. An dieser Stelle wurden sowohl der Anfangs- als auch der Endpunkt erkannt und somit ist ein Energieimpuls erkannt. Um zu garantieren, dass der erkannte Impuls gültig ist, muss die Impulsbreite nach zwei Sonderfällen überprüft werden:

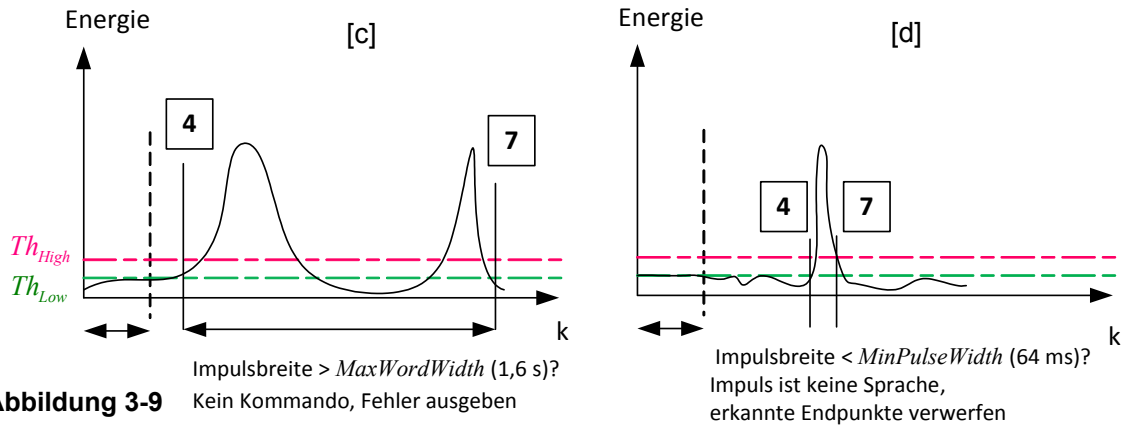


Abbildung 3-9 Sonderfälle [c] und [d] bei der Detektierung der Endpunkte

- Wenn die Impulsbreite den Minimalwert $MinPulseWidth$ (64 ms) unterschreitet, wird der Impuls als ein Geräusch erkannt und somit verworfen (Sonderfall [c]).

- Überschreitet die Impulsbreite den Maximalwert $MaxWordWidth$ (1,6 s), dann handelt sich dabei nicht um ein Kommando. In diesem Fall wird ein Fehler ausgegeben (Sonderfall [d]).

[8] Th_{Low} wird wieder überschritten. Das deutet auf einen zweiten Impuls ein. Wie bei [4] wird diese Stelle gespeichert.

[9] Th_{High} wird überschritten. Da ein Anfangspunkt schon existiert, wird hier der Sonderfall [e] geprüft. Dabei wird die Distanz zwischen den Impulsen nach Gültigkeit überprüft. Überschreitet die Distanz den Maximalwert $MaxPulseDist$, bedeutet dies, dass die beiden Impulse nicht zu einem Kommando gehören. In diesem Fall wird ein Fehler ausgegeben. Wenn aber die Impulse zu einem Kommando gehören, d.h. wenn die Bedingung der Sonderfall [e] nicht erfüllt ist, wird der letzte Endpunkt verworfen, damit ein neuer Endpunkt gesucht werden kann. Die Stelle des letzten Endpunktes wird aber weiterhin gemerkt, für den Fall, dass der neue Impuls ungültig ist.

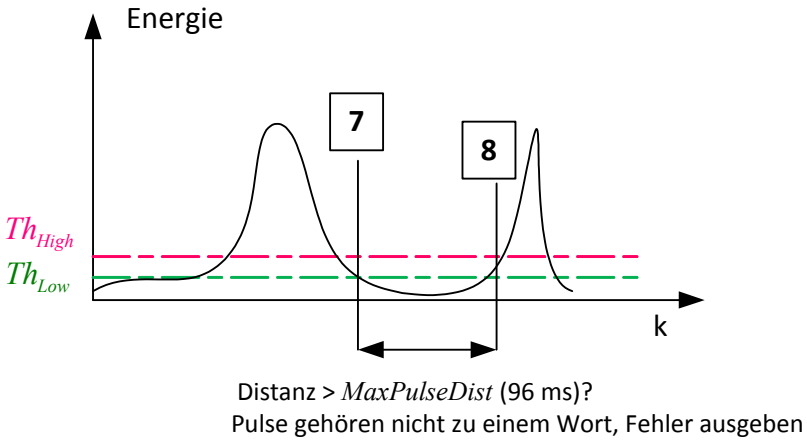


Abbildung 3-10 Sonderfall [e] bei der Detektierung der Endpunkte

[10] Th_{High} wird unterschritten. Diese Stelle wird temporär als Endpunkt übernommen. Wie bei [6] wird hier der Sonderfall [b] geprüft.

[11] Th_{Low} wird unterschritten. Diese Stelle wird als Endpunkt genommen, wenn die Bedingung der Sonderfälle [b], [c] und [d] nicht erfüllt sind.

Wenn ein Kommando nicht zu Ende gesprochen wird, bevor das Aufnahmeintervall endet, kann es zu keiner Unterschreitung der Schwellenwerte kommen, so wie der Sonderfall [f] zeigt. In diesem Fall wird kein Endpunkt erkannt und ein Fehler ausgegeben.

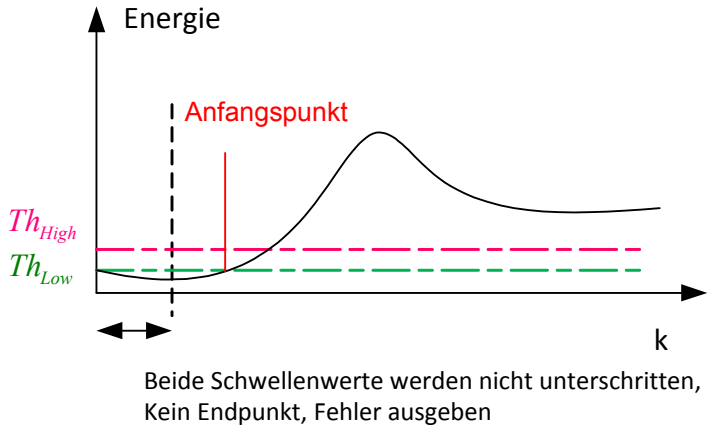


Abbildung 3-11 Sonderfall [f] bei der Detektierung der Endpunkte

3.4.3 Extraktion der Kommandomerkmale

Die Merkmalextraktion aus einem Kommando ist eine wichtige Stufe in der automatischen Spracherkennung. Die Merkmale sollen geringe Große haben und charakteristische Eigenschaften der Kommandos enthalten. Verschiedene Verfahren werden eingesetzt, um Kommandomerkmale zu extrahieren. Bei allen Verfahren wird eine Fensterung durchgeführt. Nach jeder Fensterverschiebung wird mit dem entsprechenden Verfahren einen Satz von Koeffizienten berechnet. Die gesamte Menge an Koeffizienten bildet die Kommandomerkmale.

Eine Möglichkeit besteht z.B. darin, mit 5 bis 32 Bandpass-Filtern, die eine Bandbreite von 100 Hz bis 3000 Hz abdecken, eine Filterbankanalyse durchzuführen. Aus den Parallelausgängen der Filterbank bekommt man die Koeffizienten, die als Kommandomerkmale verwendet werden.

Ein anderes Verfahren verwendet als Eingang die Filterbankkoeffizienten, und weist bessere Erkennungsraten auf. Es handelt sich dabei um die Cepstraltransformation. Dabei werden die sogenannten Cepstrumkoeffizienten (Mel Frequency

Cepstrum Coefficients, MFCC) berechnet, die als Kommandomerkmale verwendet werden [Pla05][ST95].

Ein weiteres Verfahren ist die Berechnung von Korrelationskoeffizienten mit der Autokorrelationsfunktion AKF. Die Korrelationskoeffizienten werden dann als Kommandomerkmale verwendet. Das Verfahren ist einfach zu implementieren und weist gute Erkennungsraten auf. Die Methode der linearen Vorhersage (Linear Prediction Coding, LPC [RL81]) verwendet als Eingang die Korrelationskoeffizienten. Daraus werden die LPC-Koeffizienten (LPCC) berechnet. Diese Methode weist eine bessere Erkennungsrate gegenüber der AKF-Methode auf [Rus94][ST95].

Mit der MFCC- und LPC-Methode konnten die höchsten Erkennungsraten erreicht werden. Allerdings ist der Aufwand dabei höher als bei der AKF-Methode und der Filterbankanalyse. Mit der AKF-Methode konnten bessere Erkennungsraten als mit der Filterbankanalyse erreicht werden [Sic83]. In dieser Arbeit wird die AKF-Methode eingesetzt. Es wird empfohlen in weiteren Entwicklungen die Merkmalextraktion auf die LPC-Methode zu erweitern. Abbildung 3-12 zeigt die Stufen der Merkmalextraktion mit der AKF-Methode.

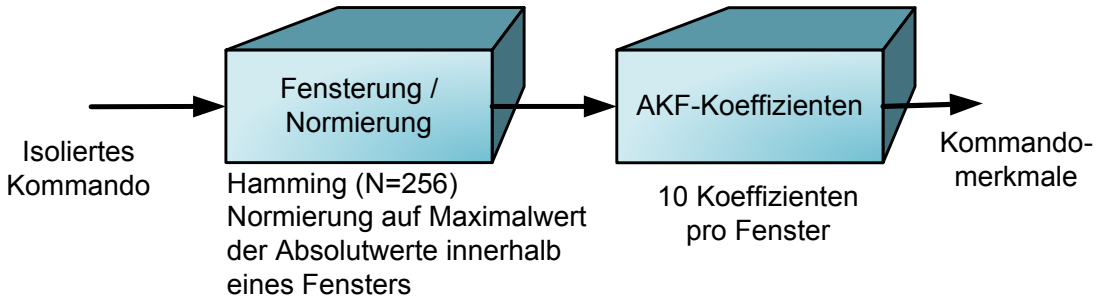


Abbildung 3-12 Merkmalextraktion eines Kommandos

3.4.3.1 Fensterung und Normierung der Abtastwerte

Um genügende Merkmale aus dem Kommando zu gewinnen, werden die gefilterten Abtastwerte, die zwischen den detektierten Endpunkten liegen, in Rahmen geteilt. Aus jedem Rahmen wird eine bestimmte Anzahl an Merkmalen

gewonnen. Um die Effekte der Rahmenbildung zu minimieren, wird ein Glättungsfenster eingesetzt. Das typischerweise eingesetzte Fenster ist das Hamming-Fenster mit folgender Gleichung:

$$\underline{w}(i) = 0,54 - 0,46 \times \cos\left(\frac{2\pi i}{WinWidth - 1}\right), \quad i = 0 \ldots WinWidth - 1 \tag{3.6}$$

Mit $WinWidth$ (256 Abtastwerte) die Rahmenbreite.

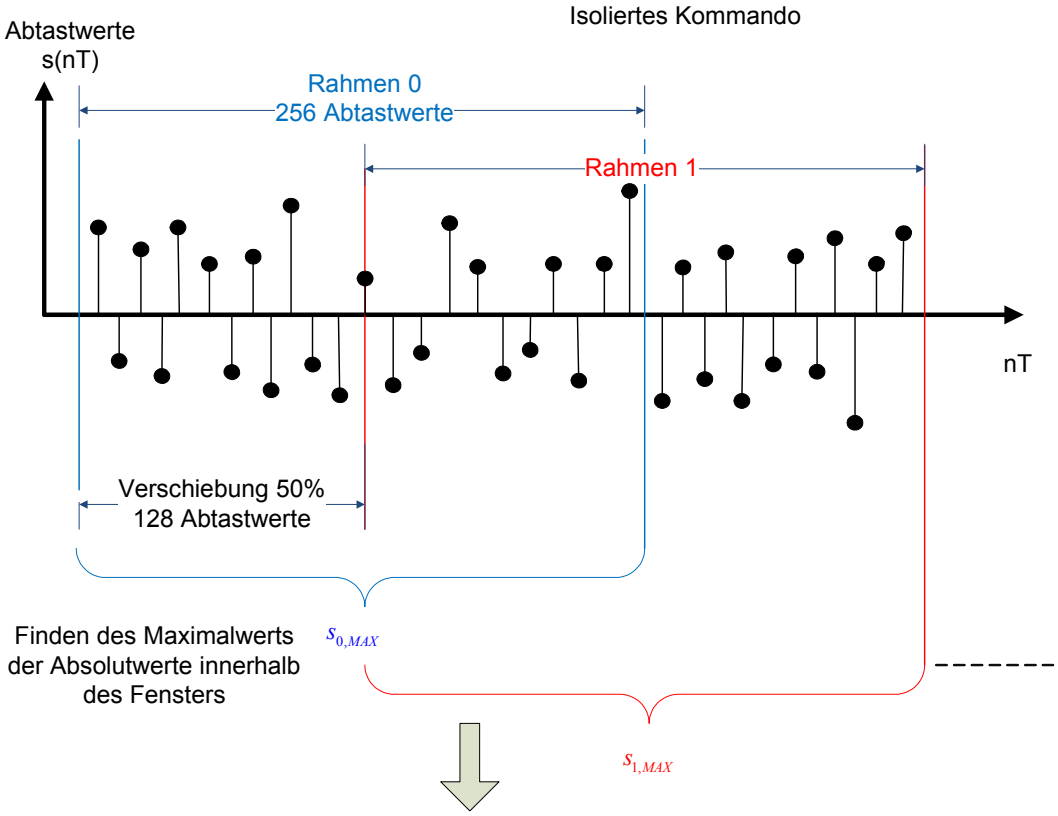
Die Abtastwerte innerhalb eines Rahmens werden mit den Koeffizienten $\underline{w}(i)$ des Fensters multipliziert. Nach der Fensterung werden die Abtastwerte innerhalb eines Rahmens auf den maximalen Betragswert innerhalb des Rahmens normiert. Das führt zu Minimierung des Einflusses von Amplitudenunterschieden bei unterschiedlichen Kommandos (Abbildung 3-13). Die Fensterung und Normierung kann man mit folgender Substitution beschreiben:

$$\underline{s}_k(i) = \frac{\underline{s}_k(i) \times \underline{w}(i)}{s_{k,MAX}}, \tag{3.7}$$

$$i = (WinShift \times k) \ldots (WinShift \times k) + WinShift,$$

$$k = 0 \ldots \frac{(Endpunkt - Anfangspunkt) - WinWidth}{WinShift}$$

Dabei sind $\underline{s}_k$ die Abtastwerte des k-ten Rahmens, $WinShift$ die Rahmenverschiebung (128 Abtastwerte) und $WinWidth$ (256 Abtastwerte) die Rahmenbreite.



$$\underline{s}_k(i) = \frac{\underline{s}_k(i) \times \underline{w}(i)}{s_{k,MAX}}, \quad i = (128 \times k) \ldots (128 \times k) + 128, \quad k = 0 \ldots \frac{(Endpunkt - Anfangspunkt) - 256}{128}$$

Abbildung 3-13 Fensterung und Normierung des isolierten Kommandos

3.4.3.2 Berechnung der Merkmale eines Kommandos

Nach der Rahmenbildung, Fensterung und Normierung können die Autokorrelationskoeffizienten für jeden Rahmen berechnet werden. Die Anzahl an Autokorrelationskoeffizienten pro Rahmen wird mit *CoeffOrder* bezeichnet. Typische Werte dafür liegen zwischen 8 und 12 [RL81]. In dieser Arbeit wurde *CoeffOrder* auf 10 festgelegt.

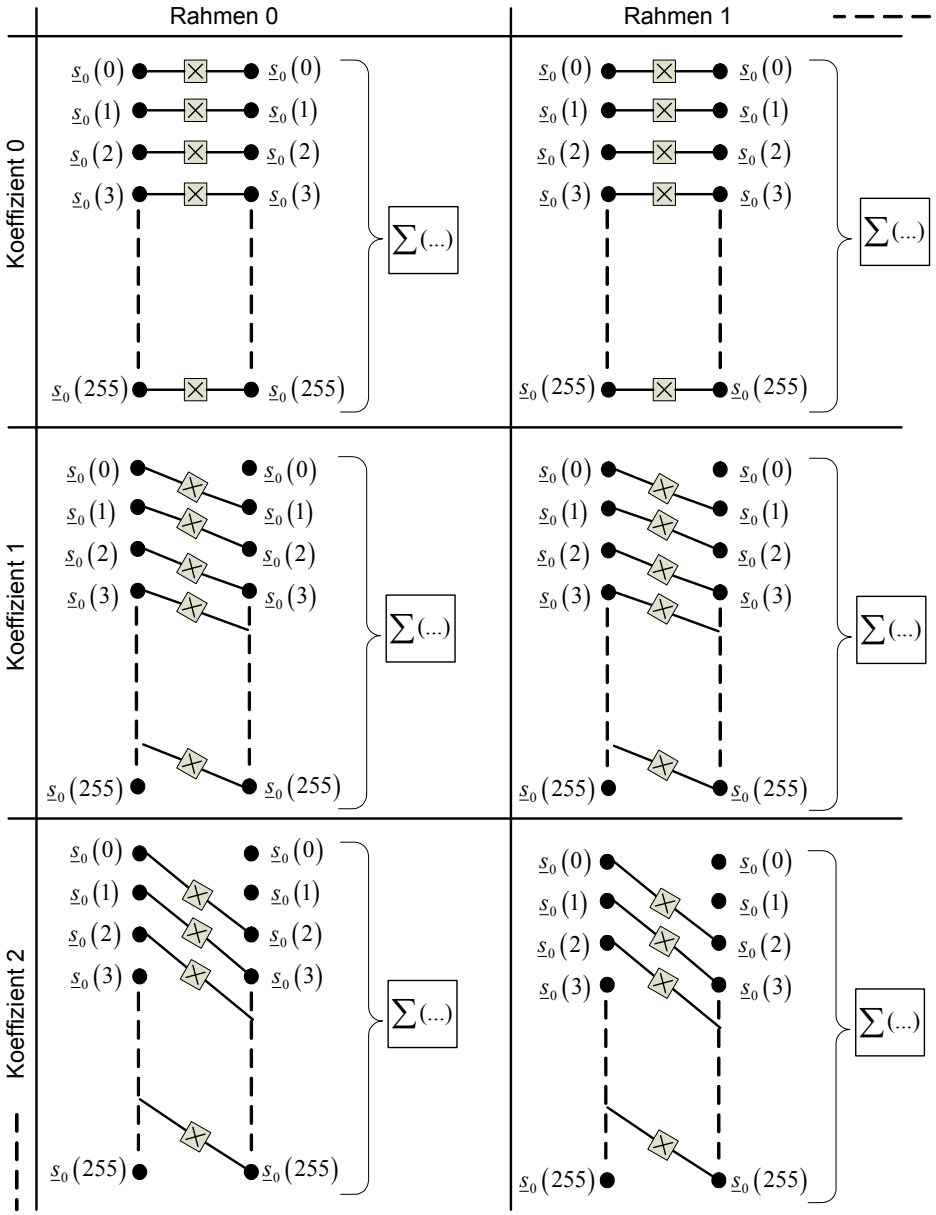


Abbildung 3-14 Berechnung der AKF Koeffizienten

Die Autokorrelationskoeffizienten können mit folgender Formel berechnet werden:

$$\underline{C}_k(p) = \sum_{i=0}^{WinWidth-p-1} \underline{s}_k(i) \times \underline{s}_k(i+p), \tag{3.8}$$

$$p = 0 \ldots CoeffOrder - 1,$$

$$k = 0 \ldots K, \; mit \; K = \frac{(Endpunkt - Anfangspunkt) - WinWidth}{WinShift}$$

Dabei ist $\underline{C}_k(p)$ der p-te Autokorrelationskoeffizient des Rahmens k.

Abbildung 3-14 erklärt wie die Autokorrelationskoeffizienten für jeden Rahmen berechnet werden. Es werden also insgesamt $p \times K$ Koeffizienten pro Referenzkommando berechnet.

Mit einer Kommandodauer von 2 Sekunden und einer Abtastfrequenz von 8 kHz beträgt die Anzahl der Abtastwerte 16000. Mit einer Fensterbreite von 256 Abtastwerten und einer Fensterverschiebung von 128 Abtastwerten beträgt der maximale Wert von K=124 Rahmen. Das entspricht 1240 Koeffizienten pro Kommando.

3.4.4 Erkennungsverfahren

In Kapitel 3.3 wurde erklärt, dass eine Trainingsphase und eine Erkennungsphase zum System gehören.

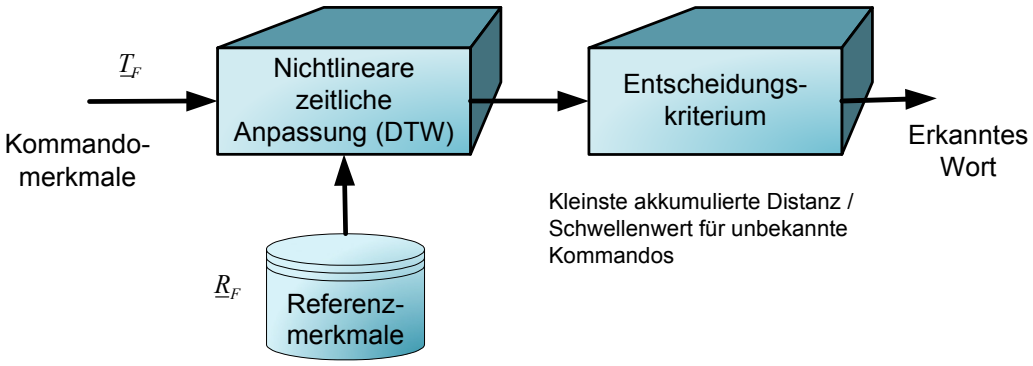


Abbildung 3-15 Erkennung eines Kommandos

Die aus der Trainingsphase gewonnenen Kommandomerkmale $\underline{R}_F$ werden in der Erkennungsstufe eingesetzt, um mit den aktuellen Kommandomerkmalen $\underline{T}_F$ verglichen zu werden (Abbildung 3-15 und Abbildung 3-16). Um das zu erreichen, wird in dieser Arbeit die Methode der nichtlinearen zeitlichen Anpassung (DTW) eingesetzt. Mit einem Entscheidungskriterium werden die Ergebnisse des DTW-Verfahren bewertet und eine entsprechende Ausgabe geliefert.

Das Prinzip des DTW-Verfahren wurde in Kapitel 2.1 erklärt und ist in Abbildung 3-16 zu sehen.

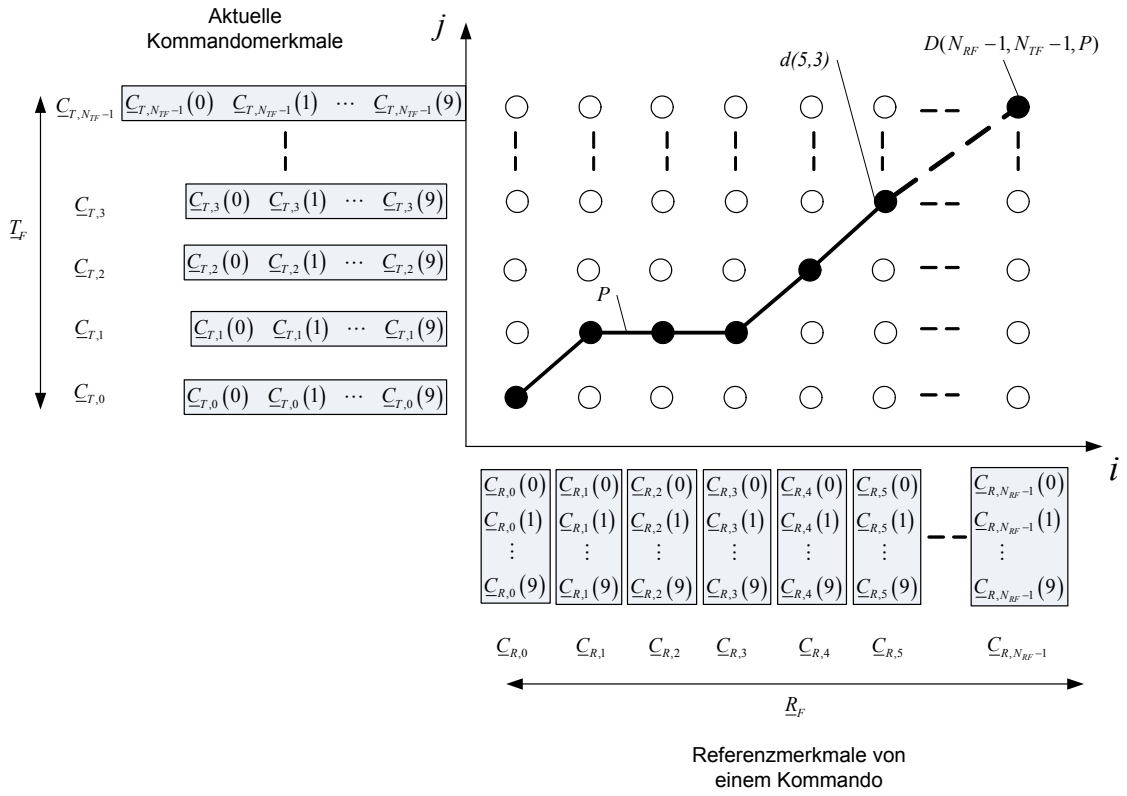


Abbildung 3-16 Prinzip des DTW-Verfahrens

N_{TF} ist die Anzahl der Rahmen der aktuellen Kommandomerkmale. N_{RF} ist die Anzahl der Rahmen eines Referenzkommandos. $d(i,j)$ ist die quadratische Euklidsche Distanz zwischen dem i-ten Vektor der Referenzmerkmale $\underline{C}_{R,i}$ und dem j-ten Vektor der aktuellen Merkmale $\underline{C}_{T,j}$.

$$d(i,j) = \sum_{p=0}^{CoeffOrder-1} \left(\underline{C}_{R,i}(p) - \underline{C}_{T,j}(p)\right)^2, \quad (3.9)$$
$$i = 0 \ldots N_{RF} - 1,$$
$$j = 0 \ldots N_{TF} - 1$$

Für einen Pfad P wird die akkumulierte Gesamtdistanz mit $D\left(N_{RF}-1, N_{TF}-1, P\right)$ bezeichnet. Sie ist die Summe aller Distanzen $d(i,j)$, die dem Pfad P folgen.

$$D\left(N_{RF}-1, N_{TF}-1, P\right) = \sum d\left(i,j\right)_P \quad (3.10)$$

Die Berechnung der Gesamtdistanzen aller möglichen Pfade entspricht allen möglichen Anpassungen der aktuellen Merkmale an die Referenzmerkmale. Der optimale Pfad P_{opt} entspricht somit der besten Anpassung bzw. der kleinsten

Gesamtdistanz $D_{Min}\left(N_{RF}-1, N_{TF}-1, P_{opt}\right)$. Im nächsten Unterkapitel wird die Methode der Dynamischen Programmierung erklärt, mit der man die Berechnung aller möglichen Pfade vermeidet.

