Fachhochschule Westküste
Hochschule für Wirtschaft & Technik

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Christoph Klaukin

Entwicklung und Realisierung eines Systems für die automatische Spracherkennung mit einem erweiterten TI DSP Modul zur Robotersteuerung

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

Christoph Klaukin

Entwicklung und Realisierung eines Systems für die automatische Spracherkennung mit einem erweiterten TI DSP Modul zur Robotersteuerung

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 27. Januar 2010

Christoph Klaukin

Thema der Masterarbeit

Entwicklung und Realisierung eines Systems für die automatische Spracherkennung mit einem erweiterten TI DSP Modul zur Robotersteuerung

Stichworte

Spracherkennung, Einzelworterkennung, *Matlab®*, DSP, Wortisolierung, LPC-Koeffizienten, Levinson-Durbin-Algorithmus, DTW, Mikroprozessor

Kurzzusammenfassung

Entwickelt wird eine Spracherkennung zur Steuerung selbst fahrender Roboter. Hierzu wird ein vollständiges Simulationsmodell in *Matlab®* programmiert, welches sowohl das Anlegen einer Referenzdatenbank als auch die Spracherkennung umfasst. Dabei wird eine Einzelworterkennung mit kleinem Wortschatz realisiert; Wort- und Sprecheranzahl können erweitert werden.
Die Spracherkennung arbeitet blockbasiert, wobei die einzelnen Blöcke überlappend angeordnet werden. Zur Vorverarbeitung wird ein FIR-Bandpassfilter eingesetzt. Eine Isolierung des Wortes erfolgt mit Hilfe der logarithmierten quadratischen Energie der einzelnen Blöcke durch eine aufwendige Wortgrenzenerkennung. Zur Identifikation werden die Merkmale eines jeden Blockes eines Wortes durch LPC-Koeffizienten, die mit dem Levinson-Durbin-Algorithmus berechnet werden, dargestellt. Für den Mustervergleich wird DTW mit einer lokalen Pfadeinschränkung und mehreren Bewertungskriterien eingesetzt. Abschließend wird die Spracherkennung auf einem DSP, der über eine selbst entwickelte Kommunikationsplatine zwei bestimmte Roboter ansteuern kann, implementiert.

Christoph Klaukin

Title of the master thesis

Development and implementation of a system for automatic speech recognition with an extended TI DSP module to navigate a robot

Keywords

Speech recognition, isolated word recognition, *Matlab®*, DSP, speech endpoint detection, LPC parameter, Durbin's method, DTW, microprocessor

Abstract

A system for automatic speech recognition is being developed to navigate a self-driving robot. Therefore a complete simulation model is being programmed in *Matlab®*, which includes the management of the reference database as well as the speech recognition. The isolated word recognition is being realized with a small word pool although the amount of words and speakers can be extended.
The speech recognition system works block-based in which the blocks are sorted overlappingly. A FIR bandgap filter is being used during the preprocessing.The isolation of a word is achieved with a complex endpoint detection, using the logarithmical quadratic energy of the individual blocks. To identify the word, the characteristics of each block of each word is being represented by LPC parameters, calculated with the Durbin's method. For the pattern comparison the DTW is being used with a local path restriction and several evaluation criterias.
After that, the speech recognition is being implemented on a DSP, which can direct two distinct robots with an especially developed communication board.