3.4.4.1 Dynamische Programmierung und Pfadeinschränkungen

Es ist nicht nötig die Gesamtdistanzen aller möglichen Pfade zu berechnen, um die minimale Gesamtdistanz $D_{Min}\left(N_{RF}-1, N_{TF}-1, P_{opt}\right)$ zu finden. Die Methode der Dynamischen Programmierung ermöglicht die Ermittlung der minimalen Gesamtdistanz mit nur einem Berechnungsdurchgang durch die Matrix der Kommandomerkmale. Das bedeutet eine enorme Ersparung an Rechenzeit und Speicherbedarf.

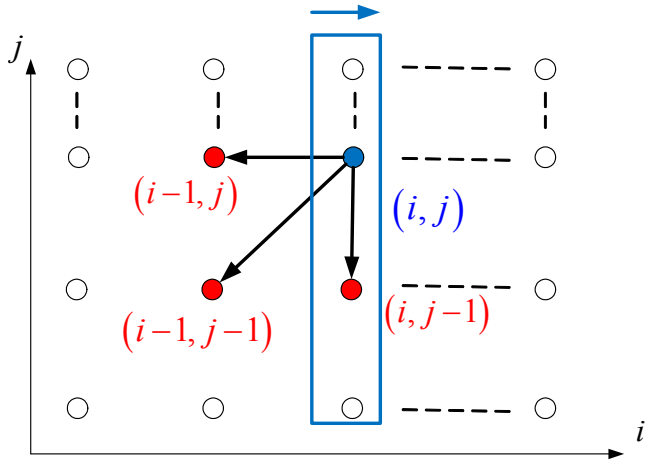


Abbildung 3-17 Prinzip der dynamischen Programmierung

Abbildung 3-17 zeigt das Prinzip der dynamischen Programmierung. Die Berechnung wird spaltenweise durchgeführt. Für jeden Punkt in der Merkmalmatrix wird der optimale lokale Pfad gesucht. Die in der Abbildung dargestellte Pfadeinschränkung hat sich als sehr praktisch erwiesen [Pla05].

Für einen Punkt (i, j) wird der optimale lokale Pfad nur aus den Vorgängern $(i-1, j)$, $(i, j-1)$ und $(i-1, j-1)$ gesucht. Damit wird ein diagonaler Verlauf erzielt. Es ist also die minimale akkumulierte Distanz bis zum Punkt (i, j) zu finden. Das erreicht man, wenn das Minimum aus folgenden Akkumulierungen berechnet wird:

$$D(i,j) = Min\begin{cases} D(i,j-1) + d(i,j) \\ D(i-1,j-1) + 2 \cdot d(i,j) \\ D(i-1,j) + d(i,j) \end{cases} \tag{3.11}$$

Beim Diagonalübergang vom Punkt $(i-1, j-1)$ ist die doppelte lokale Distanz $d(i,j)$ zu akkumulieren, damit die seitlichen Übergänge nicht immer benachteiligt werden.

Die lokale Pfadberechnung wird weitergeführt bis zur letzten Spalte. Der Nachteil bei der spaltenweise Bearbeitung ist, dass der Endpunkt $(N_{RF}-1, N_{TF}-1)$ nicht immer getroffen wird, sondern ein Punkt der letzten Spalte. Somit kann eine ideale Anpassung nicht immer erreicht werden. Dieser Punkt stellt die gesuchte minimale Gesamtdistanz D_{Min} dar.

Eine weitere Pfadeinschränkung stellt Abbildung 3-18 dar.

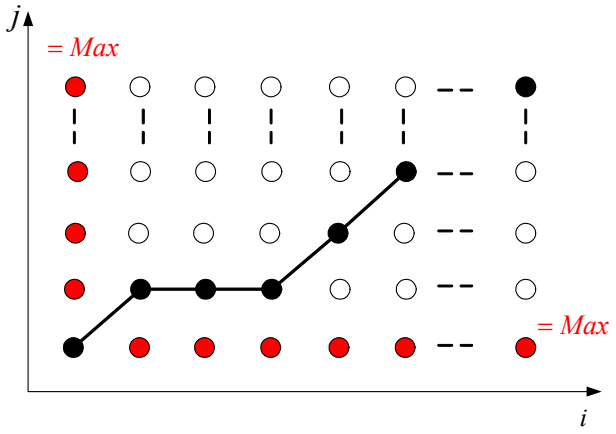


Abbildung 3-18 Pfadeinschränkung bei der dynamischen Programmierung

Damit der Pfad nicht über die Ränder verläuft, werden die Distanzwerte der 0-ten Spalte und Zeile auf Maximum gesetzt und somit bei der Suche nach der Minimaldistanz vermieden.

$$D(i,0) = Max, \quad i = 1 \cdots N_{RF} - 1 \tag{3.12}$$

$$D(0,j) = Max, \quad j = 1 \cdots N_{TF} - 1 \tag{3.13}$$

Da die Merkmalvektoren in zeitlicher Folge berechnet wurden, wird die Distanz zwischen den Vektoren $\underline{C}_{R,0}$ und $\underline{C}_{T,0}$ im Punkt $(0,0)$ in Abbildung 3-16 als

Pfadanfang übernommen. Damit die Länge der aktuellen Kommandomerkmale N_{TF} und der Referenzmerkmale N_{RF} das Ergebnis der minimalen Gesamtdistanz D_{Min} nicht beeinflusst, wird D_{Min} auf der Summe der beiden Längen normiert.

$$\tilde{D}_{Min} = \frac{D_{Min}}{\left(N_{RF} + N_{TF}\right)} \tag{3.14}$$

3.4.4.2 Entscheidungskriterium in der Erkennungsstufe

Wenn Kommandos wie „links“, „rechts“, etc. als Referenzkommandos gesprochen werden, so ist $\tilde{D}_{Min}(c)$ die minimale Gesamtdistanz zwischen dem derzeit gesprochenen und jedem einzelnen Referenzkommando.

Dann steht $Min\left\{\tilde{D}_{Min}(c)\big|_{c=1\cdots N_C}\right\}$ für die Distanz desjenigen Referenzkommandos, das dem derzeit gesprochenen Kommando am ähnlichsten ist. Dabei ist N_c die Anzahl der Referenzkommandos.

Schließlich ist

$$\arg\left\{Min\left\{\tilde{D}_{Min}(c)\big|_{c=1\cdots N_C}\right\}\right\} \tag{3.15}$$

das erkannte Kommando selbst.

Da keine anderen Informationen zur Verfügung stehen, entscheidet sich das DTW-Verfahren immer für das Kommando mit der kleinsten minimalen Gesamtdistanz. Führt man einen Distanzschwellenwert $MaxDist$ ein, könnten Ergebnisse ausgeschlossen werden, die über dem Schwellenwert liegen. Ein passender Schwellenwert muss aus Tests gewonnen werden.

3.5 Das Gesamtsystem

Abbildung 3-19 zeigt alle Systemkomponenten, die im Kapitel 3.4 vorgestellt wurden. Im nächsten Kapitel werden Algorithmen für die vorgestellten Verfahren entwickelt.

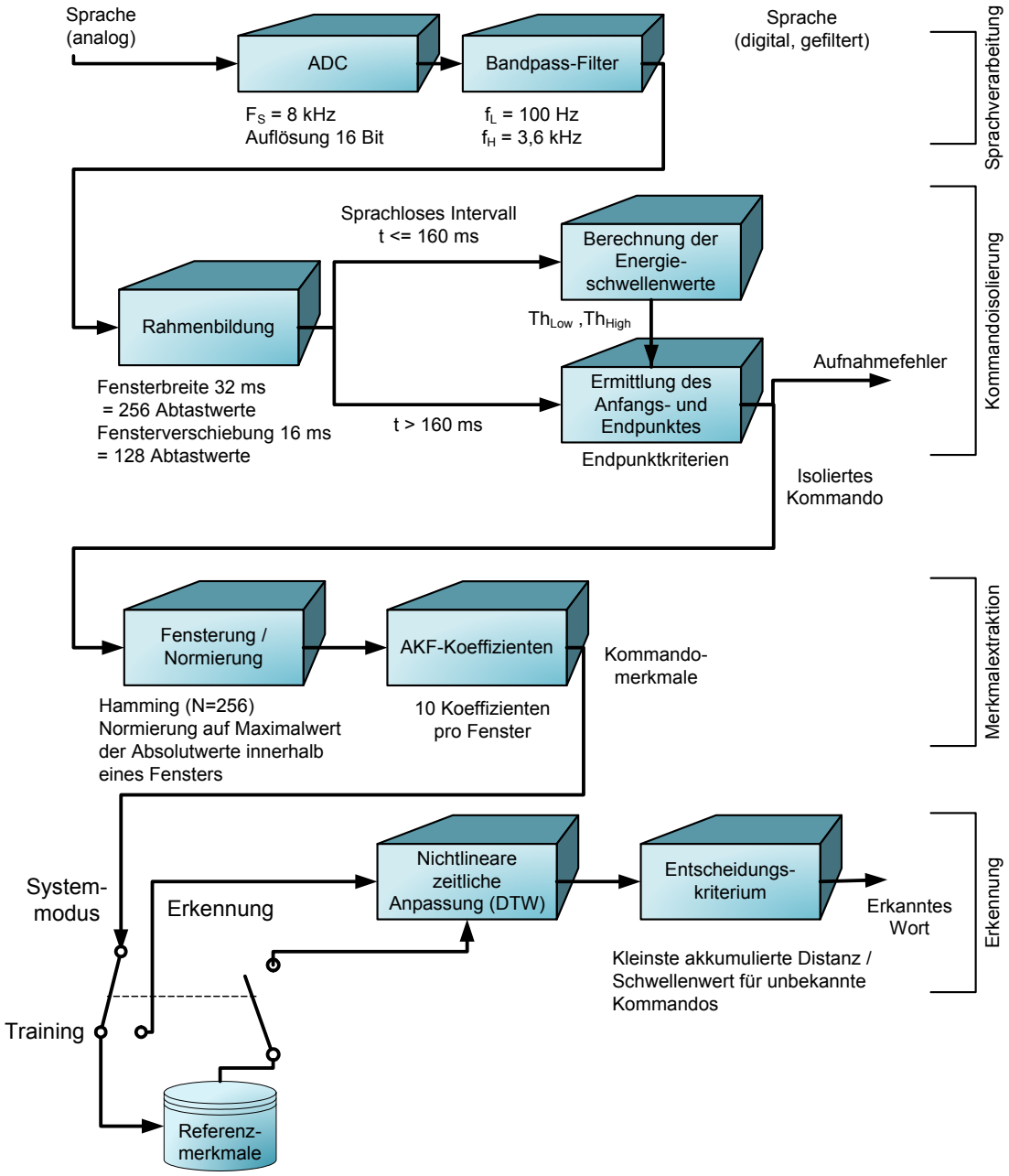


Abbildung 3-19 Gesamtsystem zur automatischen Spracherkennung

4 Entwicklung der Algorithmen

Ein Algorithmus[4] ist ein Rechenverfahren oder eine Rechenvorschrift, die eine endliche Folge von Arbeitsanweisungen umfasst, um ein bestimmtes Problem zu lösen.

Bei der Entwicklung und Simulation der Algorithmen wurde MATLAB verwendet. MATLAB wurde gewählt, weil es Vektor- und Matrizen-Operationen unterstützt und eine interaktive Umgebung für die Algorithmenentwicklung, Visualisierung und Analyse von Daten bietet. Außerdem lassen sich Algorithmen mit MATLAB einfacher und schneller simulieren und testen als mit herkömmlichen Programmiersprachen.

4.1 Algorithmusschritte

Im diesem Kapitel werden die Algorithmusschritte dargestellt, die auf den vorgestellten Verfahren basieren. Die Algorithmusschritte sind:

- Die Filterung der Abtastwerte.
- Die Kommandoisolierung.
- Die Berechnung der Kommandomerkmale.
- Die nichtlineare zeitliche Anpassung und Distanzberechnung.

In den nächsten Unterkapiteln werden diese Algorithmusschritte erklärt.

4.1.1 Filterung der Abtastwerte

Zunächst werden die Abtastwerte gefiltert. Die Bandpass-Filterung erfolgt durch zwei hintereinander geschaltete FIR-Filter. Das erste FIR-Filter ist ein Hochpass mit der Grenzfrequenz 100 Hz. Das zweite FIR-Filter ist ein Tiefpass mit der

[4] Das Wort Algorithmus ist die lateinische Umwandlung des arabischen Namens Al-Chaurizmi, entnommen aus der lateinischen Bearbeitung eines seiner Bücher über Rechenverfahren.

Grenzfrequenz 3600 Hz. Die Filterkoeffizienten wurden mit MATLAB berechnet. Die Abtastwerte können mit den gefilterten Abtastwerten überschrieben werden. Die Anzahl aller Abtastwerte NB_WAV_SAMPLES beträgt 16000, bei einem Aufnahmeintervall von 2 Sekunden und einer Abtastfrequenz von 8 kHz.

```
◦Eingabe der Abtastwerte
for Sample = 0 ... NB_WAV_SAMPLES-1
{
  ◦ Filterung
    FilteredwaveHP = FIR_HP(Sample)
    Sample = FIR_LP(FilteredwaveHP)
   ⋮
}
```

4.1.2 Kommandoisolierung

Mit den gefilterten Abtastwerten werden Rahmen gebildet. Mit jeder Rahmenverschiebung WIN_SHIFT kann ein Rahmen weitergegeben werden. Wenn also der aktuelle Abtastwert m ein Vielfaches der Rahmenverschiebung WIN_SHIFT wird und kein Fehler vorliegt, wird ein vollständiger Rahmen für weitere Verarbeitungen weitergegeben.

```
◦Rahmenbildung
if CommandErrorFlag = 0 und m = (N_WinShift*WIN_SHIFT)
{
  ◦ inkrementiere N_WinShift
   N_WinShift = N_WinShift + 1;

   ◦ weitere Verarbeitungen
   ⋮
}
```

Nach der Rahmenbildung folgt die Verarbeitung innerhalb des sprachlosen Intervalls NO_SPEECH_INTERVAL. Die Betragswerte innerhalb jedes Rahmens werden akkumuliert, um die quasi Energie gemäß (3.1) zu berechnen. Am Ende des sprachlosen Intervalls werden die Energieschwellenwerte gemäß (3.2) bis (3.5) ermittelt.

Nachdem die Energieschwellenwerte berechnet wurden, können die Endpunkte des Kommandos detektiert werden. Dazu wird die quasi Energie berechnet und die Kriterien überprüft, die in Kapitel 3.4.2.3 vorgestellt wurden.

◦Sprachloses Intervall
if m <= NO_SPEECH_INTERVAL
{
◦ ist ein Rahmen vollständig?
if m > WIN_SHIFT+WIN_WIDTH-1
{
◦ Akkumuliere die Betragwerte (quasi Energie) der letzten 256 Abtastwerte in einem Rahmen, (3.1)

$$SignalEnergy = \sum_{i=m-WIN\_WIDTH}^{m-1} |s(i)|$$

◦ finde Maximum und Minimum
MaxEnergy = Max{SignalEnergy}
MinEnergy = Min{SignalEnergy}

◦ Sprachloses Intervall zu Ende
if m = NO_SPEECH_INTERVAL
◦ berechne die Energieschwellenwerte, (3.2)...(3.5)
$I1 = 0.05 \times (MaxEnergy - MinEnergy) + MinEnergy$
$I2 = 4 \times MinEnergy$
if $(I1 < I2) \rightarrow EnergyThLow = I1$ else $EnergyThLow = I2$
$EnergyThHigh = 4 \times EnergyThLow$
}

Wie es in Kapitel 3.4.2.3 erklärt wurde, können bei der Detektierung der Endpunkte Aufnahmefehler entdeckt werden. Die Funktion DetectEndPoints(...) enthält alle Kriterien zur Endpunktdetektion. Sie liefert die Endpunkte und gibt auch den Erfolgstatus der Detektierung CommandErrorFlag zurück.

◦Außerhalb des sprachlosen Intervalls
if m > NO_SPEECH_INTERVAL
{
◦ Akkumuliere die Betragwerte (quasi Energie) der letzten 256 Abtastwerte
in einem Rahmen (3.1)

$$\text{SignalEnergy} = \sum_{i=m-WIN\_WIDTH}^{m-1} |s(i)|$$

◦ Aufruf der Funktion DetectEndPoints, um die Endpunkte zu detektieren
CommandErrorFlag = DetectEndPoints(...)
}

◦ die Funktion DetectEndPoints()

◦Energie überschreitet untere Schwelle?
if (SignalEnergy > EnergyThLow)
{
◦ zum ersten Mal?
if (LowThresholdFlag = 0)
◦ markiere die Stelle als temporären Anfangspunkt
EnergyTempStartPoint = aktuelle Stelle im Energieverlauf

◦ Energie überschreitet beide Schwellen?
if (SignalEnergy > EnergyThHigh)
{
◦ ein neuer Impuls liegt vor → setze letzten Endpunkt zurück
EnergyEndPoint = 0
EndPointFlag = 0
◦ Sonderfall [a] nicht erfüllt? → nehme den temporären Anfangspunkt
EnergyStartPoint = EnergyTempStartPoint
◦ Setze den Anfangspunktflag
StartPointFlag = 1

◦ ein Impuls ist schon vorhanden
Sonderfall [d] erfüllt → MaxPulseDist, sonst Fehler ausgeben

- Energie zum ersten Mal zwischen beiden Schwellen?
  - markiere die aktuelle Stelle als temporären Endpunkt und speichere eine Kopie davon
    EnergyEndPoint = aktuelle Stelle
    LastEndPoint = aktuelle Stelle

- Energie NICHT zum ersten Mal zwischen beiden Schwellen?
  - überprüfe die Sonderfälle [b] und [c] und gib ggf. Fehler aus

- Energie zum ersten Mal unter der unteren Schwelle?
  - Sonderfall [d] erfüllt → verwerfe aktuellen Impuls
  - sonst Impuls ist gültig
  - Sonderfall [c] erfüllt → Fehler ausgeben
  - sonst übernehme aktuelle Stelle als Endpunkt und speichere eine Kopie
    EnergyEndPoint = aktuelle Stelle im Energieverlauf
    LastEndPoint = aktuelle Stelle im Energieverlauf

Die aus dem Energieverlauf detektierten Endpunkte sind relativ zur Rahmenverschiebung WIN_SHIFT und liegen in der Rahmensmitte. Die Endpunkte müssen übersetzt werden, bevor sie in der Merkmalextraktion verwendet werden. Abbildung 4-1 gibt ein Beispiel zum Zusammenhang zwischen dem Rahmenindex und dem Abtastwertindex. Wenn der Anfangspunkt beispielsweise gleich 2 ist, entspricht das $3 \times \text{WIN\_SHIFT} = 384$ Abtastwerte. Da die Aufzählung bei Null anfängt, muss man davon 1 abziehen. Somit ist der Anfangspunkt bei 383. dasselbe gilt auch für den Endpunkt.

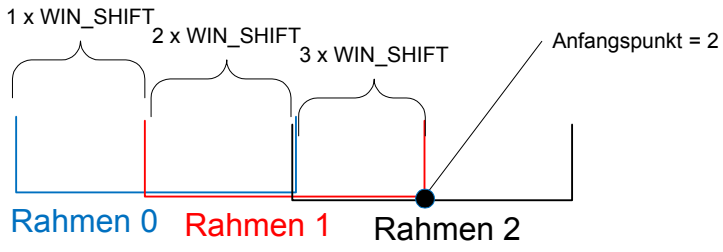


Abbildung 4-1 Übersetzung der Endpunkte

Die Endpunkte müssen also wie folgt übersetzt werden:

```
◦Übersetze die Endpunkte
  CmdStartPoint = ((EnergyStartPoint+1) × WIN_SHIFT)-1
  CmdEndPoint = ((EnergyEndPoint+1) × WIN_SHIFT)-1
```

4.1.3 Berechnung der Merkmale eines Kommandos

Nach dem ein Kommando isoliert wurde, können die Merkmale extrahiert werden. Wie es im Kapitel 3.4.3 erklärt wurde, besteht die Merkmalextraktion aus der Rahmenbildung, Fensterung, Normierung und Koeffizientenberechnung.

Bei der Rahmenbildung sind wieder dieselben Werte für WIN_SHIFT und WIN_WIDTH zu verwenden (128 und 256).

```
◦Rahmenbildung
  for m=CmdStartPoint ...CmdEndPoint, Schritt: WIN_SHIFT)
 {
   ⋮
```

Für die Fensterung werden die Abtastwerte jedes Rahmens kopiert und mit den Koeffizienten des Hamming-Fensters multipliziert (3.7). Die Koeffizienten TimeWin() werden aus MATLAB übernommen. Bei der Fensterung wird der maximale Betragswert innerhalb des Rahmens gesucht.

```
◦Fensterung
  for i = m-WIN_WIDTH ... m-1
 {
  Sample(i)=Sample(i)×TimeWin(i)
  SampleMax = Max{|Sample(i)|}
 }
```

Bei der Normierung werden alle gefensterten Abtastwerte auf den maximalen Betragswert normiert (3.7).

```
◦Normierung
  for i = 0...WIN_WIDTH-1
 {
   Sample(i)=Sample(i) / SampleMax
 }
```

Nach der Fensterung und Normierung eines Rahmens können die Merkmale nach der Vorschrift (3.8) berechnet werden. Für jeden Rahmen wird die Funktion CalcFrameFeatures(...) aufgerufen, mit der die Autokorrelationskoeffizienten berechnet werden. Für jeden Rahmen werden COEFF_ORDER Koeffizienten berechnet. COEFF_ORDER wurde auf 10 festgelegt.

◦CalcFrameFeatures(...)
for p=0 ... <COEFF_ORDER
{
◦ initialisierung
$C(p) = 0$
◦iteration
$C(p) = \sum_{i=0}^{WIN\_WIDTH-p-1} Sample(i) \times Sample(i+p)$
}

4.1.4 Nichtlineare zeitliche Anpassung und Distanzberechnung

In diesem Algorithmusschritt werden die aktuellen Kommandomerkmale an die Merkmale jedes Referenzkommandos angepasst. Dabei wird die Distanz zwischen beiden Merkmalgruppen berechnet. Die Funktion DTW_Recognition() wird für jedes Referenzkommando aufgerufen.

In der Funktion DTW_Recognition() wird als erstes der DTW-Algorithmus initialisiert. Gemäß (3.12) und (3.13) werden die 0-te Zeile und Spalte der Distanzmatrix auf Maximum gesetzt.

```
◦DTW_Recognition()
 ◦initialisiere die Ränder, (3.12), (3.13)
  for i=1...N_RF −1
  {
    ◦auf Maximum setzen
      distance(i,0) = MAXIMUM
  }
  for j=1...N_TF −1
  {
    ◦auf Maximum setzen
      distance(0,j) = MAXIMUM
  }
```

Zur Initialisierung gehört auch die Berechnung der Distanz $d(0,0)$, also die Distanz zwischen den 0-ten Vektoren der Referenzmerkmale $\underline{C}_{R,0}$ und der aktuellen Merkmale $\underline{C}_{T,0}$.

```
◦d(0,0)
  for i=0...COEFF_ORDER-1
  {
   d(0,0) = d(0,0) + (C_R,0(i) − C_T,0(i))^2
  }
```

Jetzt können die lokalen Distanzwerte $d(i,j)$ spaltenweise berechnet werden.

```
◦spaltenweise Verarbeitung nach (3.9)
for i=1...N_RF
{
 for j=1...N_TF
 {
   TempDistance = 0
   ◦berechne die lokale Distanz
     for p=0...COEFF_ORDER-1
     {
       TempDistance = TempDistance + (C_R,i(p) − C_T,j(p))^2
     }
```

◦ berechne die 3 lokalen Pfade nach (3.11)

$$\text{TempDistance1} = d(i\text{-}1,j) + \text{TempDistance}$$

$$\text{TempDistance2} = d(i\text{-}1,j) + 2 \times \text{TempDistance}$$

$$\text{TempDistance3} = d(i,j\text{-}1) + \text{TempDistance}$$

◦ finde das Minimum

$$d(i,j) = \text{Min}\{\text{TempDistance1}, \text{TempDistance2}, \text{TempDistance3}\}$$

Die minimale Gesamtdistanz D_{Min} muss nach (3.14) normiert werden.

$$\tilde{D}_{Min} = \frac{D_{Min}}{(N_{RF} + N_{TF})}$$

Für jedes Referenzkommando wird der DTW-Algorithmus ausgeführt, so dass das Kommando mit der kleinsten $\tilde{D}_{Min}$ ausgegeben wird.

$$erkanntes\ Kommando = \arg\left\{ Min\left\{ \tilde{D}_{Min}(c)\big|_{c=1\cdots N_C} \right\} \right\}$$

Ist die kleinste minimale Gesamtdistanz größer als der Distanzschwellenwert $MaxDist$, wird ein Fehler ausgegeben.

4.2 Simulation

Ein wichtiger Bestandteil der Softwareentwicklung ist die Simulation. Dadurch kann man die Funktionalität eines Algorithmus validieren und gegebenenfalls verbessern. Die Werte, die bestimmte Parameter annehmen sollen, können auch dadurch gewonnen werden.

4.2.1 MATLAB-Simulation

Die in den vorigen Abschnitten angeführten Algorithmusschritte wurden für die Simulation als M-Dateien in MATLAB programmiert. Das verwendete Zahlenformat ist „Double“.

Als Referenzkommandos wurden Kommandos gewählt, die für die Steuerung eines Roboters passend sind. Tabelle 4-1 stellt diese Referenzkommandos dar.

Typ	Kommando
Bewegung	*Vorwärts*
	Rückwärts
	Rechts
	Links
	Umdrehen
	Rechtsdrehen
	Linksdrehen
	Stopp
Quantität	*Schneller*
	Normal
	Langsamer
Name	*Hugo*

Tabelle 4-1 ReferenzKommandos

Mit `SpeechRecognition.m` steuert man die Simulation. Die Möglichkeit ein Training durchzuführen ist auch gegeben. Die Referenzmerkmale werden dann in der MAT-Datei `FrameFeaturesDataBase.mat` gespeichert. So können die Referenzmerkmale einfach eingelesen und geändert werden. In der MAT-Datei `CommandsDataBase.mat` sind die Abtastwerte von zwei Kommandosätzen als Beispiel, sowie die Kommandos und deren Sprecher in Textform gespeichert.

Zur Filterung wird die Datei `PreFilter.m` ausgeführt. Die Filterkoeffizienten der Hochpass- und Tiefpass-Filter werden in der Datei `CalcFilterCoeffs.m` berechnet und in der MAT-Datei `PreFilterCoeffs.mat` gespeichert, so dass sie bei der Filterung eingelesen werden.

Um die Energie und Energieschwellenwerte zu bestimmen wird die Datei `CalcEnergy.m` ausgeführt. Diese führt wiederum die Datei `DetectEndPoints.m` aus, um die Endpunkte zu detektieren.

Nach der Kommandoisolierung wird das Ergebnis bei fehlerhafter Aufnahme ausgegeben und bei erfolgreicher Isolierung graphisch dargestellt. Abbildung 4-2 zeigt ein Beispiel zur Isolierung des Kommandos „*Umdrehen*“. Dabei bezeichnet

die x-Achse die Rahmennummer und die y-Achse die quasi Energie. Der Benutzer hat dann die Möglichkeit, mit dem Training fortzufahren oder das aktuelle Kommando zu wiederholen.

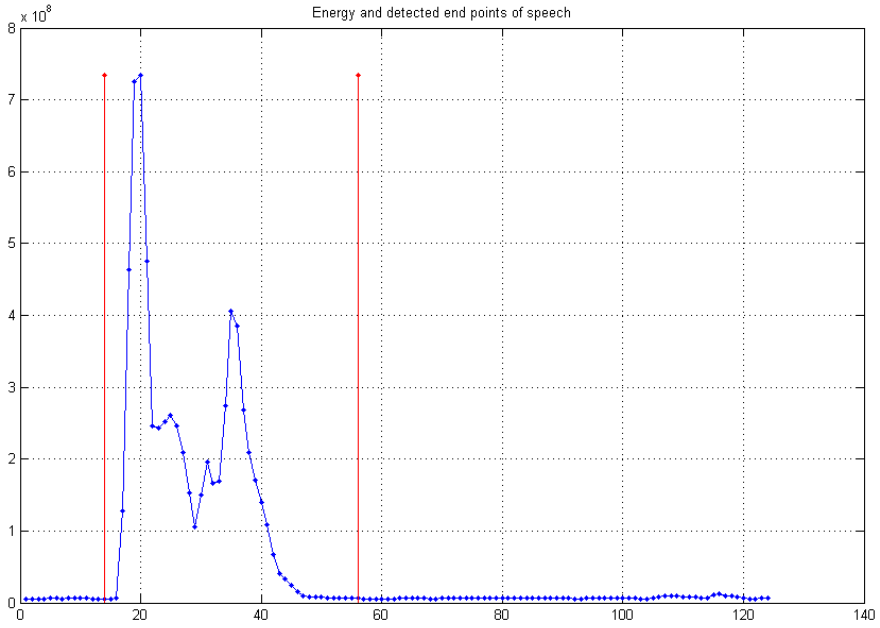


Abbildung 4-2 Isolierung des Kommandos ‚Umdrehen'

Die Fensterung und Normierung des isolierten Kommandos sowie die Berechnung der Kommandomerkmale werden in der Datei `CalcFrameFeatures.m` durchgeführt.

Der DTW-Algorithmus wird in der Datei `DTW_Recognition.m` berechnet. Das Ergebnis der Erkennung wird im MATLAB-Command Window ausgegeben.

Nach der Erkennung kann man die Datei `FindMaxDistance.m` ausführen, um statistische Werte über die minimale Gesamtdistanz zu gewinnen. Die Summe aus dem Mittelwert und der Standardabweichung kann einen solchen Wert näherungsweise liefern. Die Datei `ResultsAnalyse.m` stellt die Testergebnisse dar. Aus diesen Werten soll der Distanzschwellenwert $MaxDist$ festgelegt werden.

4.2.2 Simulationsergebnisse

Die Simulationsergebnisse zeigten, dass die ausgewählten Algorithmen für eine sprecherabhängige Einzelworterkennung sehr gut funktionieren. Abbildung 4-3 stellt das Simulationsergebnis graphisch dar.

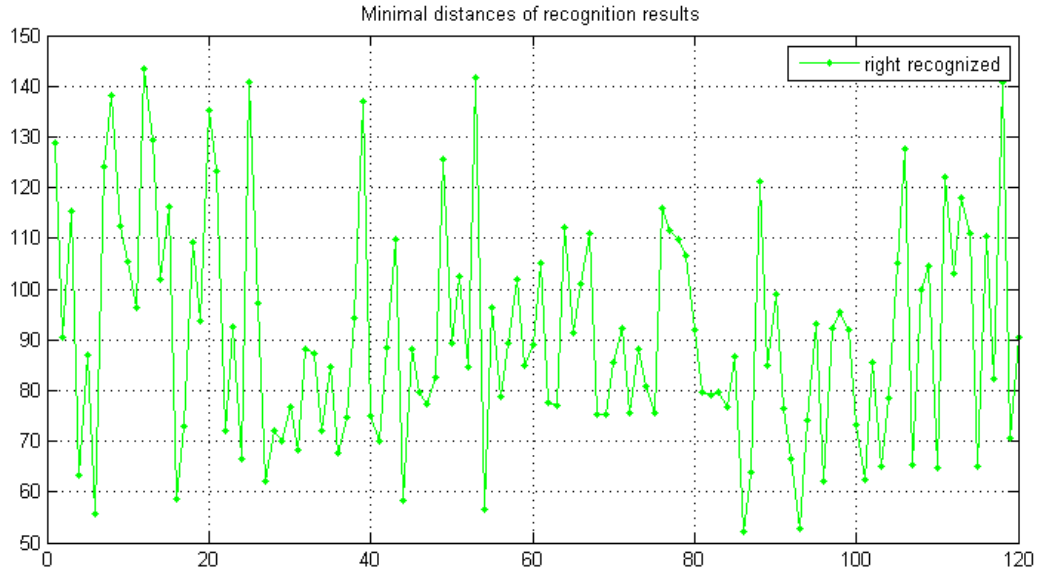


Abbildung 4-3 Simulationsergebnis

Ein Test mit 10 Versuchen zu verschiedenen Zeiten wurde durchgeführt, wobei jeder Versuch die 12 Kommandos enthält und vor jedem Versuch ein Training durchgeführt wurde. Aus den 120 Kommandos wurden alle Kommandos erkannt, was einer Erkennungsrate von 100% entspricht.

Die mittlere Gesamtdistanz war 91 und die Standardabweichung 22. Demzufolge wurde der Distanzschwellenwert $MaxDist$ auf 113 festgelegt. Mit diesem Wert konnten Kommandos mit relativ längerer Dauer oder eine normale Unterhaltung ausgeschlossen werden.

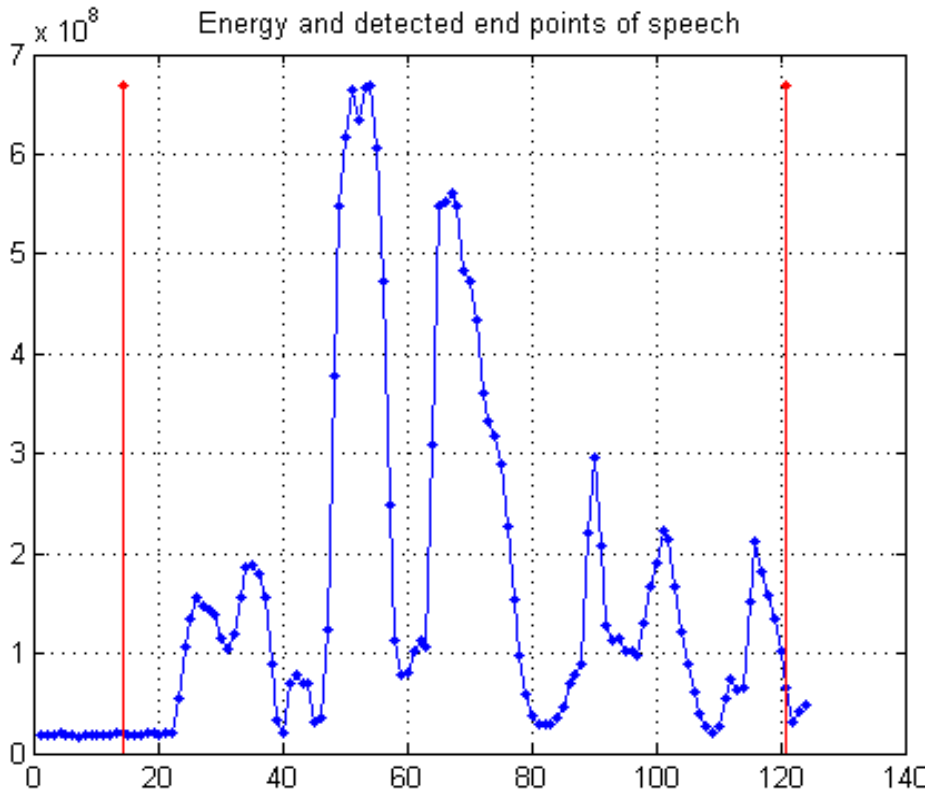


Abbildung 4-4 Beispiel zum Verlauf der quasi Energie von einer Unterhaltung

Abbildung 4-4 zeigt ein Beispiel dazu. Die minimale Gesamtdistanz betrug in diesem Beispiel 245,04 und der Fehler ‚Unknown command' wurde ausgegeben.

5 Implementierung der Algorithmen

Die entwickelten Algorithmen wurden in Modulen programmiert, um die Übersichtlichkeit zu erhöhen und die Weiterentwicklung zu vereinfachen. Die verwendete Programmiersprache ist C. Es wurden ausschließlich Integervariablen verwendet, um Rechenzeit zu sparen. Zur Evaluierung der Module wurde das TI DSK6713 Board mit dem Audio-Codec AIC23 und das TI D.MODULE.C6713 Board mit dem D.MODULE.PCM3003 Audio-Codec eingesetzt.

Abbildung 5-1 zeigt alle Dateien des entwickelten Systems und die darin enthaltenen Funktionen. Die Flussdiagramme der einzelnen Module sowie der Quellcode sind im Anhang (Flussdiagramme, C-Code) zusammengestellt.

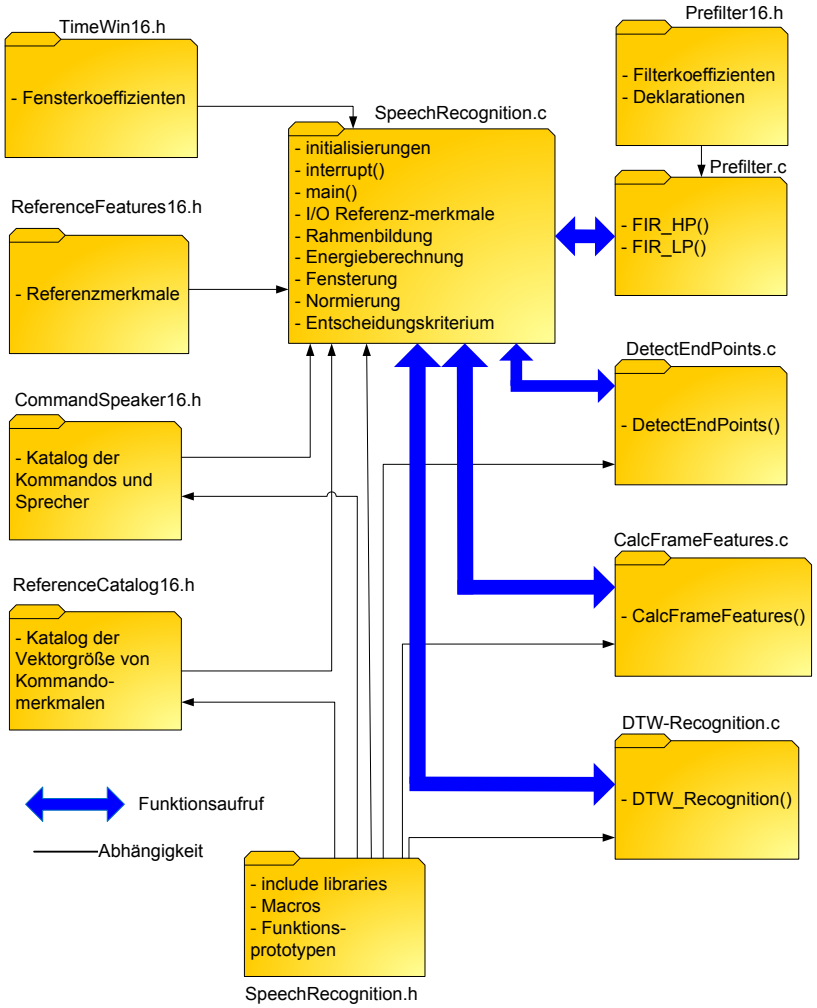


Abbildung 5-1 Modularer Aufbau des ASR-Systems

5.1 Verwendete Parameter

Bei der Entwicklung und Implementierung der Algorithmen wurden die in Tabelle 5-1 aufgelisteten Parameter verwendet.

Parameter	Beschreibung	Aktueller Wert
COEFF_ORDER	Anzahl der Merkmale innerhalb eines Rahmens	10
ENERGY_LENGTH	Anzahl der Rahmen (bei der Berechnung des Energieverlaufs)	NB_WAV_SAMPLES / WIN_SHIFT = 124
FS	Abtastfrequenz	8 kHz
MAX_DISTANCE	Maximal zulässige Gesamtdistanz	93
MAX_GROUP_SIZE	Anzahl der Sprecher von Referenzkommandos	1
MAX_PULSE_DIST	Maximal zulässige Distanz zwischen zwei auf einander folgenden Impulsen	3*WIN_WIDTH = 96 ms = 768 Abtastwerte
MAX_TH_DIST	Maximal zulässige Distanz zwischen den Stellen, an denen die Energieschwellenwerte überschritten werden	5*WIN_WIDTH = 160 ms = 1280 Abtastwerte
MAX_WORD_SIZE	Anzahl der Referenzkommandos	12
MAX_WORD_WIDTH	Maximal zulässige Breite bzw. Dauer eines Kommandos	50*WIN_WIDTH = 1,6 s = 12800 Abtastwerte
MIN_PULSE_WIDTH	Minimal zulässige Breite eines Energieimpulses	2*WIN_WIDTH = 64 ms = 512 Abtastwerte
NB_WAV_SAMPLES	Anzahl der Abtastwerte pro Aufnahmeintervall	RECORD_TIME*FS = 16000 Abtastwerte
NO_SPEECH_INTERVAL	Sprachloses Intervall	5*WIN_WIDTH = 160 ms = 1280 Abtastwerte
RECORD_TIME	Aufnahmeintervall	2 s
WIN_SHIFT	Rahmenverschiebung	6 ms*FS = 128 Abtastwerte
WIN_WIDTH	Rahmenbreite	32 ms*FS = 256 Abtastwerte

Tabelle 5-1 verwendete Parameter

Die Auswahl der Parameterwerte beruht auf einer Empfehlung der Literaturquellen, die aus den vorigen Kapiteln zu entnehmen sind. Die Parameter sind als Makros in der Header-Datei `SpeechRecognition.h` programmiert.

5.2 Einlesen der Abtastwerte

Die Abtastwerte werden durch eine Interrupt-Service-Routine übermittelt, die die aktuellen Abtastwerte vom Codec einliest und sie in einem Puffer ins SDRAM schreibt (`ShortRecordedWave[i]`).

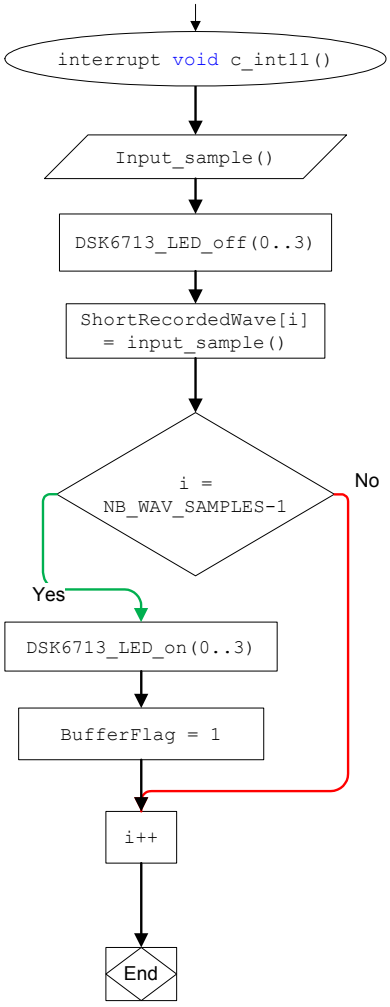


Abbildung 5-2 Flussdiagram der Interrupt-Service-Routine zum Einlesen der Abtastwerte

Die Größe des Puffers entspricht dem Aufnahmeintervall (16000). Wenn ein neues Intervall aufgenommen werden soll, wird der Interrupt aktiviert und die

Leuchtdioden ausgeschaltet. Mit der Zählervariable `i` werden die eingelesenen Abtastwerte gezählt. Ist der Puffer voll, wird ein Flag gesetzt (`BufferFlag`), damit das Hauptprogramm die Algorithmen auf die Abtastwerte anwendet. Solange das Hauptprogramm noch nicht bereit ist, neue Abtastwerte zu bearbeiten, wird der Interrupt deaktiviert und somit die Leuchtdioden (DSK6713_LED) angeschaltet. Abbildung 5-2 zeigt das Flussdiagramm der Interrupt-Service-Routine.

Wenn die Erkennung zu Ende ausgeführt wurde, wird der Interrupt reaktiviert, das Flag (`BufferFlag`) zurückgesetzt und die Zählervariable `i` auf Null gesetzt, damit ein neuer Puffer eingelesen werden kann.

5.3 Einlesen und Speichern der Referenzmerkmale

Die Referenzmerkmale (`Ref_Frame_Features[][][][]`) haben eine vierdimensionale Struktur. Es handelt sich bei der ersten Dimension um die Sprecher, bei der zweiten um die Kommandos, bei der dritten um die Rahmen und bei der vierten um die Autokorrelationskoeffizienten. Die Struktur der Referenzmerkmale wird in Abbildung 5-3 dargestellt. Es ist sehr wichtig, die Reihenfolge der Kommandos und Sprecher einheitlich zu definieren, damit das Ergebnis der Erkennung einem Sprecher und einem Kommando eindeutig zugeordnet werden kann.

In der Header-Datei `ReferenceFeatures16.h` werden die Referenzmerkmale gespeichert, damit sie später aus dieser Datei eingelesen werden. Auf jeder Zeile der Datei werden 10 Merkmale geschrieben (`COEFF_ORDER`). Das Einlesen der Referenzmerkmale nach jedem Training ist erforderlich. Dazu erscheint eine Eingabeaufforderung im Hauptprogramm. Das Hauptprogramm liest dann die Referenzmerkmale aus der Header-Datei ein und legt sie im SDRAM ab.

Damit man Rechenzeit spart, wird das DTW-Verfahren nicht über alle Rahmen angewendet, sondern nur auf die Rahmen, die die Merkmale des letzten Trainings enthalten. Daher ist ein Katalog notwendig, um zu wissen, auf wie viel Rahmen sich die tatsächlichen Merkmale eines Kommandos strecken. Der Katalog (`Ref_Frame_Catlg[][]`) wird aus dem Hauptprogramm erzeugt und ebenfalls in einer Header-Datei (`ReferenceCatalog16.h`) gespeichert

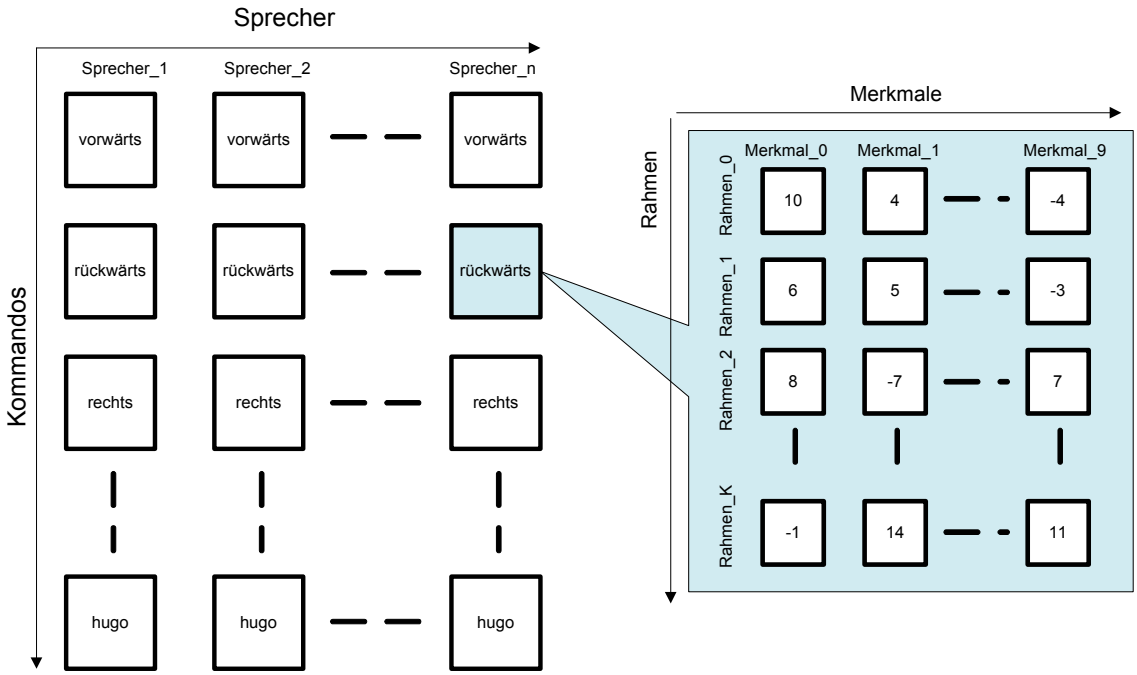


Abbildung 5-3 Struktur der Referenzmerkmale

5.4 Das Hauptprogramm (SpeechRecognition.c)

Das Hauptprogramm befindet sich in der Datei `SpeechRecognition.c`. Ähnlich wie bei der MATLAB-Simulation werden auch alle anderen Funktionen aus dieser Datei aufgerufen. Alle benötigten Parameter und Funktionsprototypen werden in der Header-Datei `SpeechRecognition.h` deklariert.

Im Hauptprogramm wird die Interrupt-Service-Routine gesteuert, um die Abtastwerte vom Kodierer einzulesen. Auch die Referenzmerkmale werden hier eingelesen und gespeichert. Ferner werden folgende Algorithmusschritte durchgeführt:

- Der Aufruf der Funktionen `FIR_HP()` und `FIR_LP()`, um die Vorfilterung durchzuführen. Damit werden die Eingangsabtastwerten im Vektor `ShortRecordedWave[]` mit den gefilterten Abtastwerten überschrieben.

- Die Rahmenbildung und die Berechnung des Energieverlaufs und der Energieschwellenwerte. Danach wird die Funktion `DetectEndPoints()` aufgerufen, um die Endpunkte eines Kommandos zu detektieren.
- Die Fensterung und die Normierung der einzelnen Rahmen. Dazu werden die Fensterkoeffizienten aus der Header-Datei `TimeWin16.h` eingelesen. Danach wird die Funktion `CalculateFrameFeatures()` aufgerufen, um die Kommandomerkmale zu extrahieren.
- Die Distanznormierung und das Entscheidungskriterium werden berechnet, nachdem die Funktion `DTW_Recognition()` für jedes Referenzkommando aufgerufen wurde.
- Das Ergebnis der Erkennung wird ausgegeben. Dazu wird die Header-Datei `CommandSpeaker16.h` als Katalog verwendet, die alle entsprechenden Kommandos und Sprecher in Textform enthält.

5.5 Funktionsschnittstellen

In folgenden Unterkapiteln werden die Schnittstellen der Funktionen dargestellt.

5.5.1 Vorfilterung (Prefilter.c)

Für die Bandpass-Filterung sind die Funktionen `FIR_HP()` und `FIR_LP()` in der Datei `Prefilter.c` vorgesehen. Die beiden Funktionen werden vom Hauptprogramm nacheinander aufgerufen. Die Funktion `FIR_HP()` besitzt folgende Schnittstelle:

`Short FIR_HP(`	
`Short FIR_HP_delays[],`	Vektor der verzögerten Abtastwerte
`Short FIR_HP_coe[],`	Vektor der Filterkoeffizienten
`Short N_HP_delays,`	Anzahl der Filterkoeffizienten
`Short x_HP)`	Eingangsabtastwert

In `Prefilter.c` wird die Header-Datei `Prefilter16.h` eingebunden, die die Koeffizienten und die Deklarationen enthält.

Die Funktion `FIR_LP()` besitzt eine entsprechende Schnittstelle.

5.5.2 Detektierung der Endpunkte (DetectEndPoints.c)

Nach der Berechnung der Energieschwellenwerte im sprachlosen Intervall wird die Energie von jedem Rahmen `SignalEnergy[]` berechnet und an die Funktion `DetectEndPoints()` übermittelt, um die Endpunkte zu detektieren. Es werden die Kriterien angewendet, die im Kapitel 3.4.2.3 erklärt wurden. Die Funktion `DetectEndPoints()` besitzt folgende Schnittstelle:

`unsigned short DetectEndPoints(`	
`unsigned short Ind,`	Index des aktuellen Rahmens
`short SignalEnergy[],`	Energievektor
`short EnergyThLow,`	untere Energieschwelle
`short EnergyThHigh,`	obere Energieschwelle
`unsigned short *EnergyEndPoint,`	Endpunkt
`unsigned short *EnergyStartPoint,`	Anfangspunkt
`unsigned short *LastEndPoint,`	letzter Endpunkt
`unsigned short *EndPointFlag,`	Flag für den Endpunkt
`unsigned short *StartPointFlag,`	Flag für den Anfangspunkt
`unsigned short *HighThresholdFlag,`	Flag für das Überschreiten der höheren Schwelle
`unsigned short *LowThresholdFlag,`	Flag für das Überschreiten der unteren Schwelle
`unsigned short *EnergyTempStartPoint,`	temporärer Anfangspunkt
`unsigned short *CheckDistFlag)`	Flag für die Überprüfung der Pulsbreite

Der Flag `CommandErrorFlag` wird zurückgegeben, um den Status der Detektion zu übermitteln.