Inhaltsverzeichnis

1 Einleitung 8
1.1 Analyse einer bereits vorhandenen Masterarbeit [2] 8
1.2 Lösungsansatz 9
1.3 Verwendete Software und Hardware 9
2 Die verwendeten Roboter 11
2.1 Der *Robotino*® 11
2.1.1 Omnidirektionaler Antrieb 12
2.1.2 Interne Hardware 13
2.1.3 Kommunikation über RS232 14
2.2 Der Roboter von *fischertechnik*® 16
2.2.1 Kommunikation mit dem *ROBO Interface* 16
2.2.2 Software für das *ROBO Interface* 17
3 Konzept des Gesamtsystems 18
4 Entwicklung eines Simulationsmodells mit *Matlab*® 20
4.1 Auswahl des Verfahrens 20
4.2 Gesamtmodell der Spracherkennung 21
4.3 Vorverarbeitung 22
4.3.1 Bandpassfilter 22
4.3.2 Blockbildung 23
4.3.3 Energieberechnung 24
4.3.4 Isolierung des Wortes 26
4.4 Merkmalsextraktion 29
4.4.1 LPC-Koeffizienten 30
4.5 Referenzdatenbank 34
4.6 Mustervergleich mit DTW 35
4.6.1 Bewertungskriterien für die DTW-Ergebnisse 38
4.7 Wichtige Systemparameter 38
4.8 Programmierte Funktionen für die Simulation mit *Matlab*® 40
4.9 Bedienung und Ablauf des Simulationsmodells 43
4.9.1 Verwaltung der Referenzdatenbank mit *manage_reference_database.m* 43
4.9.2 Spracherkennung mit *word_recognition.m* 47
5 Testergebnisse des Simulationsmodells 49
6 Hardware der Kommunikationsplatine 51
6.1 Prozessorauswahl 51
6.2 Spannungsversorgung 53
6.3 Die beiden RS232-Schnittstellen 54
6.3.1 Verbindung mit dem DSP 54
6.3.2 Verbindung mit dem *Robotino*® 54
6.4 Display und Taster 55
6.5 LEDs 56
6.6 Das *ROBO Interface* 56
6.7 Testen der Hardware vor dem Platinenlayout 57
6.8 Fertige Hardware 58
7 Software der Kommunikationsplatine 60
7.1 Modulare Struktur und Funktionsgruppen 60
7.2 Struktur der *main.c* 60
7.3 Definitionen in der *const.h* 62
7.4 Die einzelnen Quellcodedateien 63
7.4.1 Ansteuerung der Motoren (*motor.c* und *motor.h*) 63

7.4.2 Kommunikation mit dem DSP (*usart_dsp.c*) 64
7.4.3 Kommunikation mit dem *Robotino*® (*usart_mc.c*) 65
7.4.4 Display (*lcd.c*) 65
7.4.5 LEDs (*led.c*) 67
7.4.6 Taster am Roboter (*buttons.c*) 68
7.4.7 Fehler-Timer (*error_timer.c*) 68
7.4.8 Wartefunktion (*wait.c*) 69
7.4.9 Kommunikation mit dem *ROBO Interface* (*robo_interface.c*) 70
8 Implementierung des Simulationsmodells auf dem DSP 72
8.1 Verwendete Hardware 72
8.2 Die grundlegende Struktur der DSP-Software 73
8.3 Referenzdatenbank 74
8.4 Blockbasierte Aufnahme der Samples per EDMA 74
8.5 Bandpassfilter und Konvertieren zu Fließkommazahlen 75
8.6 Ping-Pong-Puffer zur Blockbildung, Hanning-Fenster, Energieberechnung 76
8.7 Isolierung des Wortes und Merkmalsextraktion 78
8.8 Mustervergleich mit DTW 80
8.9 Globale Systemparameter und globale Variablen 81
9 Testergebnisse der DSP-Implementierung 82
10 Ausblick und Fazit 83
11 Quellenverzeichnis 85
12 Anhang 87
12.1 Verzeichnis fachspezifischer Abkürzungen 87
12.2 Schaltpläne und Layout der Kommunikationsplatine 88
12.3 Stücklisten 92
12.3.1 Stückliste für den Aufbau auf einer Lochrasterplatine 92
12.3.2 Stückliste der gefertigten Kommunikationsplatine 93
12.4 Inhalt der CD-ROM 94