5.5.3 Berechnung der Kommandomerkmale (CalcFrameFeatures.c)

Die Berechnung der Autokorrelationskoeffizienten als Kommandomerkmale wird mit der Funktion `CalcFrameFeatures()` durchgeführt. Die Funktion liefert für jeden Rahmen `COEFF_ORDER = 10` Koeffizienten. Sie besitzt folgende Schnittstelle:

`void CalcFrameFeatures(`	
`short shortTempWave[],`	Abtastwerte (ein Rahmen)
`short CoeffSet[])`	Vektor der AKF-Koeffizienten

Die berechneten Koeffizienten werden im Hauptprogramm im Vektor `FrameFeatures[][]` gesammelt, um entweder in einer Datei gespeichert zu werden (Trainingsphase) oder mit den Referenzmerkmalen verglichen zu werden (Erkennungsphase).

5.5.4 Nichtlineare zeitliche Anpassung (DTW_Recognition.c)

Die Funktion `DTW_Recognition()` berechnet die Gesamtdistanz zwischen den Merkmalen eines Referenzkommandos und den Merkmalen des aktuellen Kommandos. Sie wird also für jedes Referenzkommando aufgerufen. Sie besitzt folgende Schnittstelle:

`short DTW_Recognition(`	
`unsigned short xSizeRef,`	tatsächliche Größe der Merkmale des aktuellen Referenzkommandos
`short Ref_Frame_Features[][COEFF_ORDER],`	Merkmale des übergebenen Referenzkommandos
`unsigned short xSizeAct,`	tatsächliche Größe der Merkmale des aktuellen Kommandos
`Short Act_Frame_Features[][COEFF_ORDER])`	Merkmale des aktuellen Kommandos

Die berechnete Gesamtdistanz wird danach im Hauptprogramm normiert. Nachdem die minimale Gesamtdistanz ermittelt wurde, wird sie mit dem maximal zulässigen Distanzwert `MAX_DISTANCE` verglichen.

6 Testdurchführung

Um die Funktionalität des entwickelten Systems zu validieren, wurden Anforderungen für eine Benutzeroberfläche definiert, die in einer anderen Arbeit realisiert wurde [Sem08]. Die Oberfläche enthält ein Trainings- und Erkennungsmodus (Abbildung 6-1). Im Erkennungsmodus kann man wählen, ob die aus dem Trainingsmodus gewonnenen Merkmale ins SDRAM geladen werden. Nach der Erkennung wird das Ergebnis auch angezeigt.

Die Oberfläche kann somit für die Überprüfung der Referenzkommandos verwendet werden, bevor diese auf dem DSP im Roboter übertragen werden.

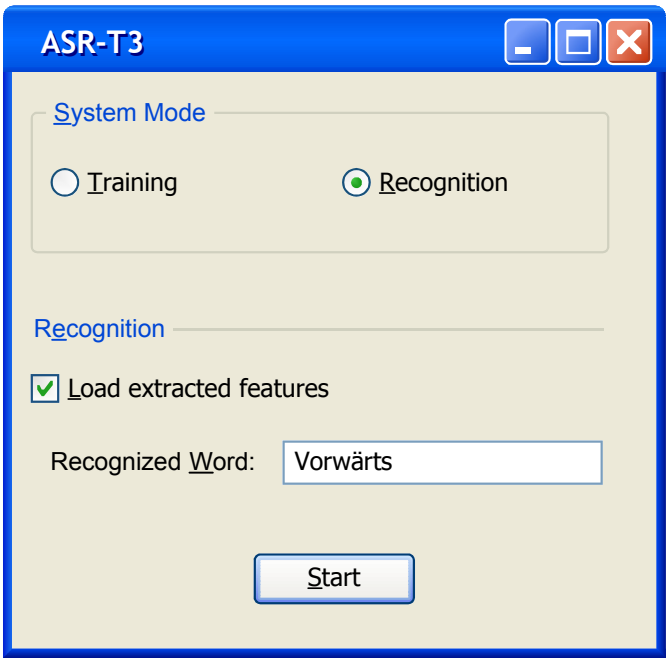


Abbildung 6-1 Benutzeroberfläche zum Durchführen von Trainings und Tests

Die Kommunikation zwischen der Oberfläche und dem DSP läuft in Echtzeit mit dem RTDX von Code Composer Studio.

Da die Oberfläche zum Zeitpunkt der Testdurchführung noch nicht zur Verfügung stand, wurde der Test „manuell“ durchgeführt. Dafür musste z.B. die Variable `SystemMode` im Code von Hand geändert werden. Das Ergebnis wurde aus `CommandSpeaker16.h` mit `Printf`-Anweisungen ausgegeben.

Derselbe Test aus der MATLAB-Simulation wurde hier durchgeführt. Es wurden 10 Versuche zu verschiedenen Zeiten durchgeführt, wobei jeder Versuch die 12 Kommandos enthält und vor jedem Versuch ein Training durchgeführt wurde. Aus den 120 Kommandos wurden 117 erkannt, was einer Erkennungsrate von 97,5% entspricht.

In der MATALB-Simulation wurde eine höhere Erkennungsrate erreicht. Der Grund zum Unterschied ist auf das verwendete Zahlenformat „short“ im DSP zurückzuführen. Die Verwendung von Integervariablen beeinträchtigt die Genauigkeit der Normierung (Divisionsergebnis) bei der Berechnung von Kommandomerkmalen und der Distanznormierung.

Die mittlere Gesamtdistanz war 74,65 und die Standardabweichung 18,32 Demzufolge wurde der Distanzschwellenwert $MaxDist$ auf 93 festgelegt. Mit diesem Wert konnten Kommandos mit relativ längerer Dauer und eine normale Unterhaltung ausgeschlossen werden.

Um die Verarbeitungsgeschwindigkeit zu messen, wurde die Anzahl der Prozessorzyklen durch „Code Profiling“ in Code Composer studio ermittelt. Für das Kommando ‚*Umdrehen*’ waren 151.370.903 Zyklen nötig. Mit einer Prozessorgeschwindigkeit von 300MHz beträgt die Verarbeitungsgeschwindigkeit 505ms. Es ist zu beachten, dass in dieser Zeit auch die Ausgabe des Ergebnisses durch mehrere `Printf`-Anweisungen enthalten ist. Man kann davon ausgehen, dass das System bei einem längeren Kommando wie ‚*Linksumdrehen*’ nach etwa 500ms antwortet. Somit ist die Echtzeitfähigkeit gegeben.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein System zur automatischen Spracherkennung entwickelt. Das System kann somit einen Roboter steuern. Die Anforderung an das System waren die Echtzeitfähigkeit und die Implementierung auf dem DSP.

Zur Auswahl der Erkennungsverfahren wurden zwei Verfahren vorgestellt:

- Die Hidden Markov Modelle (HMM).
- Die nichtlineare zeitliche Anpassung (DTW).

In dieser Arbeit wurde das DTW-Verfahren ausgewählt, weil es sich für Echtzeitimplementierung auf dem DSP besser eignet. Der Nachteil des DTW-Verfahren ist die Sprecherabhängigkeit. Da es sich um eine Robotersteuerung handelt, wurde eine Einzelworterkennung ausgewählt.

Das entwickelte System besteht aus einer Vorverarbeitungsstufe und einer Erkennungsstufe. In der Vorverarbeitungsstufe wird die digitalisierte Sprache aus dem Audio-Codec eingelesen (Abtastfrequenz: 8 kHz). Die Abtastwerte werden im DSP in einem Puffer gespeichert, der ein Aufnahmeintervall von 2 Sekunden zulässt. Aus den Abtastwerten wird das gesprochene Kommando detektiert und isoliert. Dazu werden die Abtastwerte gefiltert (Bandpass, Durchlassbereich von 100 bis 3600 Hz), um störende Frequenzen und Aufnahmegeräusche rauszufiltern. zur Detektion der Endpunkte eines Kommandos werden die Abtastwerte in Rahmen aufgeteilt (Rahmenbreite: 32 ms, Rahmenverschiebung: 16 ms). Für die Kommandoisolierung wird vorausgesetzt, dass ein sprachloses Intervall (160 ms) dem Kommando vorangeht. Während dieses Intervalls werden Energieschwellenwerte ermittelt. Danach wird der Energieverlauf berechnet und entsprechende Kriterien angewendet, um die Endpunkte eines Kommandos zu detektieren. Ferner werden die Merkmale eines Kommandos in der Vorverarbeitungsstufe extrahiert.

In dieser Arbeit wurden die Autokorrelationskoeffizienten (10 Koeffizienten pro Rahmen) als Merkmale ausgewählt. Vor der Merkmalextraktion werden die

Abtastwerte gefenstert (Hamming-Fenster) und auf dem Maximalwert innerhalb des Fensters normiert, um den Einfluss unterschiedlicher Amplituden bei der Spracheingabe zu eliminieren.

Das System besitzt zwei Phasen (die Trainingsphase und die Erkennungsphase) und ein Wörterbuch, das Referenzkommandos enthält. In der Trainingsphase werden die Merkmale jedes Referenzkommandos berechnet. Alle Referenzmerkmale werden in einer Datei gespeichert. In der Erkennungsphase müssen die Referenzmerkmale einmalig nach dem Einschalten der Versorgungsspannung aus einer Datei ins SDRAM hochgeladen werden. Danach werden die Merkmale des aktuellen Kommandos berechnet. Der DTW-Algorithmus berechnet die Distanz zwischen den aktuellen Merkmalen und den Merkmalen der einzelnen Referenzkommandos. Das Kommando mit der minimalen Distanz wird als erkannt erklärt oder eine entsprechende Fehlermeldung wird ausgegeben.

Die Algorithmen wurden zuerst in MATLAB simuliert, um die Funktionalität zu testen. Dabei wurde das Zahlenformat „double“ verwendet. Es konnte in der MATALB-Simulation eine Erkennungsrate von 100% erreicht werden.

Die Algorithmen wurden danach in C umgeschrieben und auf dem DSP implementiert. Alle verwendeten Variablen sind Ganzzahlvariablen, damit Rechenzeit gespart wird. Die gesamte Verarbeitungsgeschwindigkeit beträgt mit einem Wörterbuch von 12 Kommandos 500ms. Somit konnte ein echtzeitfähiges System realisiert werden. Hier muss gesagt werden, dass die Verarbeitungs-geschwindigkeit mit der Größe des Wörterbuchs zunimmt. Die erreichte Erkennungsrate betrug 97,5%.

Der Unterschied zwischen der DSP-Lösung und der MATALB-Simulation bei der Erkennungsrate ist allein auf das verwendete Zahlenformat „short“ im DSP zurückzuführen. Die Verwendung von Ganzzahlvariablen beeinträchtigt die Genauigkeit der Berechnung von Kommandomerkmalen und der Distanznormie-rung. Es konnte auch festgestellt werden, dass die Auswahl der

Referenzkommandos die Erkennungsrate direkt beeinflusst. Je besser die Referenzkommandos sind, desto höher wird die Erkennungsrate.

Im Folgenden werden einige Vorschläge zur Verbesserung des Systems genannt.

Um die Erkennungsrate zu erhöhen, kann man mehrere Referenzkommandos für einen Sprecher verwenden.

Die Autokorrelationskoeffizienten können auf LPC-Koeffizienten [RL81] erweitert werden, oder stattdessen die MFCC-Koeffizienten [Pla05] berechnet werden, da diese beiden Methoden zu höheren Erkennungsraten führen sollen.

Ferner sollten andere Methoden der Distanzberechnung hinsichtlich der Erkennungsrate evaluiert werden. Diese können die KNN-Methode, die Mahalanobis-Distanz oder die Kovarianz-Methode [RL81] [Pla05] sein.

Zuletzt kann die Anzahl der Kommandomerkmale erhöht werden und zusätzlich andere Merkmale hinzugefügt werden. Als Vorschlag sind 12 Koeffizienten statt 10 zu berechnen und dazu die Energie und die Nulldurchgangsrate hinzuzufügen, so dass 14 Koeffizienten pro Rahmen berechnet werden. Dadurch erhöht sich natürlich die Rechenzeit.

Um die Rechenzeit zu reduzieren, kann man beim DTW-Verfahren verschiedene Pfadeinschränkungen probieren, damit die Berechnung der Gesamtdistanz schneller erfolgt. Auch kann man einen Schwellenwert einführen, damit die Berechnung bei einigen Referenzkommandos früher gestoppt wird, ohne dass sie bis zum Ende durchgeführt werden muss („RejectionThreshold“ [RL81] [Pla05]).

Um die Sprecherunabhängigkeit zu erreichen, müssen alle Benutzerzielgruppen in den Referenzkommandos berücksichtigt werden. Da das zu einer enormen Rechenzeit führt, wurden Techniken entwickelt, um die Größe der Referenzmerkmale zu reduzieren. Dies kann man erreichen, indem man den Mittelwert der verschiedenen Merkmale ermittelt oder die Merkmale gruppiert (Clustering methods [RA79]).

8 Literaturverzeichnis

[Cha05] R. Chassaing: Digital Signal Processing and Applications with the 6713 and C6416 DSK.
Wiley, New Jersey, 2005

[Fin07] K. Fink: Skriptum Spracherkennung.
Universität Dortmund, Wintersemester 2006 / 2007.

[Ita75] F. Itakura: Minimum Prediction Residual Principle Applied to Speech Recognition.
IEEE Transcations on coustics, speech, and signal processing, February 1975

[LR83] S. E. Levinson, L. R. Rabiner, M. M. Sondhi: Speaker Independent Isolated Digit Recognition Using Hidden Markov Models
ICASSP 83, BOSTON

[Mar90] J. Mariani : Reconnaissance automatique de la parole, progrès et tendances.
Traitement du Signal, volume 7, nr 4. LIMSI-CNRS, 1990.

[Nua08] www.nuance.de, 6. Februar 2008

[Pla05] B. Plannerer: Skriptum An Introduction to Speech Recognition.
Institut für Phonetik und sprachliche Kommunikation.
Universität München, März 2005.

[RA79] L. R. Rabiner, S. E. Levinson, A. E. Rosenberg, J. G. Wilpon: Speaker-Independent Recognition of Isolated Words Using Clustering Techniques.
IEEE Transcations on acoustics, speech, and signal processing, August 1979

[Rab89] L. R. Rabiner: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.
Proceedings of the IEEE, VOL. 77, No.2, February 1989

[RJ04] L. R. Rabiner, B. H. Juang: Automatic Speech Recognition – A Brief History of the Technology Development.
Georgia Institute of Technology, Atlanta, Rutgers University and the University of California, Santa Barbara. 2004

[RL81] L. R. Rabiner, S. E. Levinson: Isolated and Connected Word Recognition - Theory and Selected Applications.
IEEE Transcations on communications, Vol. Com-29, No. 5, May 1981

[RS75] L. R. Rabiner, M. R. Sambur, An Algorithm for Determining the Endpoints of Isolated Utterances.
The Bell System Technical Journal, February 1975.

[Rus94] G. Ruske: Automatische Spracherkennung.
Methoden der Klassifikation und Merkmalextraktion.
2. Auflage, Oldenbourg 1994.

[Sic83] k. Sickert: Automatische Spracheingabe und Sprachausgabe.
Analyse, Synthese und Erkennung menschlicher Sprache mit digitalen Systemen.
Markt und Technik, München 1983.

[ST01] E. G. Schukat-Talamazzini: Skriptum Automatische Spracherkennung.
Friedrich-Schiller-Universität Jena, Wintersemester 2001 / 2002.

[ST95] E. G. Schukat-Talamazzini: Automatische Spracherkennung.
Grundlagen, statistische Modelle und effiziente Algorithmen.
Vieweg, Wiesbaden, 1995.

[VHH98] P. Vary, U. Heute, W. Hess: Digitale Sprachsignalverarbeitung.
B. G. Teubner, Stuttgart 1998.

[Wik08] http://de.wikipedia.org/wiki/Spracherkennung, 5. Februar 2008

[Sem08] J. Semedrea: Automated self-adjusting GUI screen layout generation für real-time data exchange (RTDX) applications running on Texas Instruments TMS320C6713.
Bachelorarbeit, Hochschule für angewandte Wissenschaften Hamburg, Sommersemester 2008.

9 Anhang

9.1 Flussdiagramme

Sämtliche Flussdiagramme wurden mit MS VISIO und Visustin erstellt.