Abbildungsverzeichnis

Abbildung 2-1: Der verwendete, modifizierte *Robotino*® 11
Abbildung 2-2: Funktion des omnidirektionalen Antriebes (aus [3]) 12
Abbildung 2-3: Allseitenrad des omnidirektionalen Antriebes 13
Abbildung 2-4: Interne Hardware des *Robotino*® 13
Abbildung 2-5: Der aus Teilen des *ROBO Mobile Set* von *fischertechnik*® gebaute Roboter 16
Abbildung 2-6: Das grafische Programm für das *ROBO Interface* 17
Abbildung 3-1: Überblick über das Gesamtsystem 18
Abbildung 3-2: Mikrofonverstärker und montierte Mikrofonkapsel 19
Abbildung 4-1: Simulationsmodell für das Anlegen der Referenzdatenbank 21
Abbildung 4-2: Simulationsmodell der Spracherkennung 21
Abbildung 4-3: Ablauf der Vorverarbeitung 22
Abbildung 4-4: Amplitudengang des Bandpassfilters bei der *Matlab*®-Simulation 23
Abbildung 4-5: Blockbildung beim Simulationsmodell in *Matlab*® 24
Abbildung 4-6: Amplitude und Energieverläufe des Wortes „rückwärts“ 25
Abbildung 4-7: Zustandsautomat der Wortgrenzenerkennung 26
Abbildung 4-8: Funktionsweise der Wortgrenzenerkennung am Beispiel „rückwärts“ 27
Abbildung 4-9: FIR-Prädiktor N-ter Ordnung in transversaler Struktur (nach [10]) 30
Abbildung 4-10: Rekursive Schleife des Levinson-Durbin-Algorithmus 32
Abbildung 4-11: Strukturierung der Referenzdatenbank 34
Abbildung 4-12: Ablauf der DTW-Berechnung 37
Abbildung 4-13: Flussdiagramm der Funktion `wav_to_lpc` 41
Abbildung 4-14: Hauptmenü von *manage_reference_database.m* 43
Abbildung 4-15: Flussdiagramm des Menüpunktes `apply a new speaker` 44
Abbildung 4-16: Flussdiagramm des Menüpunktes `apply a new word / renew a word` 45
Abbildung 4-17: Hauptmenü von *word_recognition.m* 47
Abbildung 4-18: Flussdiagramm des Menüpunktes `word recognition` 47
Abbildung 6-1: Übersicht über die einzelnen Hardwarekomponenten (Blockschaltbild) 51
Abbildung 6-2: Anschlussbelegung des Pfostensteckers für das LCD und die vier Taster 55
Abbildung 6-3: Aufbau der Hardware auf einer Lochrasterplatine 57
Abbildung 6-4: Gefertigte, bestückte Platine (Oberseite) 58
Abbildung 6-5: Gefertigte, bestückte Platine (Seitenansicht) 59
Abbildung 7-1: Flussdiagramm des Hauptprogrammes für den Mikroprozessor (*main.c*) 61
Abbildung 7-2: Timing des Displays beim Schreibzugriff [14] 65
Abbildung 8-1: *D.Module.C6713*, *D.Module.PCM3003* und Kommunikationsplatine 72
Abbildung 8-2: Ablauf des Hauptprogrammes auf dem DSP 73
Abbildung 8-3: Amplitudengang des Bandpassfilters bei der DSP-Implementierung 76
Abbildung 8-4: Blockbildung bei der DSP-Implementierung 77
Abbildung 8-5: Zustandsautomat der Wortgrenzenerkennung beim DSP 79
Abbildung 12-1: Schaltplan für den Aufbau auf einer Lochrasterplatine 88
Abbildung 12-2: Schaltplan der gefertigten Kommunikationsplatine 89
Abbildung 12-3: Obere Seite der Kommunikationsplatine in Originalgröße (TOP) 90
Abbildung 12-4: Untere Seite der Kommunikationsplatine in Originalgröße (BOTTOM) 90
Abbildung 12-5: Layer der Kommunikationsplatine übereinander (Originalgröße) 91

Tabellenverzeichnis

Tabelle 2-1: Aufbau des Steuerpaketes zum Motortreiber (nach [4]) 14
Tabelle 2-2: Aufbau des Antwortpaketes vom Motortreiber (nach [4]) 15
Tabelle 2-3: Kodierung der Befehle für die Steuerung des *ROBO Interface* 17
Tabelle 4-1: Vergleich zwischen HMM und DTW [8] 20
Tabelle 4-2: Parameter für das Bandpassfilter 22
Tabelle 4-3: Aufbau des zweidimensionalen Feldes *D* (nach [8]) 36
Tabelle 4-4: Berücksichtigung der vorherigen Werte für die Berechnung von $D_{n,m}$ (nach [8]) 36
Tabelle 5-1: Abschließendes Testergebnis der Spracherkennung bei bekannten Wörtern 50
Tabelle 6-1: Pinbelegung des Mikroprozessors (Port, Pin und Funktion nach [11]) 52
Tabelle 6-2: Pinbelegung des Steckers X14 für die Spannungsversorgung (nach [5]) 53
Tabelle 6-3: Pinbelegung des Steckers X10 für die serielle Schnittstelle (nach [5]) 55
Tabelle 6-4: Pinbelegung des 26-poligen Pfostensteckers am *ROBO Interface* [15] 56
Tabelle 7-1: Interpretation des Befehlbytes 62
Tabelle 7-2: Vorgegebene Wartezeiten bei der LCD-Ansteuerung (nach [14]) 66
Tabelle 12-1: Stückliste für den Aufbau auf einer Lochrasterplatine 92
Tabelle 12-2: Stückliste der gefertigten Kommunikationsplatine 93

Gleichungsverzeichnis

Gleichung 4-1: Berechnung der Energie eines Blocks [8] 24
Gleichung 4-2: Logarithmieren der Blockenergie [8] 25
Gleichung 4-3: Berechnung des Schätzwertes der linearen Prädiktion (nach [8], [9], [10]) ... 30
Gleichung 4-4: Bestimmungsgleichung für den Prädiktionsfehler (nach [8], [9], [10]) 30
Gleichung 4-5: Berechnung der Autokorrelationskoeffizienten (nach [8], [9], [10]) 31
Gleichung 4-6: Gleichungssystem der LPC-Koeffizienten-Berechnung (nach [8], [9], [10]) . 31
Gleichung 4-7: Direkte Berechnung der LPC-Koeffizienten (nach [8]) 31
Gleichung 4-8: Vorabberechnung von $alpha_{1,1}$ vor der rekursiven Berechnung [10] 31
Gleichung 4-9: Vorabberechnung von E_1 vor der rekursiven Berechnung [10] 32
Gleichung 4-10: Vorabberechnung von q_1 vor der rekursiven Berechnung [10] 32
Gleichung 4-11: Berechnung von k_i innerhalb der Rekursionsschleife (nach [10]) 33
Gleichung 4-12: Berechnung von E_i innerhalb der Rekursionsschleife (nach [10]) 33
Gleichung 4-13: Berechnung von $alpha_{i,j}$ innerhalb der Rekursionsschleife (nach [10]) 33
Gleichung 4-14: Setzen von $alpha_{i,i}$ innerhalb der Rekursionsschleife (nach [10]) 33
Gleichung 4-15: Berechnung von q_i innerhalb der Rekursionsschleife (nach [10]) 33
Gleichung 4-16: Kopieren der LPC-Koeffizienten a_i aus der Matrix *alpha* (nach [10]) 33
Gleichung 4-17: Übertragungsfunktion für den Prädiktionsfehler in Abbildung 4-9 (nach [10]) 33
Gleichung 4-18: Zusammenhang zwischen a_i und b_i (nach [10]) 33
Gleichung 4-19: Vektor-Quantisierung mit Betragsabstand (nach [8]) 35
Gleichung 4-20: Vektor-Quantisierung mit quadriertem Euklidischen Abstand (nach [8]) 35
Gleichung 4-21: Berechnen der einzelnen Feldelemente $D_{n,m}$ (nach [8]) 36
Gleichung 4-22: Zuweisung des ersten Feldelementes $D_{1,1}$ (nach [8]) 36
Gleichung 4-23: Berechnen der einzelnen Feldelemente $D_{n,1}$ der ersten Spalte (nach [8]) 36
Gleichung 4-24: Berechnen der einzelnen Feldelemente $D_{1,m}$ der ersten Zeile (nach [8]) 37
Gleichung 4-25: Normieren von $D_{N,M}$ (nach [2]) 38
Gleichung 7-1: Zeitdauer des Time-out (*error_timer.c*) 68
Gleichung 7-2: Wert für das Output Compare Register der Funktion `wait_one_ms` 69
Gleichung 7-3: Grundwert für das Output Compare Register (*robo_interface.c*) 70