9.1.1 Interrupt-Service-Routine, c_int11()

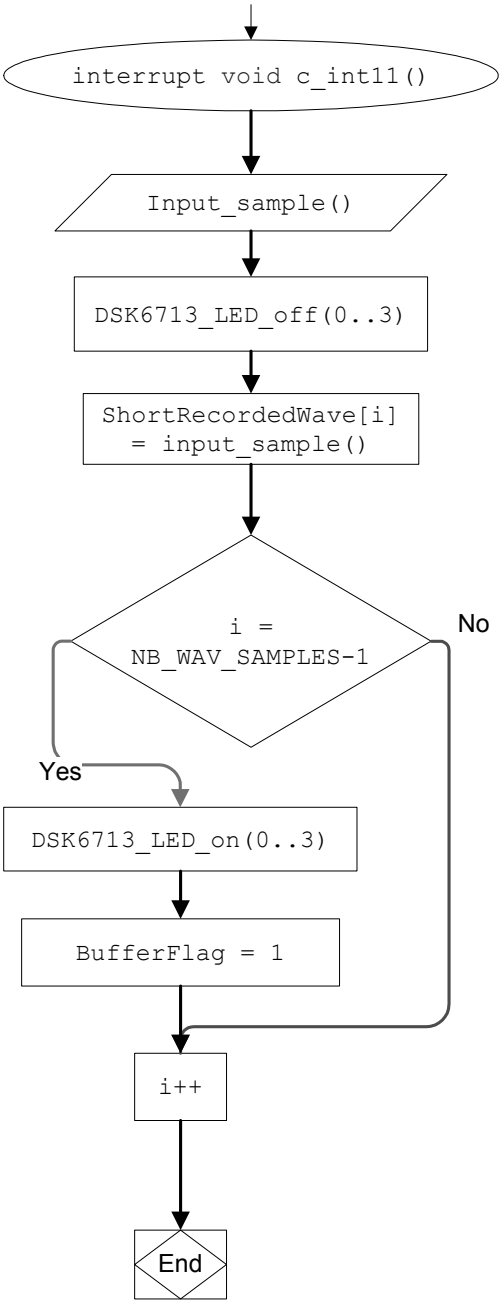


Abbildung 9-1 Flussdiagramm ISR

9.1.2 Hauptprogramm - Main()

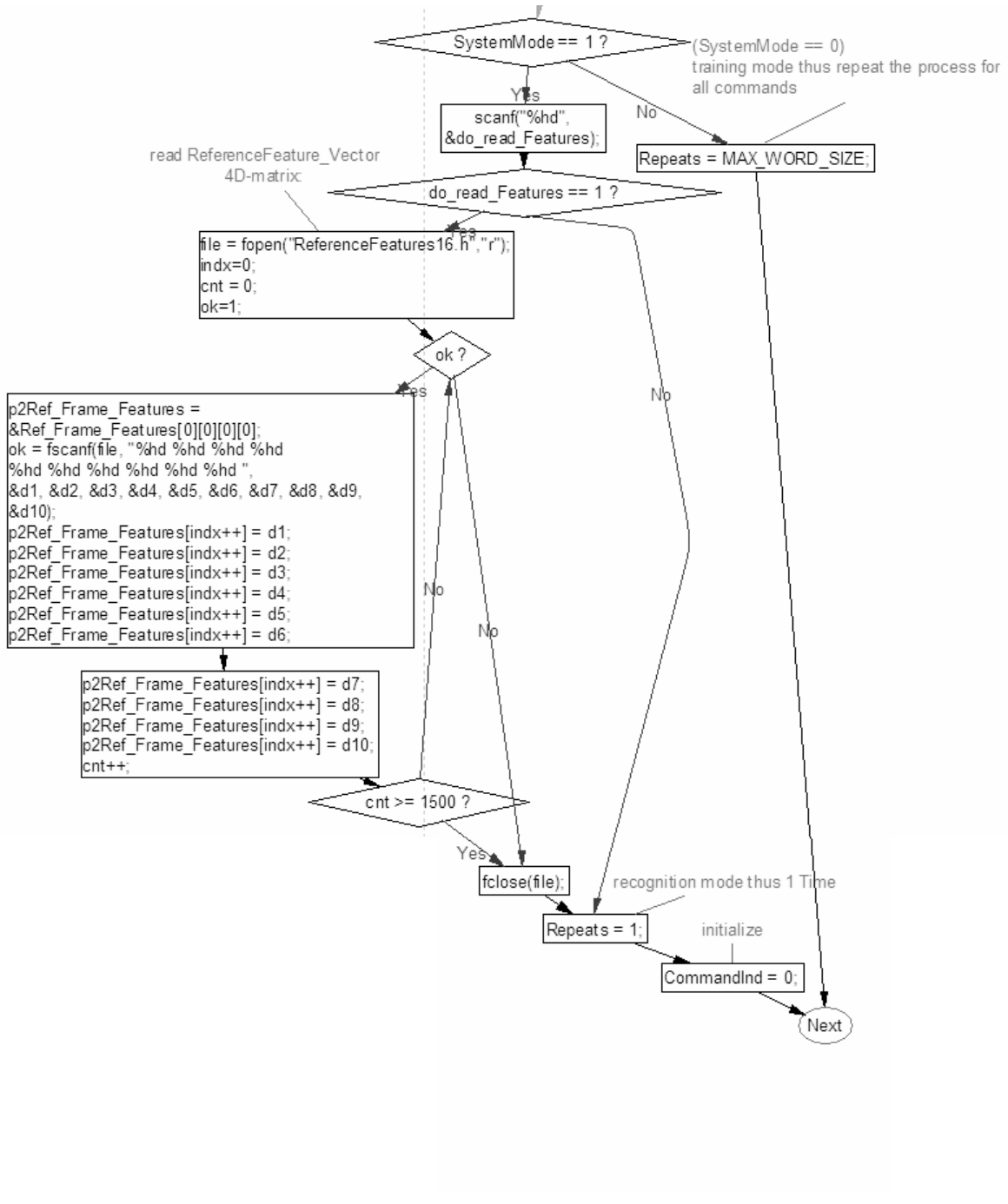
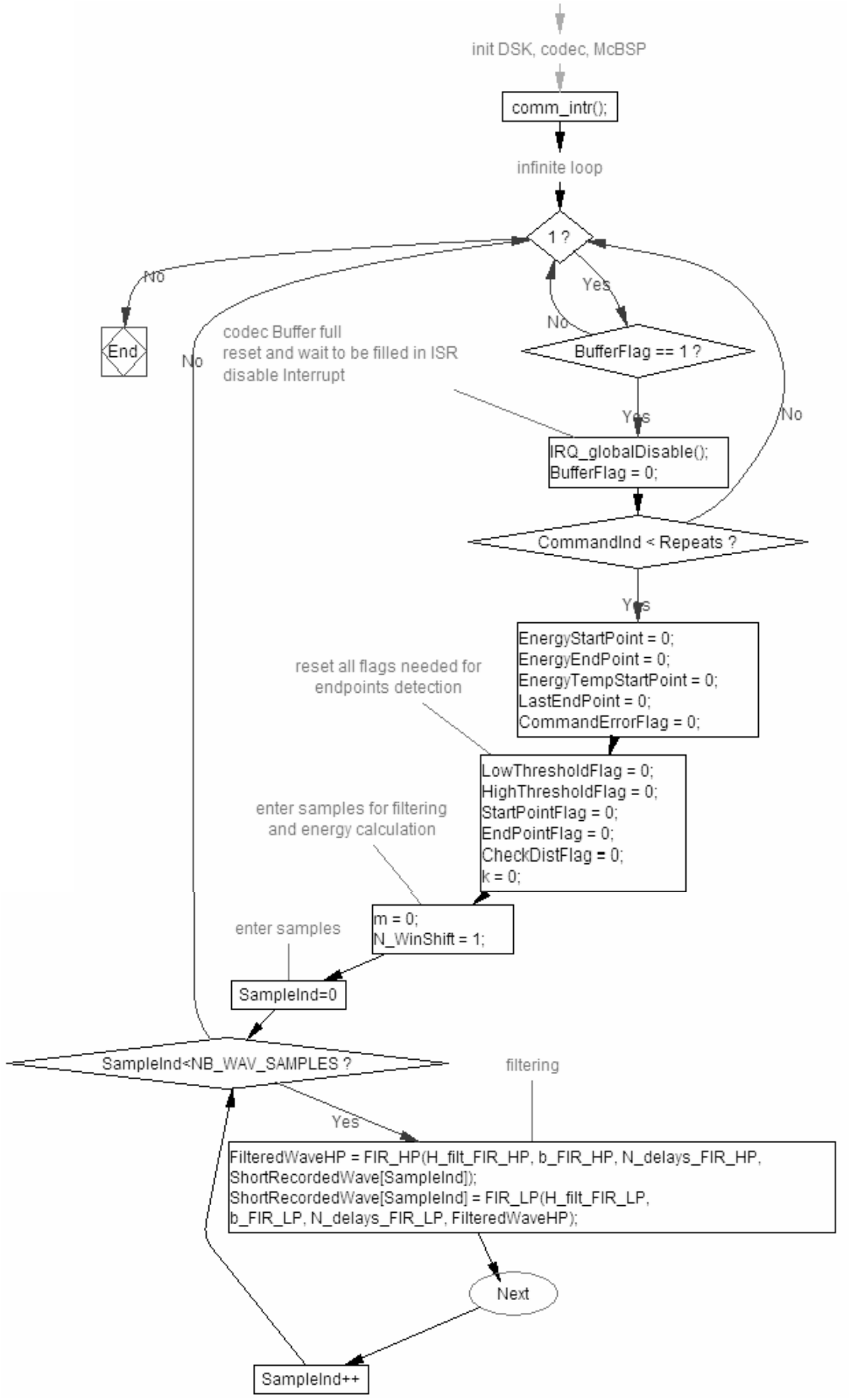
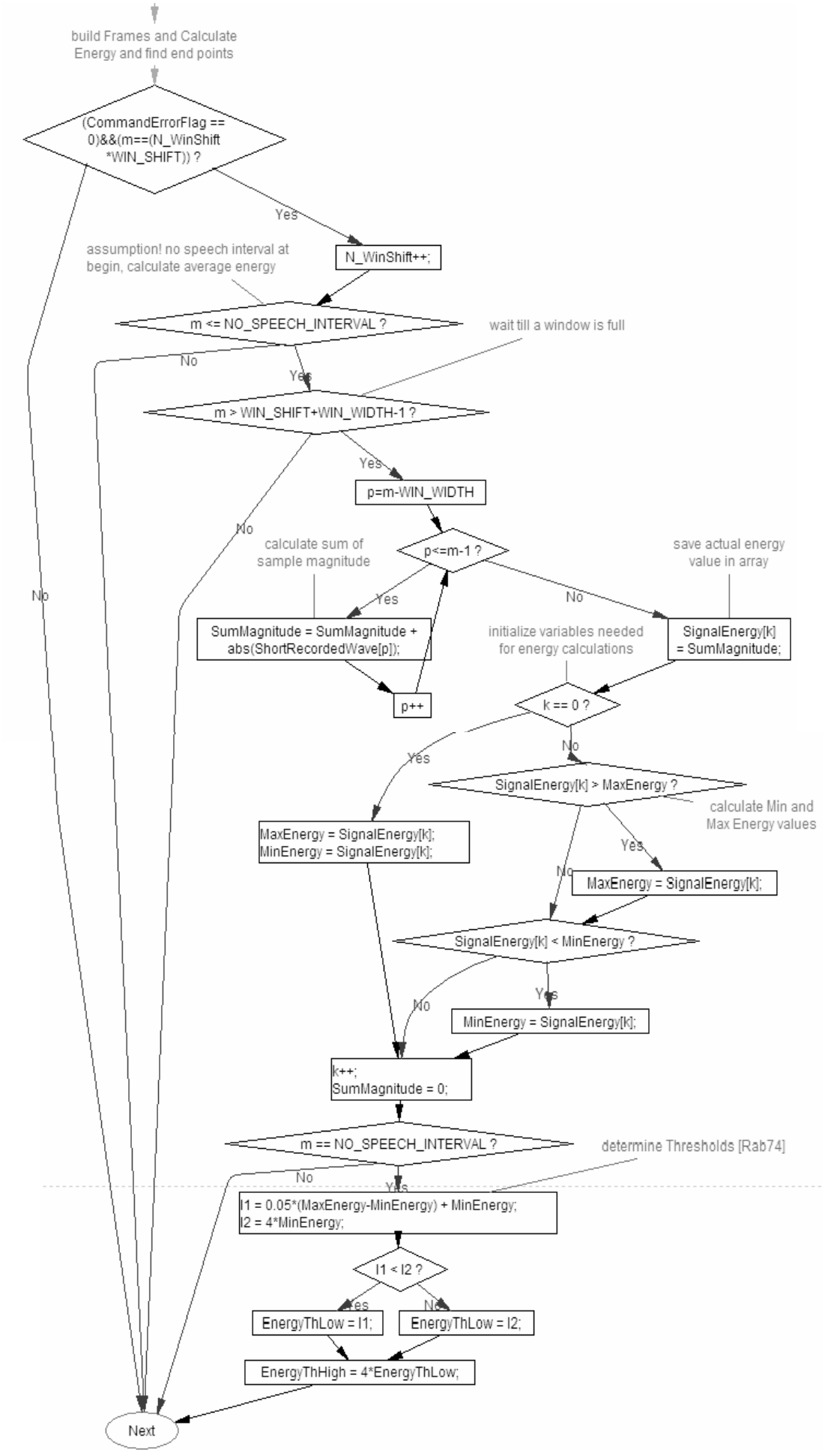
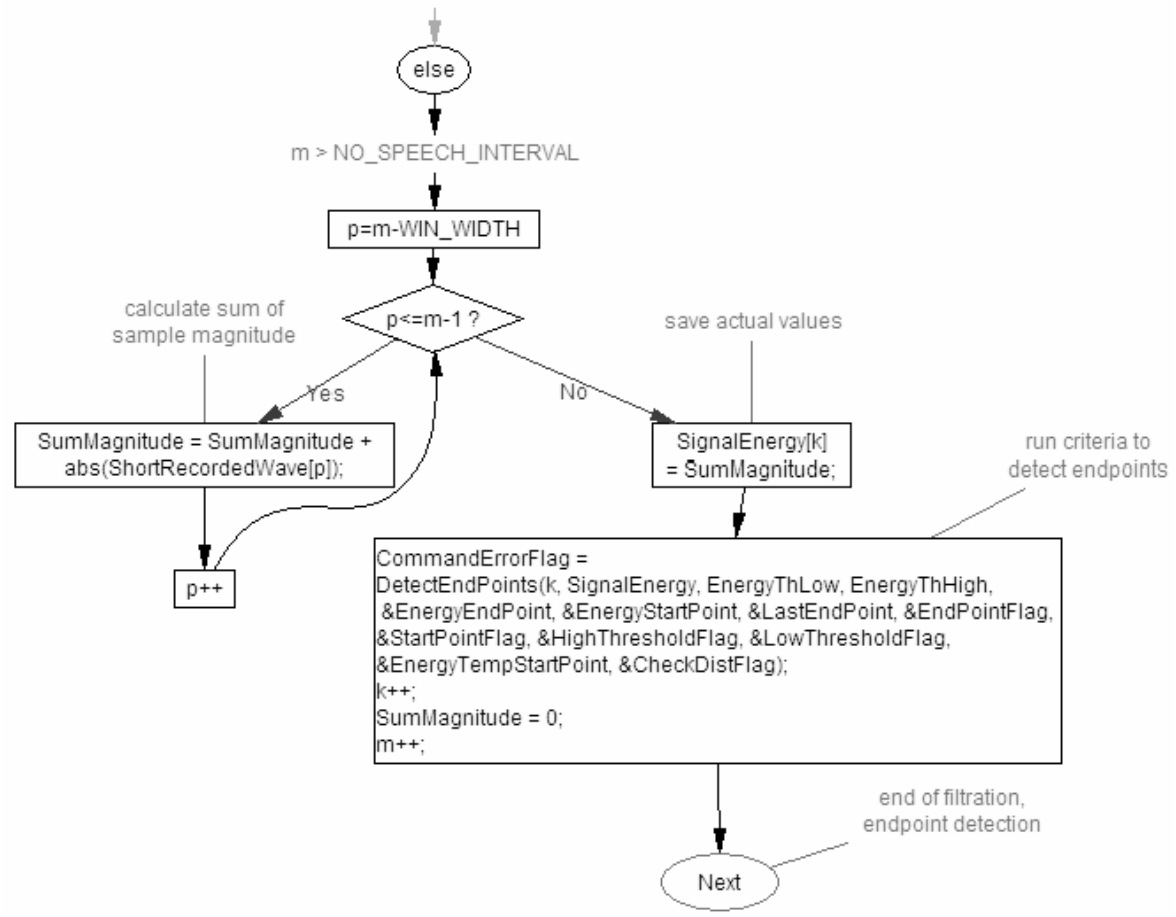
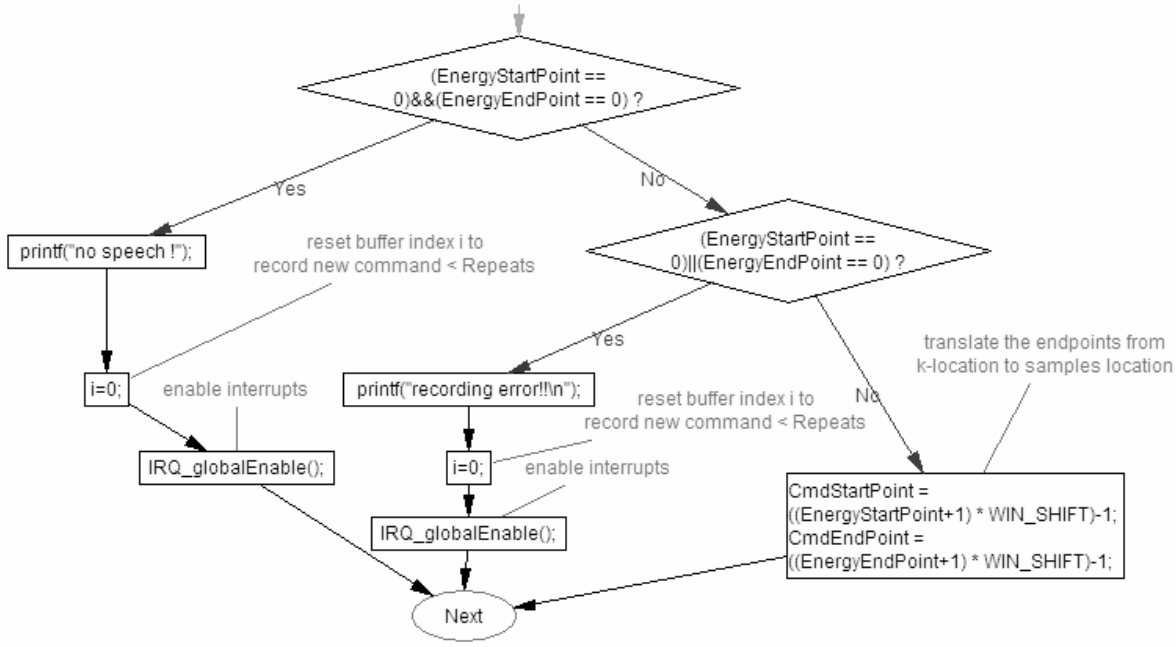


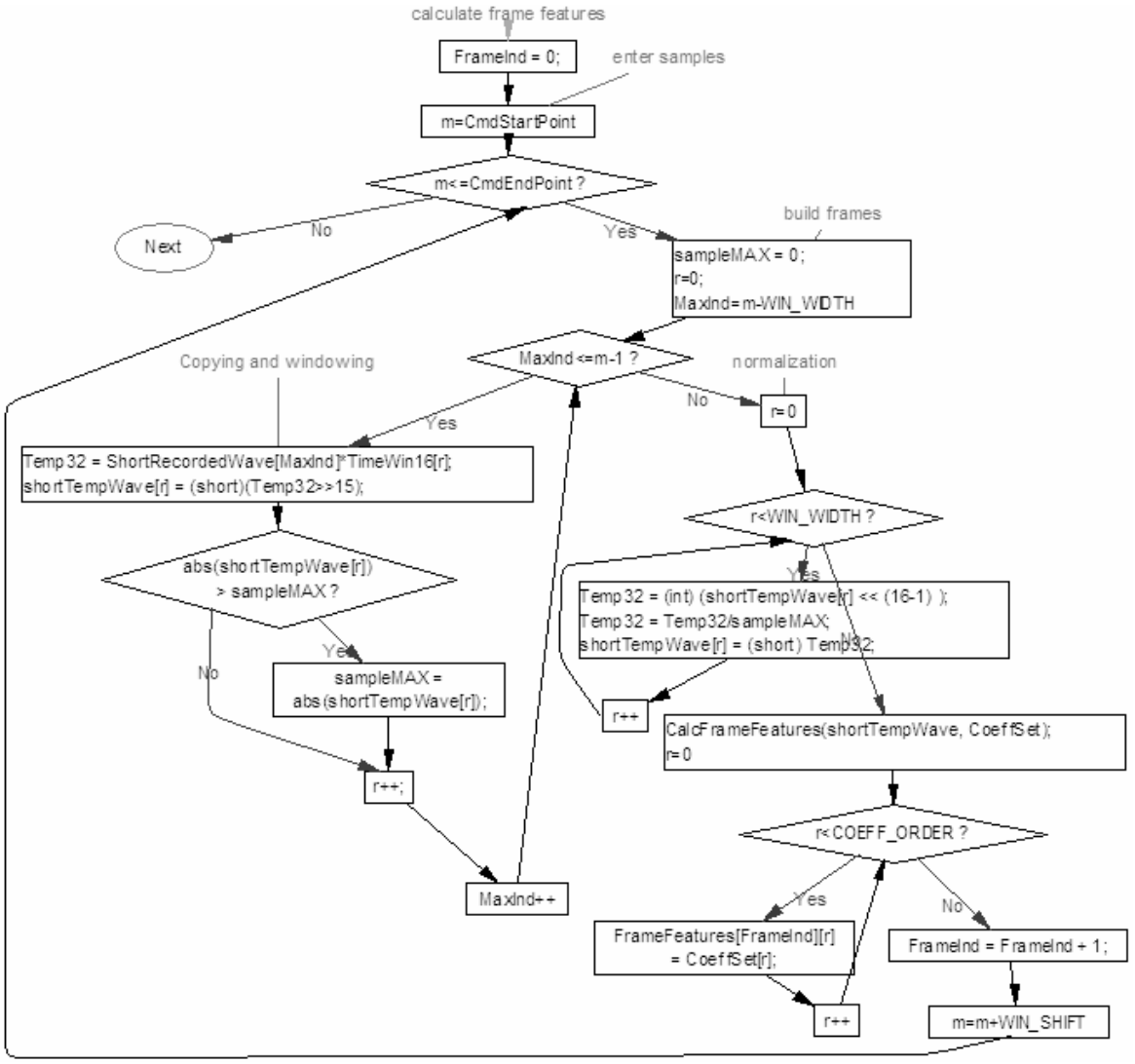
Abbildung 9-2 Flussdiagram vom Hauptprogramm

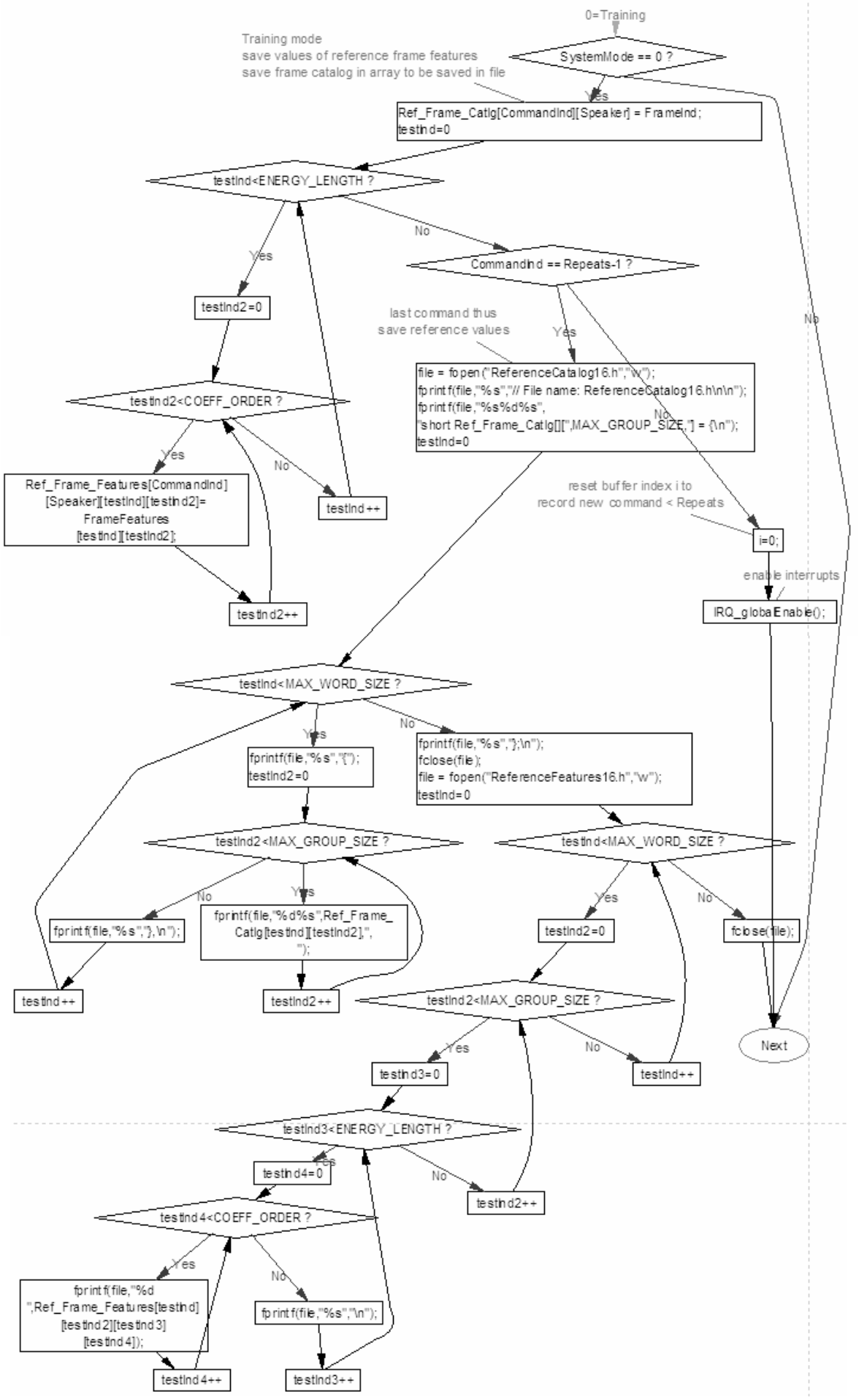

init DSK, codec, McBSP
comm_intr();
infinite loop
1 ?
No
End
Yes
No
BufferFlag == 1 ?
codec Buffer full
reset and wait to be filled in ISR
disable Interrupt
No
Yes
No
IRQ_globalDisable();
BufferFlag = 0;
CommandInd < Repeats ?
Yes
EnergyStartPoint = 0;
EnergyEndPoint = 0;
EnergyTempStartPoint = 0;
LastEndPoint = 0;
CommandErrorFlag = 0;
reset all flags needed for
endpoints detection
LowThresholdFlag = 0;
HighThresholdFlag = 0;
StartPointFlag = 0;
EndPointFlag = 0;
CheckDistFlag = 0;
k = 0;
enter samples for filtering
and energy calculation
m = 0;
N_WinShift = 1;
enter samples
SampleInd=0
SampleInd<NB_WAV_SAMPLES ?
filtering
Yes
FilteredWaveHP = FIR_HP(H_filt_FIR_HP, b_FIR_HP, N_delays_FIR_HP,
ShortRecordedWave[SampleInd]);
ShortRecordedWave[SampleInd] = FIR_LP(H_filt_FIR_LP,
b_FIR_LP, N_delays_FIR_LP, FilteredWaveHP);
Next
SampleInd++

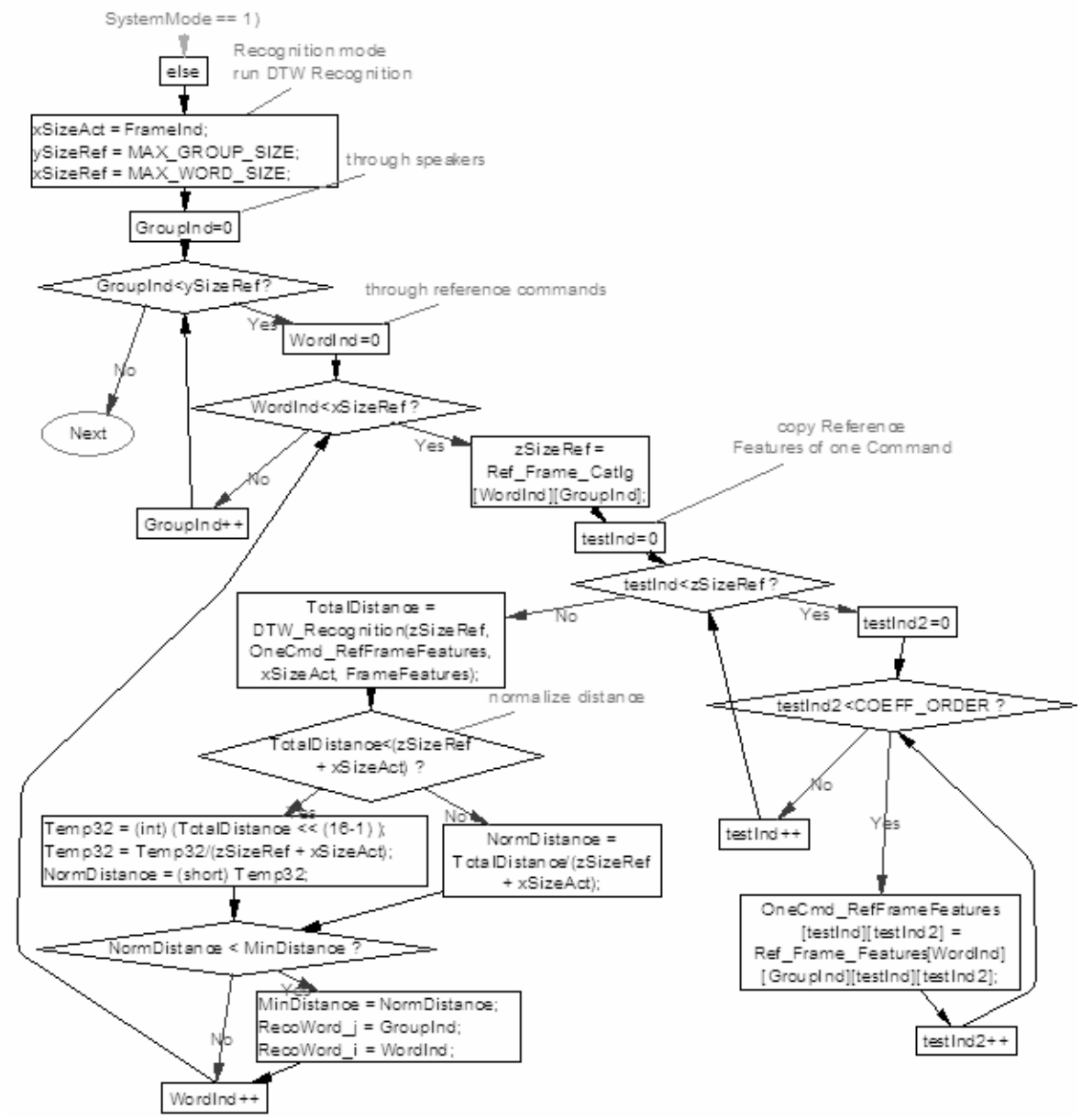

build Frames and Calculate Energy and find end points
(CommandErrorFlag == 0)&&(m==(N_WinShift *WIN_SHIFT)) ?
Yes
No
assumption! no speech interval at begin, calculate average energy
N_WinShift++;
m <= NO_SPEECH_INTERVAL ?
wait till a window is full
No
Yes
m > WIN_SHIFT+WIN_WIDTH-1 ?
Yes
No
p=m-WIN_WIDTH
calculate sum of sample magnitude
p<=m-1 ?
save actual energy value in array
Yes
No
SumMagnitude = SumMagnitude + abs(ShortRecordedWave[p]);
initialize variables needed for energy calculations
SignalEnergy[k] = SumMagnitude;
p++
k == 0 ?
No
Yes
SignalEnergy[k] > MaxEnergy ?
calculate Min and Max Energy values
MaxEnergy = SignalEnergy[k];
MinEnergy = SignalEnergy[k];
Yes
No
MaxEnergy = SignalEnergy[k];
SignalEnergy[k] < MinEnergy ?
No
Yes
MinEnergy = SignalEnergy[k];
k++;
SumMagnitude = 0;
m == NO_SPEECH_INTERVAL ?
determine Thresholds [Rab74]
No
Yes
I1 = 0.05*(MaxEnergy-MinEnergy) + MinEnergy;
I2 = 4*MinEnergy;
I1 < I2 ?
Yes
No
EnergyThLow = I1;
EnergyThLow = I2;
EnergyThHigh = 4*EnergyThLow;
Next

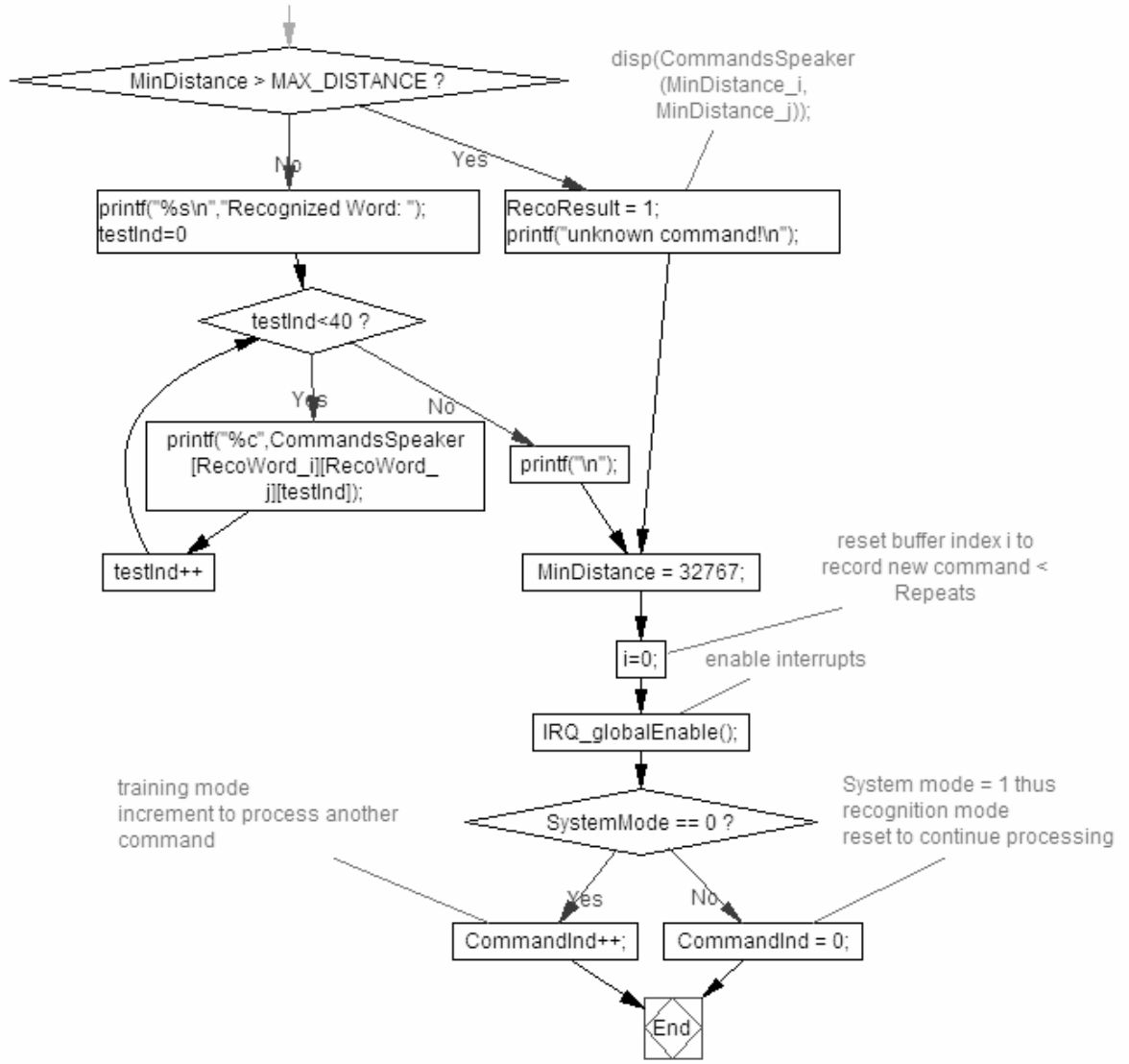

else
m > NO_SPEECH_INTERVAL
p=m-WIN_WIDTH
p<=m-1 ?
calculate sum of sample magnitude
save actual values
Yes
No
SumMagnitude = SumMagnitude + abs(ShortRecordedWave[p]);
SignalEnergy[k] = SumMagnitude;
run criteria to detect endpoints
p++
CommandErrorFlag = DetectEndPoints(k, SignalEnergy, EnergyThLow, EnergyThHigh, &EnergyEndPoint, &EnergyStartPoint, &LastEndPoint, &EndPointFlag, &StartPointFlag, &HighThresholdFlag, &LowThresholdFlag, &EnergyTempStartPoint, &CheckDistFlag);
k++;
SumMagnitude = 0;
m++;
end of filtration, endpoint detection
Next


(EnergyStartPoint == 0)&&(EnergyEndPoint == 0) ?
Yes
No
printf("no speech !");
reset buffer index i to record new command < Repeats
(EnergyStartPoint == 0)||(EnergyEndPoint == 0) ?
Yes
translate the endpoints from k-location to samples location
i=0;
enable interrupts
printf("recording error!!\n");
reset buffer index i to record new command < Repeats
No
IRQ_globalEnable();
i=0;
enable interrupts
CmdStartPoint = ((EnergyStartPoint+1) * WIN_SHIFT)-1;
CmdEndPoint = ((EnergyEndPoint+1) * WIN_SHIFT)-1;
IRQ_globalEnable();
Next


calculate frame features
FrameInd = 0;
enter samples
m=CmdStartPoint
m<=CmdEndPoint ?
No
Next
Yes
build frames
sampleMAX = 0;
r=0;
MaxInd=m-WIN_WIDTH
MaxInd<=m-1 ?
Copying and windowing
normalization
No
Yes
r=0
Temp32 = ShortRecordedWave[MaxInd]*TimeWin16[r];
shortTempWave[r] = (short)(Temp32>>15);
r<WIN_WIDTH ?
abs(shortTempWave[r])
> sampleMAX ?
Yes
Temp32 = (int) (shortTempWave[r] << (16-1));
Temp32 = Temp32/sampleMAX;
shortTempWave[r] = (short) Temp32;
No
Yes
sampleMAX =
abs(shortTempWave[r]);
r++
r++;
CalcFrameFeatures(shortTempWave, CoeffSet);
r=0
r<COEFF_ORDER ?
Yes
No
MaxInd++
FrameFeatures[FrameInd][r]
= CoeffSet[r];
FrameInd = FrameInd + 1;
r++
m=m+WIN_SHIFT


0=Training
SystemMode == 0 ?
Training mode
save values of reference frame features
save frame catalog in array to be saved in file
Yes
Ref_Frame_Catlg[CommandInd][Speaker] = FrameInd;
testInd=0
testInd<ENERGY_LENGTH ?
No
Yes
CommandInd == Repeats-1 ?
testInd2=0
last command thus
save reference values
Yes
No
file = fopen("ReferenceCatalog16.h","w");
fprintf(file,"%s","// File name: ReferenceCatalog16.h\n\n");
fprintf(file,"%s%d%s",
"short Ref_Frame_Catlg[][",MAX_GROUP_SIZE,"] = {\n");
testInd=0
testInd2<COEFF_ORDER ?
Yes
No
Ref_Frame_Features[CommandInd]
[Speaker][testInd][testInd2]=
FrameFeatures
[testInd][testInd2];
testInd++
reset buffer index i to
record new command < Repeats
i=0;
enable interrupts
IRQ_globalEnable();
testInd2++
testInd<MAX_WORD_SIZE ?
Yes
No
fprintf(file,"%s","{");
testInd2=0
fprintf(file,"%s","};\n");
fclose(file);
file = fopen("ReferenceFeatures16.h","w");
testInd=0
testInd2<MAX_GROUP_SIZE ?
testInd<MAX_WORD_SIZE ?
No
Yes
fprintf(file,"%s","},\n");
fprintf(file,"%d%s",Ref_Frame_
Catlg[testInd][testInd2],",
");
Yes
No
testInd2=0
fclose(file);
testInd++
testInd2++
testInd2<MAX_GROUP_SIZE ?
Next
Yes
No
testInd3=0
testInd++
testInd3<ENERGY_LENGTH ?
Yes
No
testInd4=0
testInd2++
testInd4<COEFF_ORDER ?
Yes
No
fprintf(file,"%d
",Ref_Frame_Features[testInd]
[testInd2][testInd3]
[testInd4]);
fprintf(file,"%s","\n");
testInd4++
testInd3++


SystemMode == 1)
else
Recognition mode
run DTW Recognition
xSizeAct = FrameInd;
ySizeRef = MAX_GROUP_SIZE;
xSizeRef = MAX_WORD_SIZE;
through speakers
GroupInd=0
GroupInd<ySizeRef ?
Yes
No
through reference commands
WordInd=0
WordInd<xSizeRef ?
Yes
No
Next
GroupInd++
zSizeRef =
Ref_Frame_Catlg
[WordInd][GroupInd];
copy Reference
Features of one Command
testInd=0
testInd<zSizeRef ?
No
Yes
TotalDistance =
DTW_Recognition(zSizeRef,
OneCmd_RefFrameFeatures,
xSizeAct, FrameFeatures);
testInd2=0
testInd2<COEFF_ORDER ?
No
Yes
testInd++
normalize distance
TotalDistance<(zSizeRef
+ xSizeAct) ?
Yes
No
Temp32 = (int) (TotalDistance << (16-1));
Temp32 = Temp32/(zSizeRef + xSizeAct);
NormDistance = (short) Temp32;
NormDistance =
TotalDistance/(zSizeRef
+ xSizeAct);
OneCmd_RefFrameFeatures
[testInd][testInd2] =
Ref_Frame_Features[WordInd]
[GroupInd][testInd][testInd2];
testInd2++
NormDistance < MinDistance ?
Yes
No
MinDistance = NormDistance;
RecoWord_j = GroupInd;
RecoWord_i = WordInd;
WordInd++


MinDistance > MAX_DISTANCE ?
disp(CommandsSpeaker (MinDistance_i, MinDistance_j));
No
Yes
printf("%s\n","Recognized Word: ");
testInd=0
RecoResult = 1;
printf("unknown command!\n");
testInd<40 ?
Yes
No
printf("%c",CommandsSpeaker [RecoWord_i][RecoWord_ j][testInd]);
printf("\n");
testInd++
MinDistance = 32767;
reset buffer index i to record new command < Repeats
i=0;
enable interrupts
IRQ_globalEnable();
training mode increment to process another command
SystemMode == 0 ?
System mode = 1 thus recognition mode reset to continue processing
Yes
No
CommandInd++;
CommandInd = 0;
End

9.1.3 FIR_HP()

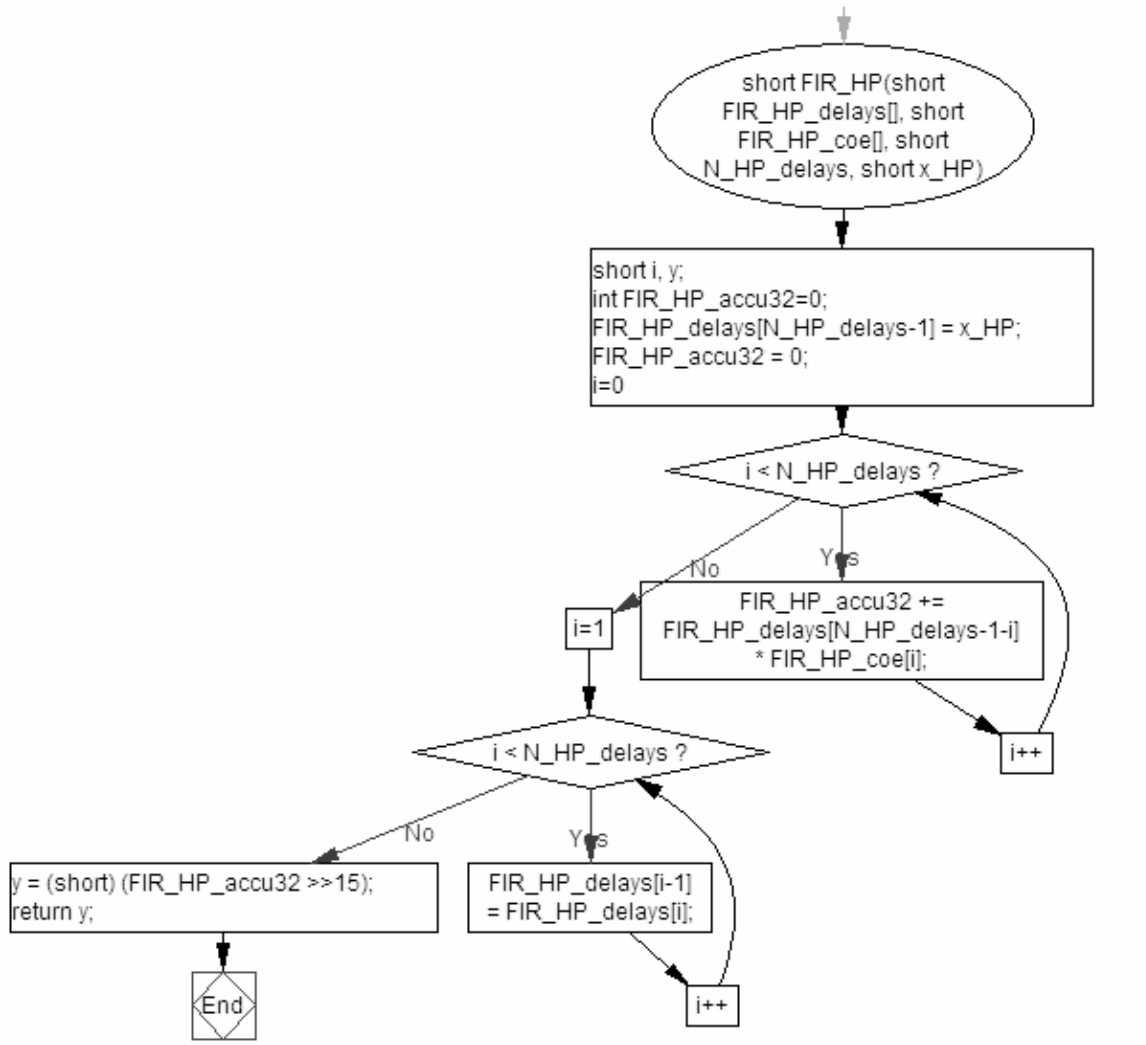


Abbildung 9-3 Flussdiagramm FIR_HP()

9.1.4 DetectEndPoints()

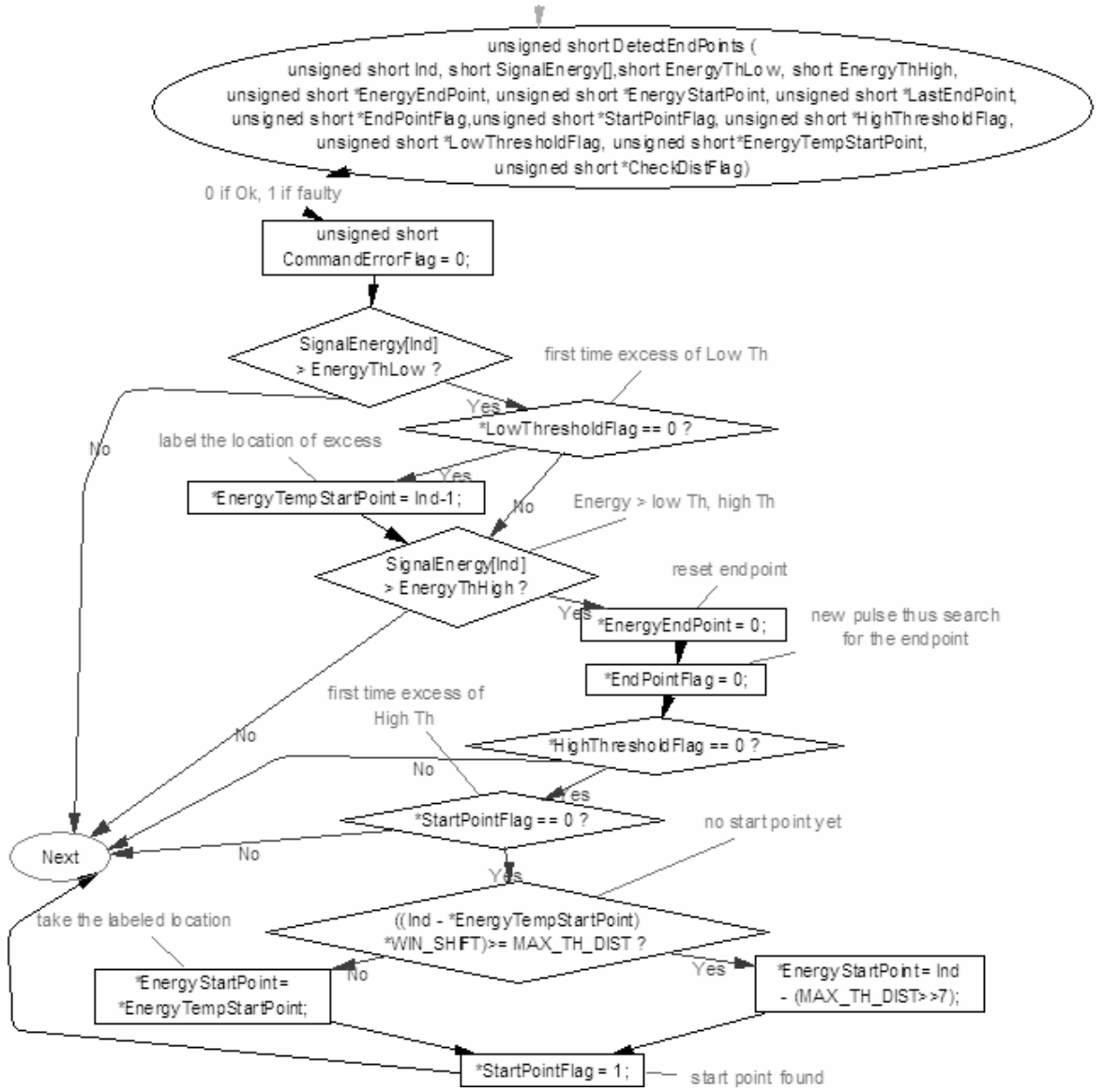
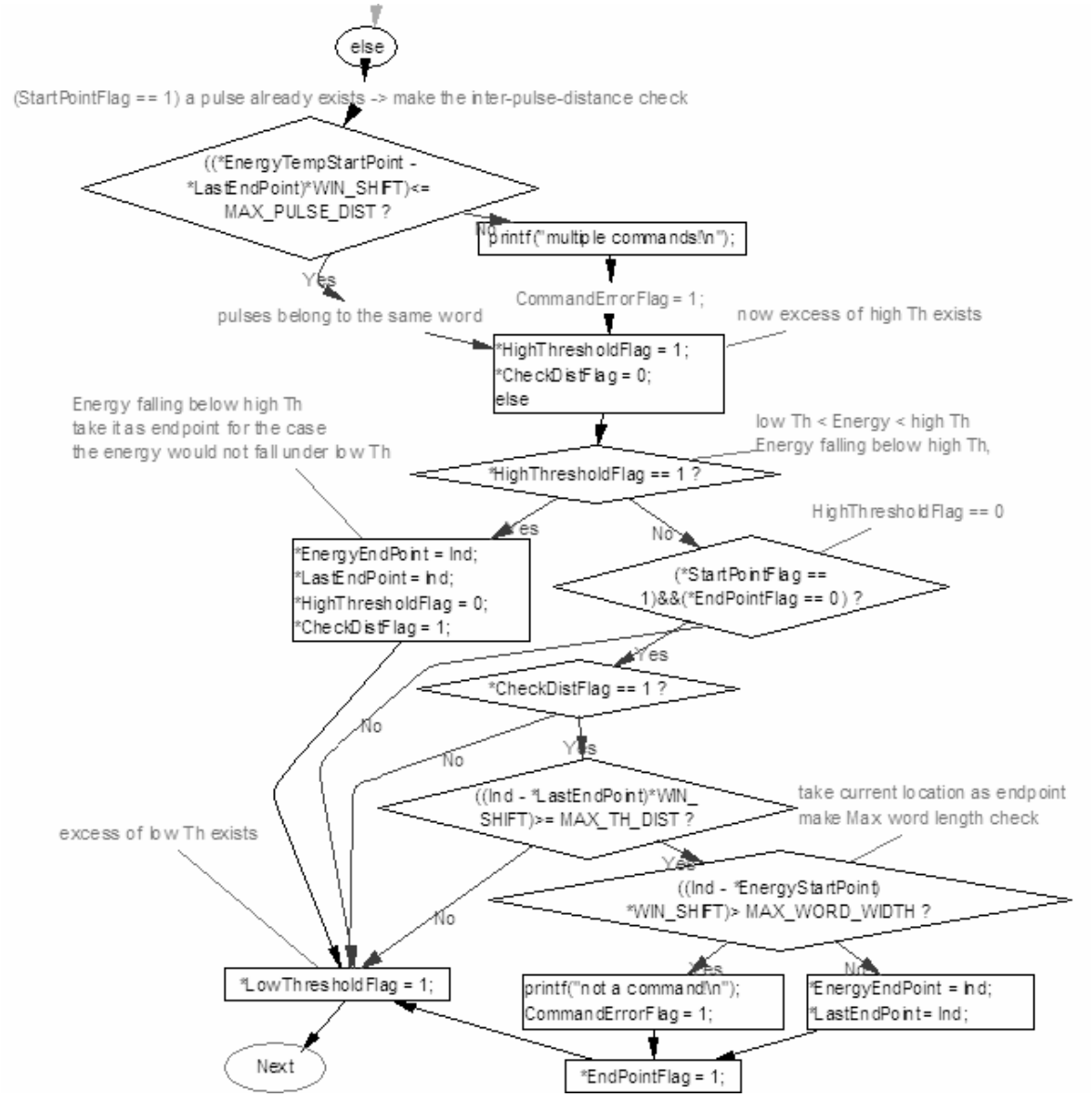
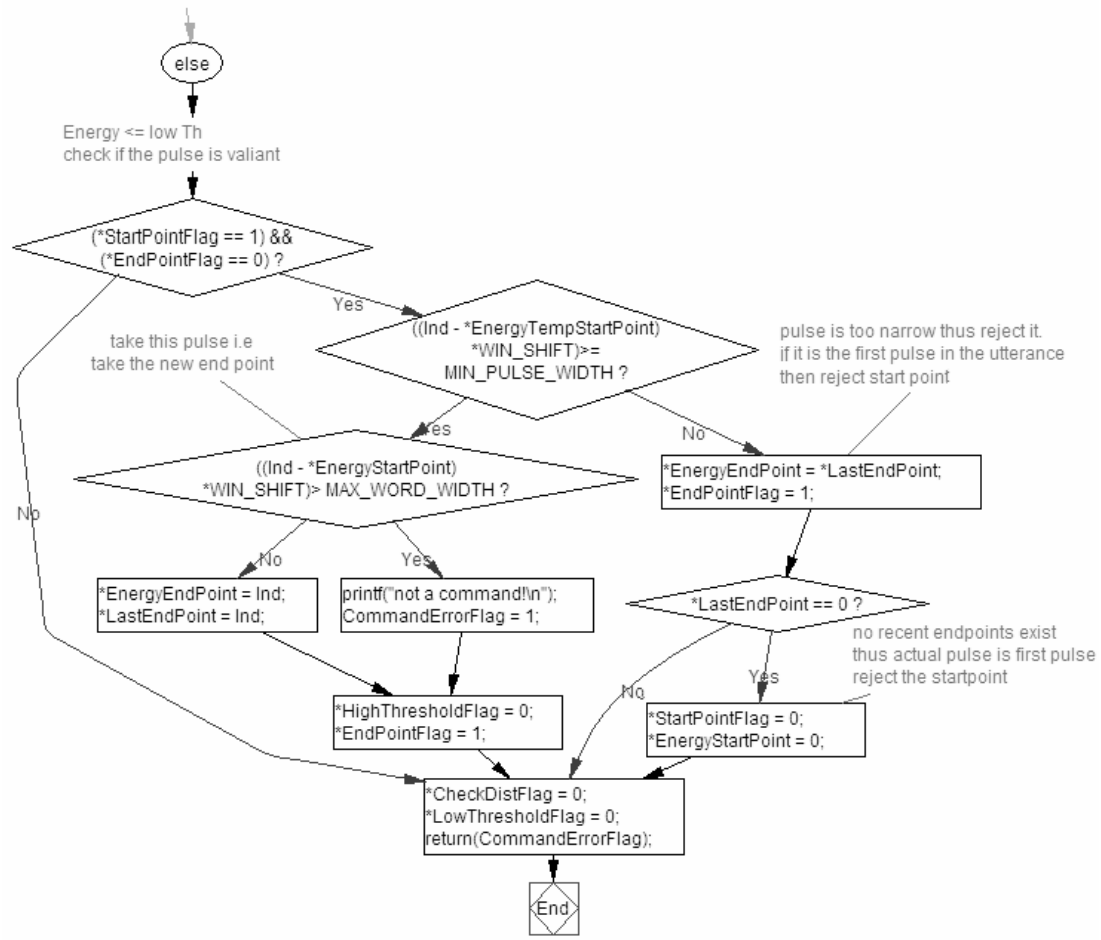


Abbildung 9-4 Flussdiagramm DetectEndPoints()


else
(StartPointFlag == 1) a pulse already exists -> make the inter-pulse-distance check
((*EnergyTempStartPoint - *LastEndPoint)*WIN_SHIFT)<= MAX_PULSE_DIST ?
No
printf("multiple commands!\n");
Yes
CommandErrorFlag = 1;
pulses belong to the same word
now excess of high Th exists
*HighThresholdFlag = 1;
*CheckDistFlag = 0;
else
Energy falling below high Th
take it as endpoint for the case
the energy would not fall under low Th
low Th < Energy < high Th
Energy falling below high Th,
*HighThresholdFlag == 1 ?
Yes
No
HighThresholdFlag == 0
*EnergyEndPoint = Ind;
*LastEndPoint = Ind;
*HighThresholdFlag = 0;
*CheckDistFlag = 1;
(*StartPointFlag == 1)&&(*EndPointFlag == 0) ?
Yes
*CheckDistFlag == 1 ?
No
Yes
No
((Ind - *LastEndPoint)*WIN_SHIFT)>= MAX_TH_DIST ?
take current location as endpoint
make Max word length check
excess of low Th exists
Yes
((Ind - *EnergyStartPoint) *WIN_SHIFT)> MAX_WORD_WIDTH ?
No
Yes
No
*LowThresholdFlag = 1;
printf("not a command!\n");
CommandErrorFlag = 1;
*EnergyEndPoint = Ind;
*LastEndPoint= Ind;
Next
*EndPointFlag = 1;


else
Energy <= low Th
check if the pulse is valiant
(*StartPointFlag == 1) &&
(*EndPointFlag == 0) ?
Yes
No
((Ind - *EnergyTempStartPoint)
*WIN_SHIFT)>=
MIN_PULSE_WIDTH ?
take this pulse i.e
take the new end point
pulse is too narrow thus reject it.
if it is the first pulse in the utterance
then reject start point
Yes
No
((Ind - *EnergyStartPoint)
*WIN_SHIFT)> MAX_WORD_WIDTH ?
*EnergyEndPoint = *LastEndPoint;
*EndPointFlag = 1;
No
Yes
*EnergyEndPoint = Ind;
*LastEndPoint = Ind;
printf("not a command!\n");
CommandErrorFlag = 1;
*LastEndPoint == 0 ?
no recent endpoints exist
thus actual pulse is first pulse
reject the startpoint
Yes
No
*HighThresholdFlag = 0;
*EndPointFlag = 1;
*StartPointFlag = 0;
*EnergyStartPoint = 0;
*CheckDistFlag = 0;
*LowThresholdFlag = 0;
return(CommandErrorFlag);
End