1 Einleitung

Die Spracherkennung ist ein relativ neues Anwendungsfeld. Erst Anfang der 1990er Jahre kamen erste Programme für PCs auf den Markt, obwohl die Forschung auf diesem Gebiet in den 1960er Jahren begann, da erst seit dieser Zeit die nötige Rechenleistung seitens der Hardware gegeben war. Ein bereits 1984 von *IBM* vorgestelltes Spracherkennungssystem, welches etwa 5000 englische Einzelworte erkennen konnte, benötigte für einen Erkennungsvorgang mehrere Minuten Rechenzeit auf einem Großrechner [1]. Heutzutage werden automatische Systeme zur Spracherkennung mit begrenztem Wortschatz z. B. bei Dialogsystemen zur Fahrplanauskunft oder Systemen für telefonische Bestellungen mit Erkennungsraten von nahezu 100% eingesetzt.

Im Rahmen dieser Masterarbeit ist eine Spracherkennung zur Steuerung zweier unterschiedlicher, selbst fahrender Roboter zu entwickeln und aufzubauen. Die Anzahl der Sprachkommandos, die verstanden und von den Robotern umgesetzt werden sollen, ist auf acht begrenzt. Eine Ergänzung weiterer Worte soll möglich sein. Die Spracherkennung soll keine kontinuierliche Sprache, sondern einzelne Wörter erkennen können. Dieses wird als Einzelworterkennung bezeichnet.
Das zu entwickelnde System zur Spracherkennung soll zuerst ausführlich mit *Matlab*® simuliert, getestet und optimiert werden. Im Anschluss soll das so optimierte System auf einem DSP (engl.: Digital Signal Processor) implementiert werden, wobei die Verarbeitungszeit so kurz wie möglich sein soll. Der DSP soll dann die empfangenen Kommandos an den zu steuernden Roboter schicken. Für die Kommunikation zwischen DSP und Roboter ist eine geeignete Platine zu entwickeln, die das entsprechende Kommunikationsprotokoll der Roboter unterstützt und interpretieren kann.

1.1 Analyse einer bereits vorhandenen Masterarbeit [2]

Als Grundlage dient die im April 2008 abgegebene Masterarbeit von Herrn Ziad Ghadieh mit dem Titel „Automatische Spracherkennung zur Robotersteuerung auf dem TI DSP D.MODULE.C6713“.
In dieser Arbeit wird eine Einzelworterkennung simuliert und auf einem DSP umgesetzt. Eine Robotersteuerung erfolgt nicht. Die Spracherkennung arbeitet sprecherabhängig; der Sprecher muss vor dem Erkennungsvorgang antrainiert werden. Hierzu verfügt die Software über zwei Phasen, die Lernphase und die Erkennungsphase.
Es sind zwölf Kommandos vorgesehen: „vorwärts“, „rückwärts“, „rechts“, „links“, „umdrehen“, „rechtsdrehen“, „linksdrehen“, „stopp“, „schneller“, „normal“, „langsamer“ und der Name „Hugo“.
Um ein Wort mit dem Simulationsmodell zu erkennen, wird ein Bereich von zwei Sekunden aufgenommen. Dieser wird anschließend analysiert und das Ergebnis ausgegeben.
Bei der Umsetzung auf dem DSP ist die Vorgehensweise ähnlich. Auch hier wird ein Bereich mit zwei Sekunden Dauer aufgenommen und anschließend analysiert. Direkt im Anschluss wird ein neuer Bereich mit zwei Sekunden Dauer aufgenommen usw. Der aktuelle Status (Aufnahme / Analyse) wird über eine LED des verwendeten DSP-Boards angezeigt. Die Analyse benötigt eine Verarbeitungszeit von über 500 ms.