9.1.5 CalcFrameFeatures()

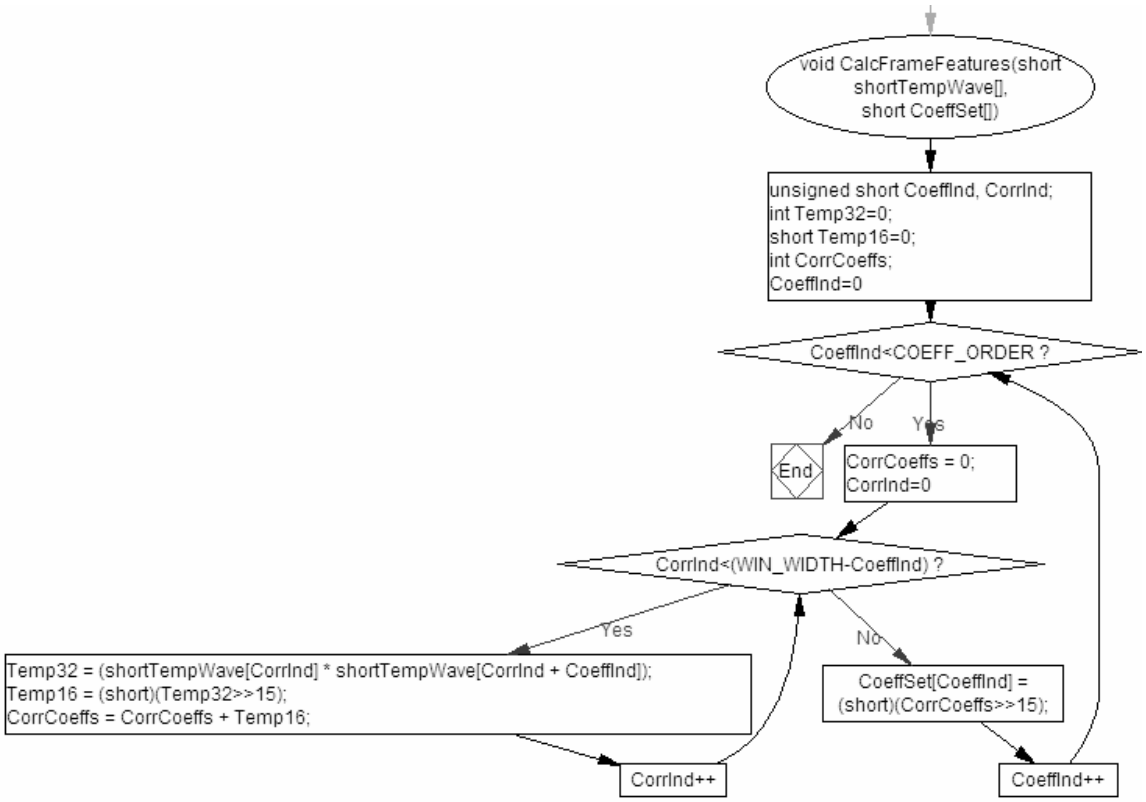


Abbildung 9-5 Flussdiagramm CalcFrameFeatures()

9.1.6 DTW_Recognition()

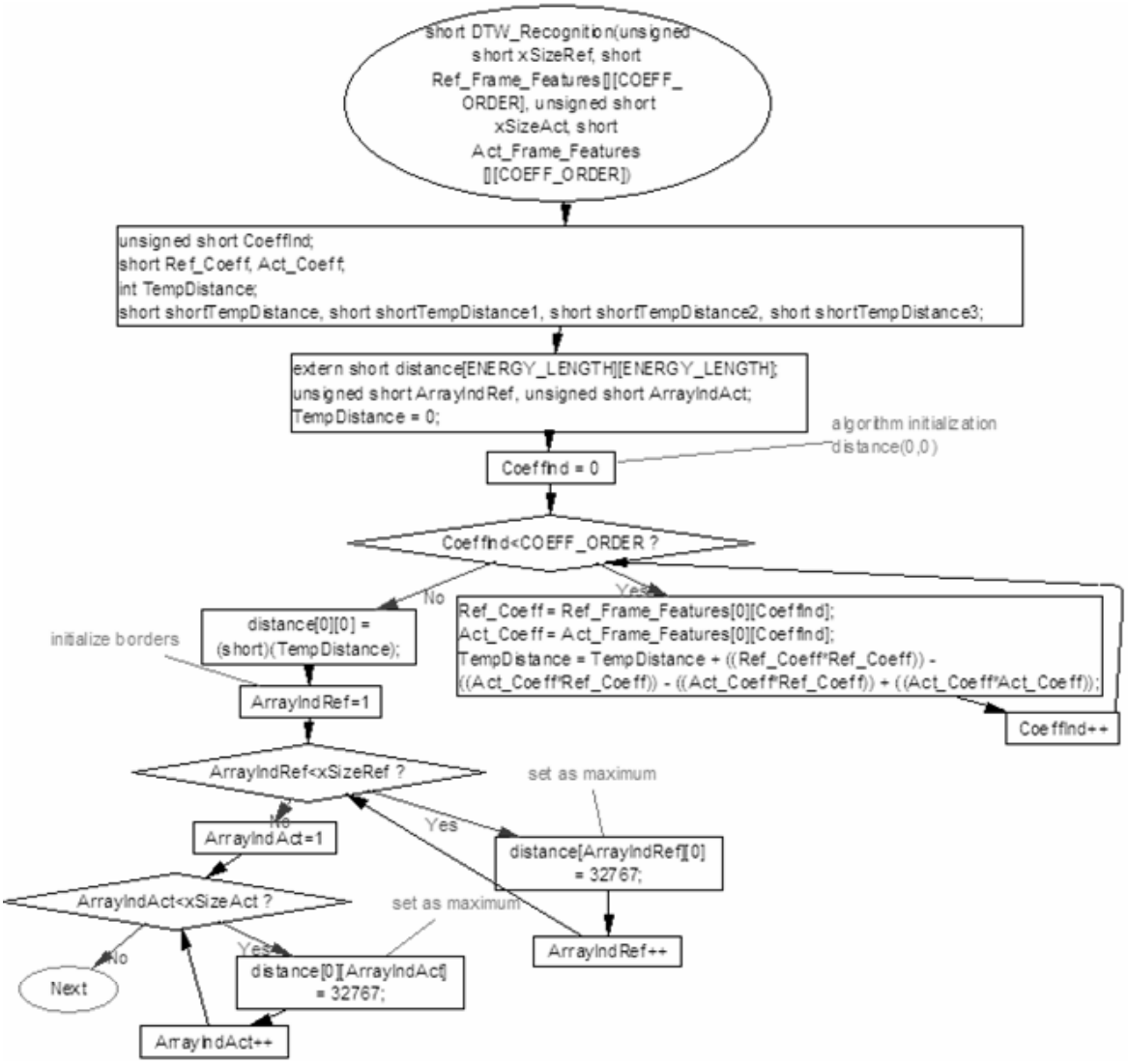
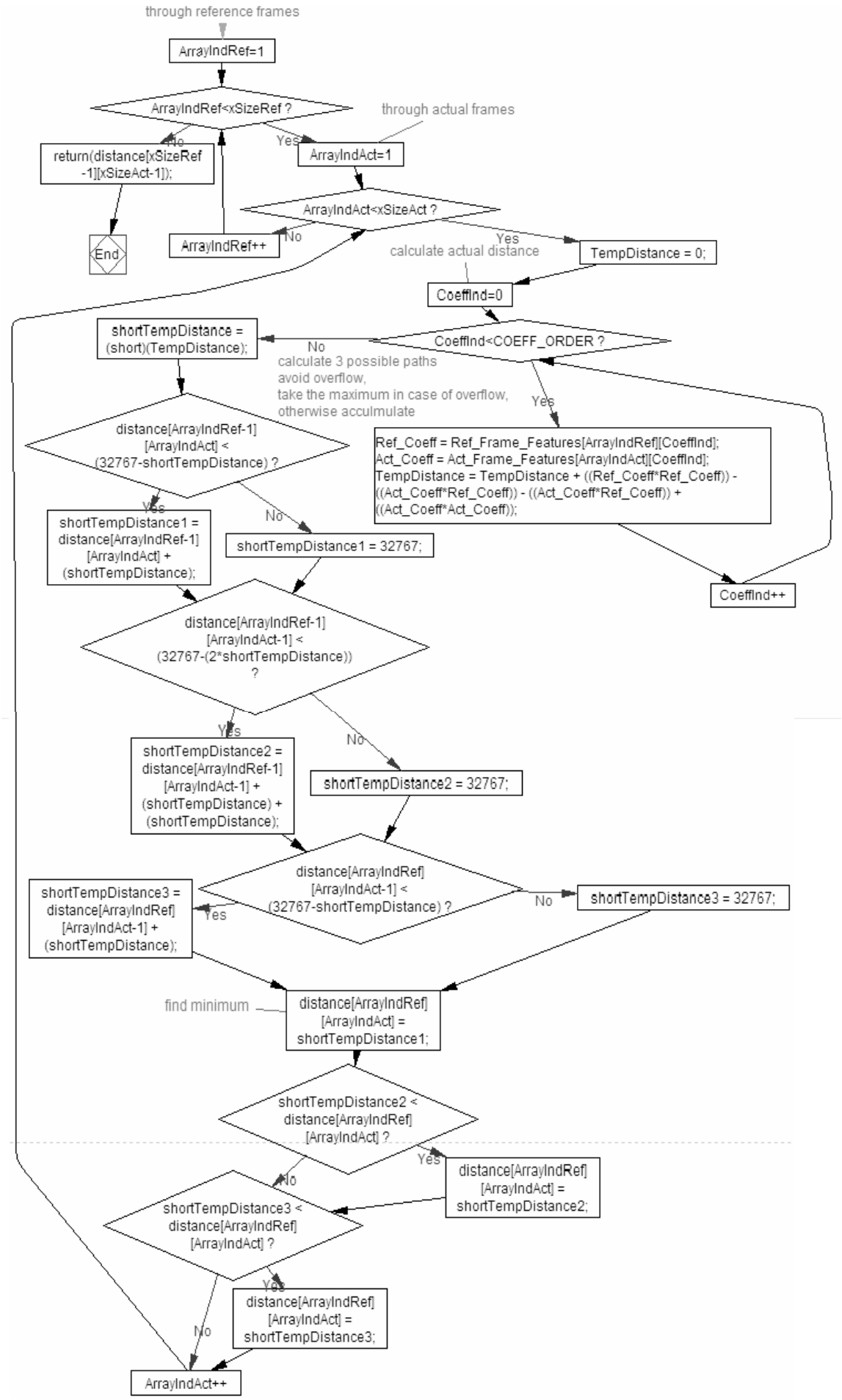


Abbildung 9-6 Flussdiagramm DTW_Recognition()


through reference frames
ArrayIndRef=1
ArrayIndRef<xSizeRef ?
No
Yes
return(distance[xSizeRef -1][xSizeAct-1]);
through actual frames
ArrayIndAct=1
ArrayIndAct<xSizeAct ?
No
Yes
End
ArrayIndRef++
calculate actual distance
TempDistance = 0;
CoeffInd=0
shortTempDistance = (short)(TempDistance);
CoeffInd<COEFF_ORDER ?
No
calculate 3 possible paths
avoid overflow,
take the maximum in case of overflow,
otherwise acculmulate
Yes
distance[ArrayIndRef-1] [ArrayIndAct] < (32767-shortTempDistance) ?
Ref_Coeff = Ref_Frame_Features[ArrayIndRef][CoeffInd];
Act_Coeff = Act_Frame_Features[ArrayIndAct][CoeffInd];
TempDistance = TempDistance + ((Ref_Coeff*Ref_Coeff)) - ((Act_Coeff*Ref_Coeff)) - ((Act_Coeff*Ref_Coeff)) + ((Act_Coeff*Act_Coeff));
Yes
No
shortTempDistance1 = distance[ArrayIndRef-1] [ArrayIndAct] + (shortTempDistance);
shortTempDistance1 = 32767;
CoeffInd++
distance[ArrayIndRef-1] [ArrayIndAct-1] < (32767-(2*shortTempDistance)) ?
Yes
No
shortTempDistance2 = distance[ArrayIndRef-1] [ArrayIndAct-1] + (shortTempDistance) + (shortTempDistance);
shortTempDistance2 = 32767;
distance[ArrayIndRef] [ArrayIndAct-1] < (32767-shortTempDistance) ?
shortTempDistance3 = distance[ArrayIndRef] [ArrayIndAct-1] + (shortTempDistance);
Yes
No
shortTempDistance3 = 32767;
find minimum
distance[ArrayIndRef] [ArrayIndAct] = shortTempDistance1;
shortTempDistance2 < distance[ArrayIndRef] [ArrayIndAct] ?
Yes
No
distance[ArrayIndRef] [ArrayIndAct] = shortTempDistance2;
shortTempDistance3 < distance[ArrayIndRef] [ArrayIndAct] ?
Yes
No
distance[ArrayIndRef] [ArrayIndAct] = shortTempDistance3;
ArrayIndAct++

9.2 C-Code

9.2.1 SpeechRecognition.c

```
//    C-File name             |     Autor             |       Last Update
// -------------------------|-----------------------|-------------------
//   SpeechRecognition.c     |    Ziad Ghadieh       |       01.03.2008
//
//
//
// Description:
// ------------
// this file contains the commands in text form
// originaly created in MATLAB using SaveCommandSpeaker16.m
//
// Input parameters:
// --------------------
// none
//
// Output parameters:
// --------------------
// none
// ------------------------------------------------------------------------

#include "SpeechRecognition.h"
#include "Prefilter16.h"
#include "CommandSpeaker16.h"
#include "Debug\ReferenceCatalog16.h"
#include "TimeWin16.h"
#include "dsk6713_aic23.h"               //codec-DSK support file

#pragma DATA_SECTION(Ref_Frame_Features, ".EXT_RAM")
short
Ref_Frame_Features[MAX_WORD_SIZE][MAX_GROUP_SIZE][ENERGY_LENGTH][COEFF_OR
DER];
#pragma DATA_SECTION(ShortRecordedWave, ".EXT_RAM")
short ShortRecordedWave[NB_WAV_SAMPLES];
#pragma DATA_SECTION(distance, ".EXT_RAM")
short distance[ENERGY_LENGTH][ENERGY_LENGTH];
#pragma DATA_SECTION(FrameFeatures, ".EXT_RAM")
short FrameFeatures[ENERGY_LENGTH][COEFF_ORDER];
#pragma DATA_SECTION(SignalEnergy, ".EXT_RAM")
short SignalEnergy[ENERGY_LENGTH];
#pragma DATA_SECTION(shortTempWave, ".EXT_RAM")
short shortTempWave[WIN_WIDTH];
#pragma DATA_SECTION(OneCmd_RefFrameFeatures, ".EXT_RAM")
short OneCmd_RefFrameFeatures[ENERGY_LENGTH][COEFF_ORDER];

unsigned short testInd;
unsigned short testInd2;
unsigned short testInd3;
unsigned short testInd4;
unsigned short SampleInd;
short FilteredWaveHP;
unsigned short EnergyStartPoint;
unsigned short EnergyEndPoint;
unsigned short LastEndPoint;
unsigned short EnergyTempStartPoint;
unsigned short CommandErrorFlag;
```

```
short SumMagnitude = 0;
short MaxEnergy;
short MinEnergy;
short I1;
short I2;
short EnergyThHigh;
short EnergyThLow;

// flags needed for endpoints detection
unsigned short LowThresholdFlag;
unsigned short HighThresholdFlag;
unsigned short StartPointFlag;
unsigned short EndPointFlag;
unsigned short CheckDistFlag;
unsigned short CmdStartPoint;
unsigned short CmdEndPoint;
unsigned short k; //frame index
unsigned short m; //sample index for energy calculation
unsigned short p; //sample index within a frame
unsigned short FrameInd; // Frame index
unsigned short xSizeAct;
unsigned short RecoWord_i;
unsigned short RecoWord_j;
unsigned short RecoResult=0;

FILE *file;
short sampleMAX;
unsigned short r, MaxInd;
int Temp32=0;
short CoeffSet[COEFF_ORDER];
unsigned short GroupInd;
unsigned short WordInd;
unsigned short zSizeRef;
unsigned short ySizeRef;
unsigned short xSizeRef;
short MinDistance = 32767; //~infinity
short NormDistance;
short TotalDistance;
short N_WinShift;
short do_read_Features;
short CommandInd = 0;
short SystemMode = 1; // 1 for Recognition, 0 for Training
short Repeats = 1;
short Speaker;

Uint32 fs=DSK6713_AIC23_FREQ_8 KHZ;//set sampling rate
short input, output;
//short bufferlength = 16000;  //buffer size 2s
short i = 0;
short BufferFlag = 0;

interrupt void c_int11()        //ISR
{
      input = input_sample();

      output_sample(input); //for test

      if(i < NB_WAV_SAMPLES)
      {
            DSK6713_LED_off(0);
            DSK6713_LED_off(1);
```

```
            DSK6713_LED_off(2);
            DSK6713_LED_off(3);
            ShortRecordedWave[i] = input;

            if(i == NB_WAV_SAMPLES - 1)
            {
                  DSK6713_LED_on(0);
                  DSK6713_LED_on(1);
                  DSK6713_LED_on(2);
                  DSK6713_LED_on(3);
                  BufferFlag = 1; //buffer is full
            }
            i++;
      }
}


main()
{
      int ok=1, indx=0;
      short d1=0, d2=0, d3=0, d4=0, d5=0, d6=0, d7=0, d8=0, d9=0, d10=0;
      short *p2Ref_Frame_Features = NULL;
      int cnt = 0;

      // clear FIR delays
      for (i=0;i<N_delays_FIR_HP;i++)
            H_filt_FIR_HP[i] = 0;
      for (i=0;i<N_delays_FIR_LP;i++)
            H_filt_FIR_LP[i] = 0;

      // ------------- enter here SystemMode -------------
      SystemMode = 1; // 0 for Training, 1 for Recognition
      //enter hier which speaker
      Speaker = 0; // 0 Default
      // -------------------------------------------

      DSK6713_LED_on(0);
      DSK6713_LED_on(1);
      DSK6713_LED_on(2);
      DSK6713_LED_on(3);

      if (SystemMode == 1)
      {
            scanf("%hd",&do_read_Features);
            if (do_read_Features == 1)
            {
                  // read ReferenceFeature_Vector 4D-matrix:
                  file = fopen("ReferenceFeatures16.h","r");
                  indx=0;
                  cnt = 0;
                  ok=1;
                  while (ok)
                  {
                        p2Ref_Frame_Features =
                        &Ref_Frame_Features[0][0][0][0];
                        ok = fscanf(file, "%hd %hd %hd %hd %hd %hd %hd
                              %hd %hd %hd ",
                              &d1, &d2,  &d3, &d4, &d5, &d6, &d7,  &d8,
                              &d9, &d10);

                        p2Ref_Frame_Features[indx++] = d1;
```

```
                        p2Ref_Frame_Features[indx++] = d2;
                        p2Ref_Frame_Features[indx++] = d3;
                        p2Ref_Frame_Features[indx++] = d4;
                        p2Ref_Frame_Features[indx++] = d5;
                        p2Ref_Frame_Features[indx++] = d6;
                        p2Ref_Frame_Features[indx++] = d7;
                        p2Ref_Frame_Features[indx++] = d8;
                        p2Ref_Frame_Features[indx++] = d9;
                        p2Ref_Frame_Features[indx++] = d10;

                        cnt++;
                        if (cnt >= 1500)
                              break;
                  }
                  fclose(file);
            }

            // recognition mode thus 1 Time
            Repeats = 1;
            CommandInd = 0; //initialize
      }
      else //(SystemMode == 0)
      {
            // training mode thus repeat the process for all commands
            Repeats = MAX_WORD_SIZE;
      }


      comm_intr();      //init DSK, codec, McBSP
      while(1)       //infinite loop
      {

            if(BufferFlag == 1)
            {
                  // codec Buffer full
                  // reset and wait to be filled in ISR

                  // set a break point hier for training!!!
                  // -------------------------------------
                  IRQ_globalDisable(); //disable Interrupt
                  BufferFlag = 0;

                  if(CommandInd < Repeats)
                  {

                        EnergyStartPoint = 0;
                        EnergyEndPoint = 0;
                        EnergyTempStartPoint = 0;
                        LastEndPoint = 0;
                        CommandErrorFlag = 0;

                        //reset all flags needed for endpoints detection
                        LowThresholdFlag = 0;
                        HighThresholdFlag = 0;
                        StartPointFlag = 0;
                        EndPointFlag = 0;
                        CheckDistFlag = 0;

                        k = 0;
```

```
                    //enter samples for filtering and energy
                    calculation
                    m = 0;
                    N_WinShift = 1;

for (SampleInd=0; SampleInd<NB_WAV_SAMPLES;
      SampleInd++) //enter samples
{
      // filtering
      FilteredWaveHP = FIR_HP(H_filt_FIR_HP, b_FIR_HP,
                  N_delays_FIR_HP, ShortRecordedWave[SampleInd]);
      ShortRecordedWave[SampleInd] = FIR_LP(H_filt_FIR_LP,
                  b_FIR_LP, N_delays_FIR_LP, FilteredWaveHP);

      // build Frames and Calculate Energy and
      find end points
      if((CommandErrorFlag == 0)&&(m==(N_WinShift*WIN_SHIFT)))
      {
            N_WinShift++;

            // assumption! no speech interval at
            // begin, calculate average energy
            if (m <= NO_SPEECH_INTERVAL)
            {
                  // wait till a window is full
                  if (m > WIN_SHIFT+WIN_WIDTH-1)
                  {
                        for (p=m-WIN_WIDTH;p<=m-1; p++)
                        {
                              //calculate sum of sample magnitude
                              SumMagnitude = SumMagnitude +
                              abs(ShortRecordedWave[p]);
                        }
                        //save actual energy value in array
                        SignalEnergy[k] = SumMagnitude;

                        if (k == 0) //initialize variables needed
                                    for energy calculations
                        {
                              MaxEnergy = SignalEnergy[k];
                              MinEnergy = SignalEnergy[k];
                        }
                        else
                        {
                              // calculate Min and Max Energy
                              values
                              if (SignalEnergy[k] > MaxEnergy)
                                    MaxEnergy = SignalEnergy[k];
                              if (SignalEnergy[k] < MinEnergy)
                                    MinEnergy = SignalEnergy[k];
                        }

                        k++;
                        SumMagnitude = 0;

                        if (m == NO_SPEECH_INTERVAL)
                        {
                        // determine Thresholds [Rab74]

                              I1 = 0.05*(MaxEnergy-MinEnergy) +
                                    MinEnergy;
```

```
                                    I2 = 4*MinEnergy;

                                    if (I1 < I2)
                                          EnergyThLow = I1;
                                    else
                                          EnergyThLow = I2;
                                    EnergyThHigh = 4*EnergyThLow;
                               }
                        }
                  }
                  else
                  {
                        //m > NO_SPEECH_INTERVAL
                        for (p=m-WIN_WIDTH; p<=m-1; p++)
                        {
                              //calculate sum of sample magnitude
                              SumMagnitude = SumMagnitude +
                                    abs(ShortRecordedWave[p]);
                        }
                        //save actual values
                        SignalEnergy[k] = SumMagnitude;

                        //run criteria to detect endpoints
                        CommandErrorFlag = DetectEndPoints(k,
                        SignalEnergy, EnergyThLow, EnergyThHigh,
                        &EnergyEndPoint, &EnergyStartPoint,
                        &LastEndPoint, &EndPointFlag, &StartPointFlag,
                        &HighThresholdFlag, &LowThresholdFlag,
                        &EnergyTempStartPoint, &CheckDistFlag);

                        k++;
                        SumMagnitude = 0;

                  }


            }
            m++;
            //end of filtration, endpoint detection

      }

      if ((EnergyStartPoint == 0)&&(EnergyEndPoint == 0))
      {
            printf("no speech !");
            // reset buffer index i to record new command < Repeats
            i=0;
            IRQ_globalEnable(); //enable interrupts
      }
      else
      {
            if ((EnergyStartPoint == 0)||(EnergyEndPoint == 0))
            {
                  printf("recording error!!\n");
                  // reset buffer index i to record new command < Repeats
                  i=0;
                  IRQ_globalEnable(); //enable interrupts

            }
            else
            {
```

```
                  // translate the endpoints from k-location to samples
                  location
                  CmdStartPoint = ((EnergyStartPoint+1) * WIN_SHIFT)-1;
                  CmdEndPoint = ((EnergyEndPoint+1) * WIN_SHIFT)-1;

                  // calculate frame features
                  FrameInd = 0;
                  // enter samples
                  for (m=CmdStartPoint; m<=CmdEndPoint; m=m+WIN_SHIFT)
                  {
                        // build frames
                        sampleMAX = 0;
                        r=0;
                        for (MaxInd=m-WIN_WIDTH; MaxInd<=m-1; MaxInd++)
                        {
                              //Copying and windowing
                              Temp32 =
                              ShortRecordedWave[MaxInd]*TimeWin16[r];

                              shortTempWave[r] = (short)(Temp32>>15);
                              if (abs(shortTempWave[r]) > sampleMAX)
                                    sampleMAX = abs(shortTempWave[r]);

                              r++;
                        }

                        //normalization
                        for (r=0;r<WIN_WIDTH;r++)
                        {
                              Temp32 = (int) (shortTempWave[r]<<(16-1) );
                              Temp32 = Temp32/sampleMAX;
                              shortTempWave[r] = (short) Temp32;
                        }
                        CalcFrameFeatures(shortTempWave, CoeffSet);
                        for (r=0;r<COEFF_ORDER;r++)
                        {
                              FrameFeatures[FrameInd][r] = CoeffSet[r];
                        }
                        FrameInd = FrameInd + 1;
                  }

if (SystemMode == 0) // 0=Training
{
      // Training mode
      //save values of reference frame features
      //save frame catalog in array to be saved in file

      Ref_Frame_Catlg[CommandInd][Speaker] = FrameInd;
      for (testInd=0; testInd<ENERGY_LENGTH; testInd++)
      {
            for (testInd2=0; testInd2<COEFF_ORDER;
            testInd2++)
            {
                  Ref_Frame_Features[CommandInd]
                  [Speaker][testInd][testInd2]=
                  FrameFeatures[testInd][testInd2];
            }
      }

      if(CommandInd == Repeats-1)
      {
```

```
            //last command thus save reference values
            file = fopen("ReferenceCatalog16.h","w");
            fprintf(file,"%s","// File name: ReferenceCatalog16.h\n\n");
            fprintf(file,"%s%d%s","short Ref_Frame_Catlg[][",
            MAX_GROUP_SIZE,"] ={\n");

            for (testInd=0; testInd<MAX_WORD_SIZE; testInd++)
            {
                  fprintf(file,"%s","{");
                  for (testInd2=0; testInd2<MAX_GROUP_SIZE; testInd2++)
                  {
                        fprintf(file,"%d%s",
                        Ref_Frame_Catlg[testInd][testInd2], ", ");
                  }
                  fprintf(file,"%s","},\n");
            }

            fprintf(file,"%s","};\n");
            fclose(file);

            file = fopen("ReferenceFeatures16.h","w");
            for(testInd=0; testInd<MAX_WORD_SIZE; testInd++)
            {
                  for(testInd2=0; testInd2<MAX_GROUP_SIZE; testInd2++)
                  {
                        for(testInd3=0; testInd3<ENERGY_LENGTH;
                        testInd3++)
                        {
                              for(testInd4=0; testInd4<COEFF_ORDER;
                              testInd4++)
                              {
                              fprintf(file,"%d ",
Ref_Frame_Features[testInd][testInd2][testInd3][testInd4]);
                              }
                              fprintf(file,"%s","\n");
                        }
                  }
            }
            fclose(file);
      }
      else
      {
            // reset buffer index i to record new command < Repeats
            i=0;
            IRQ_globalEnable(); //enable interrupts
      }

}
else //SystemMode == 1)
{
      // Recognition mode
      // run DTW Recognition

      xSizeAct = FrameInd;
      ySizeRef = MAX_GROUP_SIZE;
      xSizeRef = MAX_WORD_SIZE;
      //through speakers
      for(GroupInd=0; GroupInd<ySizeRef; GroupInd++)
      {
            //through reference commands
            for(WordInd=0;WordInd<xSizeRef;WordInd++)
```

```
        {
              zSizeRef = Ref_Frame_Catlg[WordInd][GroupInd];
              // copy Reference Features of one Command
              for (testInd=0; testInd<zSizeRef; testInd++)
              {
                    for (testInd2=0; testInd2<COEFF_ORDER;
                          testInd2++)
                    {
                    OneCmd_RefFrameFeatures[testInd][testInd2] =
      Ref_Frame_Features[WordInd][GroupInd][testInd][testInd2];
                    }
              }
              TotalDistance = DTW_Recognition(zSizeRef,
                    OneCmd_RefFrameFeatures,xSizeAct, FrameFeatures);

              // normalize distance
              if(TotalDistance<(zSizeRef + xSizeAct))
              {
                    Temp32 = (int) (TotalDistance << (16-1) );
                    Temp32 = Temp32/(zSizeRef + xSizeAct);
                    NormDistance = (short) Temp32;
              }
              else
              {
                    NormDistance = TotalDistance/(zSizeRef +
                    xSizeAct);
              }

              // printf("NormDistance = %hd\n", NormDistance);
              if (NormDistance < MinDistance)
              {
                    MinDistance = NormDistance;
                    RecoWord_j = GroupInd;
                    RecoWord_i = WordInd;
              }
        }
  }

              if (MinDistance > MAX_DISTANCE)
              {
                    //disp(CommandsSpeaker
                    (MinDistance_i,MinDistance_j));
                    RecoResult = 1;
                    printf("unknown command!\n");
              }
              else
              {
                          printf("%s\n","Recognized Word: ");
                    for(testInd=0;testInd<40;testInd++)
                    {
                          printf("%c",
                                CommandsSpeaker[RecoWord_i]
                                [RecoWord_j][testInd]);
                          }
                          printf("\n");
                    }
                    MinDistance = 32767;
                    // reset buffer index i to record new command <
                    Repeats
                    i=0;
                    IRQ_globalEnable(); //enable interrupts
```

```
                }
            }
        }
}

                if(SystemMode == 0)
                {
                    //training mode
                    // increment to process another command
                    CommandInd++;
                }
                else
                {
                    //System mode = 1 thus recognition mode
                    //reset to continue processing
                    CommandInd = 0;
                }
            }
        }
}
```

9.2.2 SpeechRecognition.h

```
//    Header-File name   |          Autor         |       Last Update
// ----------------------|------------------------|-----------------------
//   SpeechRecognition.h |    Ziad Ghadieh        |       01.03.2007
//
//
//
// Description:
// ------------
// contains libraries, Macros and function prototypes declarations
//
// Input parameters:
// --------------------
// none
//
// Output parameters:
// --------------------
// none
// ---------------------------------------------------------------------

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <math.h> // abs()

// MACROS
#define FS  8000 //Sample Frequency in Hz
#define RECORD_TIME 2 //recording time in s
#define WIN_WIDTH (32*FS/1000) //32ms
#define WIN_SHIFT (16*FS/1000) //16ms
#define NB_WAV_SAMPLES (RECORD_TIME*FS) //Number of recorded wave Samples
#define NO_SPEECH_INTERVAL 5*WIN_WIDTH // no-speech-interval
#define MIN_PULSE_WIDTH 2*WIN_WIDTH //minimal acceptable pulse width
#define     MAX_PULSE_DIST 3*WIN_WIDTH //maximal acceptable distance
between pulses
```

```
#define     MAX_TH_DIST 5*WIN_WIDTH //maximal acceptable distance between
ThLow and ThHigh
#define     MAX_WORD_WIDTH 50*WIN_WIDTH //maximal accceptable word length
[ca 1,6s]
#define ENERGY_LENGTH NB_WAV_SAMPLES/WIN_SHIFT
#define COEFF_ORDER  10 //between 8 and 12 [Rab81] Z.G: other values TB
tested
#define MAX_DISTANCE 93 //from tests
#define MAX_GROUP_SIZE 1 // number of Speakers (or several templates for
one command)
#define MAX_WORD_SIZE 12 // number of commands

// functions prototypes
short FIR_HP(short FIR_HP_delays[], short FIR_HP_coe[], short
N_HP_delays, short x_HP);
short FIR_LP(short FIR_LP_delays[], short FIR_LP_coe[], short
N_LP_delays, short x_LP);

unsigned short DetectEndPoints(unsigned short Ind, short SignalEnergy[],
short EnergyThLow, short EnergyThHigh,
unsigned short *EnergyEndPoint, unsigned short *EnergyStartPoint,
unsigned short *LastEndPoint, unsigned short *EndPointFlag,
unsigned short *StartPointFlag, unsigned short *HighThresholdFlag,
unsigned short *LowThresholdFlag, unsigned short *EnergyTempStartPoint,
unsigned short *CheckDistFlag);

void CalcFrameFeatures(short shortTempWave[], short CoeffSet[]);

short DTW_Recognition(unsigned short xSizeRef, short
Ref_Frame_Features[][COEFF_ORDER], unsigned short xSizeAct,
short Act_Frame_Features[][COEFF_ORDER]);
```

9.2.3 Prefilter.c

```
//    C-File name       |     Autor             |       Last Update
// ---------------------|-----------------------|--------------------
//   Prefilter.c        |    Ziad Ghadieh       |       01.03.2008
//
// Description:
// ------------
// based on:
// US 17-Jun-06
//
// Description:
// ============
// programm to do FIR filter computation in a subroutine
//
// Input parameters:
// --------------------
// short FIR_HP_delays[], FIR_LP_delays[]: filter delays
// short FIR_HP_coe[], FIR_LP_coe[]: filter coefficients
// short N_HP_delays, N_LP_delays: number of coefficients
// short x_HP, x_LP: input sample
//
// Output parameters:
// --------------------
// short y: filter result
// ------------------------------------------------------------------------
```

```
short FIR_HP(short FIR_HP_delays[], short FIR_HP_coe[], short
      N_HP_delays, short x_HP)
{
      short i, y;
      int FIR_HP_accu32=0;

      // read input sample
      FIR_HP_delays[N_HP_delays-1] = x_HP;

      // accumulate in 32 bit variable
      // clear accu
      FIR_HP_accu32     = 0;
      // FIR filter routine
      for(i=0; i < N_HP_delays; i++)
            FIR_HP_accu32 += FIR_HP_delays[N_HP_delays-1-i] *
                  FIR_HP_coe[i];

      // loop to shift the delays
      for(i=1; i < N_HP_delays; i++)
            FIR_HP_delays[i-1] = FIR_HP_delays[i];

      // shift back by 15 bit to obtain short int 16 bit output
      y = (short) (FIR_HP_accu32 >>15);
      return y;
}

short FIR_LP(short FIR_LP_delays[], short FIR_LP_coe[], short
      N_LP_delays, short x_LP)
{
      short i, y;
      int FIR_LP_accu32=0;

      // read input sample
      FIR_LP_delays[N_LP_delays-1] = x_LP;
      // accumulate in 32 bit variable
      // clear accu
      FIR_LP_accu32     = 0;
      // FIR filter routine
      for(i=0; i < N_LP_delays; i++)
            FIR_LP_accu32 += FIR_LP_delays[N_LP_delays-1-i] *
                  FIR_LP_coe[i];

      // loop to shift the delays
      for(i=1; i < N_LP_delays; i++)
            FIR_LP_delays[i-1] = FIR_LP_delays[i];

      // shift back by 15 bit to obtain short int 16 bit output
      y = (short) (FIR_LP_accu32 >>15);
      return y;
}
```

9.2.4 Prefilter.h

```
//    C-File name       |     Autor             |       Last Update
// --------------------|-----------------------|-------------------
//   Prefilter16.h      |    Ziad Ghadieh       |       01.03.2008
//
//
// Description:
// ------------
// this file contains FIR filter coefficients and needed declarations
// originaly created in MATLAB using CalcFilterCoeffs16.m
//
// Input parameters:
// --------------------
// none
//
// Output parameters:
// --------------------
// none
// ------------------------------------------------------------------------

#define N_delays_FIR_LP 18
#define N_delays_FIR_HP 19
short H_filt_FIR_LP[N_delays_FIR_LP];
short H_filt_FIR_HP[N_delays_FIR_HP];

short b_FIR_LP[N_delays_FIR_LP]={
       303, -285, -14, 733, -1601, 1957,
       -857, -3148, 19438, 19438, -3148, -857,
       1957, -1601, 733, -14, -285, 303,};

short b_FIR_HP[N_delays_FIR_HP]={
       215, 197, 92, -263, -921, -1851,
       -2923, -3932, -4654, 27851, -4654, -3932,
       -2923, -1851, -921, -263, 92, 197, 215,};
```

9.2.5 TimeWin16.h

```
//    C-File name       |     Autor             |       Last Update
// --------------------|-----------------------|-------------------
//   CommandSpeaker16.h |    Ziad Ghadieh       |       01.03.2008
//
//
// Description:
// ------------
// this file contains the coeficients of the hamming Window
// the number of coefficients is WIN_WIDTH
// originaly created in MATLAB TimeWin16.m
//
// Input parameters:
// --------------------
// none
//
// Output parameters:
// --------------------
// none
// ------------------------------------------------------------------------
```

```
#pragma DATA_SECTION(TimeWin16, ".EXT_RAM")
short TimeWin16[]={
   2621,   2626,   2640,   2663,   2695,   2736,   2786,   2845,
   2913,   2991,   3077,   3172,   3276,   3388,   3509,   3639,
   3778,   3925,   4080,   4243,   4415,   4595,   4782,   4978,
   5181,   5392,   5610,   5836,   6069,   6309,   6555,   6809,
   7069,   7336,   7609,   7888,   8173,   8464,   8760,   9062,
   9369,   9681,   9998,  10319,  10646,  10976,  11310,  11649,
  11991,  12336,  12685,  13037,  13391,  13749,  14108,  14470,
  14834,  15199,  15566,  15935,  16304,  16674,  17045,  17416,
  17788,  18159,  18530,  18900,  19270,  19639,  20007,  20373,
  20738,  21101,  21461,  21820,  22176,  22529,  22879,  23226,
  23570,  23910,  24247,  24579,  24907,  25231,  25551,  25865,
  26175,  26479,  26778,  27072,  27360,  27642,  27918,  28188,
  28451,  28708,  28958,  29202,  29438,  29667,  29889,  30104,
  30311,  30510,  30702,  30886,  31061,  31229,  31388,  31539,
  31682,  31816,  31942,  32059,  32167,  32266,  32357,  32439,
  32511,  32575,  32630,  32675,  32712,  32739,  32758,  32767,
  32767,  32758,  32739,  32712,  32675,  32630,  32575,  32511,
  32439,  32357,  32266,  32167,  32059,  31942,  31816,  31682,
  31539,  31388,  31229,  31061,  30886,  30702,  30510,  30311,
  30104,  29889,  29667,  29438,  29202,  28958,  28708,  28451,
  28188,  27918,  27642,  27360,  27072,  26778,  26479,  26175,
  25865,  25551,  25231,  24907,  24579,  24247,  23910,  23570,
  23226,  22879,  22529,  22176,  21820,  21461,  21101,  20738,
  20373,  20007,  19639,  19270,  18900,  18530,  18159,  17788,
  17416,  17045,  16674,  16304,  15935,  15566,  15199,  14834,
  14470,  14108,  13749,  13391,  13037,  12685,  12336,  11991,
  11649,  11310,  10976,  10646,  10319,   9998,   9681,   9369,
   9062,   8760,   8464,   8173,   7888,   7609,   7336,   7069,
   6809,   6555,   6309,   6069,   5836,   5610,   5392,   5181,
   4978,   4782,   4595,   4415,   4243,   4080,   3925,   3778,
   3639,   3509,   3388,   3276,   3172,   3077,   2991,   2913,
   2845,   2786,   2736,   2695,   2663,   2640,   2626,   2621,
};
```

9.2.6 DetectEndPoints.c

```
//    C-File name             |     Autor             |       Last Update
// -------------------------|-----------------------|--------------------
//   DetectEndPoints.c        |    Ziad Ghadieh       |       01.03.2008
//
//
//
// Description:
// ------------
// DetectEndPoints() applies criteria to detect endpoints of a command.
//
// Input parameters:
// --------------------
// unsigned short Ind: Index of actual frame
// short SignalEnergy[]: energy of actual frame
// short EnergyThLow: low energy threshold
// short EnergyThHigh: high energy threshold
// unsigned short *LastEndPoint: Pointer to the last found endpoint
// unsigned short *EndPointFlag: Pointer to Flag wether an endpoint
//                                              has been already found
// unsigned short *StartPointFlag: Pointer to Flag wether a startpoint
//                                              has been already found
```

```
// unsigned short *HighThresholdFlag: Pointer to Flag wether the high
//                                          threshold has been crossed
// unsigned short *LowThresholdFlag: Pointer to Flag wether the low
//                                          threshold has been crossed
// unsigned short *EnergyTempStartPoint: poiner to the temporary stored
//                                                startpoint
// unsigned short *CheckDistFlag: pointer to flag wether the pulse width
//                      and distance between pulses need to be checked
//
// Output parameters:
// --------------------
// unsigned short *EnergyEndPoint: Pointer to endpoint
// unsigned short *EnergyStartPoint: Pointer to startpoint
// ------------------------------------------------------------------------

#include "SpeechRecognition.h"


unsigned short DetectEndPoints(unsigned short Ind, short SignalEnergy[],
short EnergyThLow, short EnergyThHigh,
unsigned short *EnergyEndPoint, unsigned short *EnergyStartPoint,
unsigned short *LastEndPoint, unsigned short *EndPointFlag,
unsigned short *StartPointFlag, unsigned short *HighThresholdFlag,
unsigned short *LowThresholdFlag, unsigned short *EnergyTempStartPoint,
unsigned short *CheckDistFlag)
{
      unsigned short CommandErrorFlag = 0; // 0 if Ok, 1 if faulty

      if (SignalEnergy[Ind] > EnergyThLow)
      {
            // first time excess of Low Th
            if (*LowThresholdFlag == 0)
                  //label the location of excess
                  *EnergyTempStartPoint = Ind-1;

            // Energy > low Th, high Th
            if (SignalEnergy[Ind] > EnergyThHigh)
            {
                  //reset endpoint
                  *EnergyEndPoint = 0;
                  // new pulse thus search for the endpoint
                  *EndPointFlag = 0;
                  if (*HighThresholdFlag == 0)
                  {
                        // first time excess of High Th
                        if (*StartPointFlag == 0)
                        {
                              // no start point yet
                              if (((Ind -
                                    *EnergyTempStartPoint)*WIN_SHIFT)>=
                                          MAX_TH_DIST)
                                    *EnergyStartPoint = Ind -
                                          (MAX_TH_DIST>>7);
                              else
                                    //take the labeled location
                                    *EnergyStartPoint =
                                          *EnergyTempStartPoint;

                              // start point found
                              *StartPointFlag = 1;
                        }
```

```
                        else
                        {
                              //(StartPointFlag == 1)
                              // a pulse already exists
                              // make the inter-pulse-distance check
                              if (((*EnergyTempStartPoint -
                                    *LastEndPoint)*WIN_SHIFT)<=
                                          MAX_PULSE_DIST)
                              {
                                    // pulses belong to the same word
                              }
                              else
                              {
                                    printf("multiple commands!\n");
                                    //CommandErrorFlag = 1;
                              }
                        }
                        //now excess of high Th exists
                        *HighThresholdFlag = 1;
                  }
                  *CheckDistFlag = 0;
            }

            else
            {
                  // low Th < Energy < high Th
                  if (*HighThresholdFlag == 1) // Energy falling below
                                               high Th,
                  {
                        // Energy falling below high Th
                        // take it as endpoint for the case
                        // the energy would not fall under low Th
                        *EnergyEndPoint = Ind;
                        *LastEndPoint = Ind;
                        *HighThresholdFlag = 0;
                        *CheckDistFlag = 1;
                  }
                  else
                  {
                        //HighThresholdFlag == 0
                        if ((*StartPointFlag == 1)&&(*EndPointFlag == 0))
                        {
                              if (*CheckDistFlag == 1)
                              {
                                    if (((Ind - *LastEndPoint)*WIN_SHIFT)
                                          >= MAX_TH_DIST)
                                    {
                                          // take current location as
                                          endpoint
                                          // make Max word length check
                                          if (((Ind - *EnergyStartPoint)
                                          *WIN_SHIFT)> MAX_WORD_WIDTH)
                                          {
                                                printf("not a
                                                      command!\n");
                                                CommandErrorFlag = 1;
                                          }
                                          else
                                          {
                                                *EnergyEndPoint = Ind;
                                                *LastEndPoint = Ind;
```

```
                                        }
                                        *EndPointFlag = 1;
                                  }
                            }
                      }
                }
          }
          //excess of low Th exists
          *LowThresholdFlag = 1;
    }

    else
    {
          // Energy <= low Th
          // check if the pulse is valiant
          if ((*StartPointFlag == 1) && (*EndPointFlag == 0))
          {
                if (((Ind - *EnergyTempStartPoint)*WIN_SHIFT)>=
                      MIN_PULSE_WIDTH)
                {
                      //take this pulse i.e take the new end point
                      if (((Ind - *EnergyStartPoint)*WIN_SHIFT)>
                            MAX_WORD_WIDTH)
                      {
                            printf("not a command!\n");
                            CommandErrorFlag = 1;
                      }
                      else
                      {
                            *EnergyEndPoint = Ind;
                            *LastEndPoint = Ind;
                      }
                      *HighThresholdFlag = 0;
                      *EndPointFlag = 1;
                }
                else // pulse is too narrow thus reject it.
                {
                      // if it is the first pulse in the utterance
                      // then reject start point
                      *EnergyEndPoint = *LastEndPoint;
                      *EndPointFlag = 1;
                      if (*LastEndPoint == 0)
                      {
                            // no recent endpoints exist
                            //thus actual pulse is first pulse
                            // reject the startpoint
                            *StartPointFlag = 0;
                            *EnergyStartPoint = 0;
                      }
                }
          }
          *CheckDistFlag = 0;
          *LowThresholdFlag = 0;
    }
    return(CommandErrorFlag);
}
```

9.2.7 CalcFrameFeatures.c

```
//    C-File name             |     Autor             |       Last Update
// -------------------------|-----------------------|-------------------
//   CalcFrameFeatures.c     |    Ziad Ghadieh       |       01.03.2008
//
//
// Description:
// ------------
// CalcFrameFeatures() calculates the frame features of each assigned
// command, in this case the autocorrelation coefficients are calculated
//
// Input parameters:
// --------------------
// shortTempWave[]: input samples (filtered, windowed and normalized)
//
// Output parameters:
// --------------------
// short CoeffSet[]: one Set of coefficients (the number of Coefficients
// is COEFF_ORDER)
// ---------------------------------------------------------------------

#include "SpeechRecognition.h"


void CalcFrameFeatures(short shortTempWave[], short CoeffSet[])
{

      unsigned short CoeffInd, CorrInd;
      int Temp32=0;
      short Temp16=0;
      int CorrCoeffs;

      // Calculate frame features (Autocorrelation coefficients)
      for(CoeffInd=0; CoeffInd<COEFF_ORDER; CoeffInd++)
      {

            // reset accumulating Coefficient
            CorrCoeffs = 0;

            for(CorrInd=0;CorrInd<(WIN_WIDTH-CoeffInd);CorrInd++)
            {
                  Temp32 = (shortTempWave[CorrInd] *
                        shortTempWave[CorrInd + CoeffInd]);
                  // right shift to avoid overflow
                  Temp16 = (short)(Temp32>>15);
                  CorrCoeffs = CorrCoeffs + Temp16;
            }
            // right shift to save result in short
            CoeffSet[CoeffInd] = (short)(CorrCoeffs>>15);
      }
}
```

9.2.8 DTW_Recognition.c

```
//    Header-File name   |          Autor         |       Last Update
// ----------------------|------------------------|-----------------------
//   DTW_Recognition.c   |    Ziad Ghadieh        |       03.02.2008
//
//
// Description:
// ------------
// DTW_Recognition() perform the regognition task
//
// Input parameters:
// -------------------
// unsigned short xSizeRef: length of Reference frame features
// short Ref_Frame_Features[][COEFF_ORDER]: Reference frame features
//                                                    of one command
// unsigned short xSizeAct: length of actual frame features
// short Act_Frame_Features[][COEFF_ORDER]: actual frame features
//
// Output parameters:
// --------------------
// short distance[xSizeRef-1][xSizeAct-1]: accumulated minimal distance
// ---------------------------------------------------------------------

#include "SpeechRecognition.h"

short DTW_Recognition(unsigned short xSizeRef,
                              short Ref_Frame_Features[][COEFF_ORDER],
                              unsigned short xSizeAct,
                              short Act_Frame_Features[][COEFF_ORDER])
{
      unsigned short CoeffInd;
      short Ref_Coeff;
      short Act_Coeff;
      int TempDistance;
      short shortTempDistance;
      short shortTempDistance1;
      short shortTempDistance2;
      short shortTempDistance3;
      extern short distance[ENERGY_LENGTH][ENERGY_LENGTH];
      unsigned short ArrayIndRef;
      unsigned short ArrayIndAct;
      TempDistance = 0;
       // algorithm initialization
       // distance(0,0)
       for (CoeffInd = 0; CoeffInd<COEFF_ORDER; CoeffInd++)
       {
             Ref_Coeff = Ref_Frame_Features[0][CoeffInd];
             Act_Coeff = Act_Frame_Features[0][CoeffInd];

             TempDistance = TempDistance + ((Ref_Coeff*Ref_Coeff))
                   - ((Act_Coeff*Ref_Coeff)) - ((Act_Coeff*Ref_Coeff))
                   + ((Act_Coeff*Act_Coeff));
       }
       distance[0][0] = (short)(TempDistance);

       //initialize borders
       for (ArrayIndRef=1;ArrayIndRef<xSizeRef;ArrayIndRef++)
       {
             // set as maximum
             distance[ArrayIndRef][0] = 32767;
```

```
}
for (ArrayIndAct=1;ArrayIndAct<xSizeAct;ArrayIndAct++)
{
      // set as maximum
      distance[0][ArrayIndAct] = 32767;
}
// through reference frames
for (ArrayIndRef=1;ArrayIndRef<xSizeRef;ArrayIndRef++)
{
      // through actual frames
      for(ArrayIndAct=1;ArrayIndAct<xSizeAct;ArrayIndAct++)
      {
            TempDistance = 0;
            // calculate actual distance
            for (CoeffInd=0;CoeffInd<COEFF_ORDER;CoeffInd++)
            {
                  Ref_Coeff =
                        Ref_Frame_Features[ArrayIndRef][CoeffInd];
                  Act_Coeff =
                        Act_Frame_Features[ArrayIndAct][CoeffInd];
                  TempDistance = TempDistance +
                 ((Ref_Coeff*Ref_Coeff))-((Act_Coeff*Ref_Coeff))
                -((Act_Coeff*Ref_Coeff)) + ((Act_Coeff*Act_Coeff));
            }
            shortTempDistance = (short)(TempDistance);
            // calculate 3 possible paths
            //avoid overflow,
            //take the maximum in case of overflow, otherwise
                 acculmulate
            if(distance[ArrayIndRef-1][ArrayIndAct] < (32767-
                 shortTempDistance))
                 shortTempDistance1 =
                       distance[ArrayIndRef-1][ArrayIndAct] +
                             (shortTempDistance);
            else
                  shortTempDistance1 = 32767;

            if(distance[ArrayIndRef-1][ArrayIndAct-1] < (32767-
                 (2*shortTempDistance)))
                 shortTempDistance2 =
                       distance[ArrayIndRef-1][ArrayIndAct-1] +
                       (shortTempDistance) + (shortTempDistance);
            else
                  shortTempDistance2 = 32767;
            if(distance[ArrayIndRef][ArrayIndAct-1] < (32767-
                 shortTempDistance))
                  shortTempDistance3 =
                       distance[ArrayIndRef][ArrayIndAct-1] +
                             (shortTempDistance);
            else
                  shortTempDistance3 = 32767;

            //find minimum
            distance[ArrayIndRef][ArrayIndAct] =
                       shortTempDistance1;

            if (shortTempDistance2 <
                 distance[ArrayIndRef][ArrayIndAct])
                  distance[ArrayIndRef][ArrayIndAct] =
                       shortTempDistance2;
```

```
                    if (shortTempDistance3 <
                         distance[ArrayIndRef][ArrayIndAct])
                         distance[ArrayIndRef][ArrayIndAct] =
                               shortTempDistance3;
               }
         }

         return(distance[xSizeRef-1][xSizeAct-1]);
}
```

9.2.9 CommandSpeaker16.h

```
//    C-File name             |     Autor             |       Last Update
// -------------------------|-----------------------|-------------------
//   CommandSpeaker16.h       |    Ziad Ghadieh       |       01.03.2008
//
//
// Description:
// ------------
// this file contains the commands in text form
// originaly created in MATLAB using SaveCommandSpeaker16.m
//
// Input parameters:
// --------------------
// Output parameters:
// --------------------
// ------------------------------------------------------------------------

#include "SpeechRecognition.h"

#pragma DATA_SECTION(CommandsSpeaker, ".EXT_RAM")
char CommandsSpeaker[MAX_WORD_SIZE][MAX_GROUP_SIZE][40] = {
{{"vorwaerts by ZiadGh                    "},
},
{{"rueckwaerts by ZiadGh                  "},
},
{{"rechts by ZiadGh                       "},
},
{{"links by ZiadGh                        "},
},
{{"umdrehen by ZiadGh                     "},
},
{{"rechtsdrehen by ZiadGh                 "},
},
{{"linksdrehen by ZiadGh                  "},
},
{{"stopp by ZiadGh                        "},
},
{{"schneller by ZiadGh                    "},
},
{{"normal by ZiadGh                       "},
},
{{"langsamer by ZiadGh                    "},
},
{{"hugo by ZiadGh                         "},
},
};
```

- Ende des Dokumentes -

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 24. April 2008. _____

Ziad Ghadieh