Im Folgenden werden die eingesetzten Verfahren und einige wichtige Parameter aufgeführt:
- In der *Matlab*®-Simulation wird mit dem Datentyp `double` gearbeitet, auf dem DSP mit dem Datentyp `short`.
- Als Eingangsfilter werden ein FIR-Hochpassfilter und ein FIR-Tiefpassfilter eingesetzt.

- Das System arbeitet mit einer Samplefrequenz von 8 KHz.
- Ein Datenblock umfasst 256 Samples, was 32 ms entspricht.
- Die Überlappung der Datenblöcke entspricht 50% (128 Samples), was 16 ms entspricht.
- Zum Isolieren eines Wortes wird die Energie als Summe der Beträge aller in einem Block vorhandenen Samples gebildet.
- Die Schwellwerte für die Isolierung eines Wortes werden aus den ersten 160 ms der Aufnahme automatisch bestimmt.
- Als Koeffizienten werden Autokorrelationskoeffizienten verwendet; die Anzahl pro Block wird auf 10 eingestellt.
- Vor dem Berechnen der Autokorrelationskoeffizienten wird ein Datenblock mit dem Hamming-Fenster bearbeitet und normiert.
- Zum Mustervergleich wird DTW (engl.: Dynamic Time Warping) mit dem quadrierten Euklidischen Abstand und einer lokalen Pfadeinschränkung eingesetzt.
- Als Bewertungskriterium der Ergebnisse des Mustervergleiches wird das kleinste Ergebnis ausgewählt. Weitere Kriterien sind nicht vorhanden.

Die in der abschließenden Zusammenfassung der Masterarbeit von Herrn Ziad Ghadieh aufgeführte Erkennungsrate beim Simulationsmodell von 100% konnte beim Testen seines Simulationsmodells nicht bestätigt werden.
Nicht in der Referenzdatenbank enthaltene Wörter wurden nicht abgewiesen.
Die DSP-Implementierung wurde nicht erneut getestet.

1.2 Lösungsansatz

Um eine zuverlässigere Erkennungs- und Abweisungsrate zu erreichen, wird ein komplett neues Simulationsmodell programmiert. Lediglich einige Schritte werden dem Sinn nach aus der vorhandenen Arbeit verwendet und angepasst. Dabei handelt es sich um den Mustervergleich per DTW und die Wortisolierung durch die Interpretation des Energieverlaufes. Es wird kein Quellcode übernommen.
Die Anzahl der vorgegebenen Wörter wird auf acht beschränkt. Diese sind „links“, „rechts“, „vorwärts“, „rückwärts“, „schneller“, „langsamer“, „halt“ und „drehe“.

Für die Kommunikation zwischen DSP und Roboter soll eine Kommunikationsplatine entwickelt werden. Diese soll von dem DSP nur dann ein kurzes Kommando erhalten, wenn dieser ein Wort identifiziert hat. Die Verbindung zwischen dem DSP und der Kommunikationsplatine soll über eine RS232-Schnittstelle stattfinden, da diese auf dem verwendeten DSP-Board vorhanden ist.
Die Kommunikationsplatine übernimmt die komplette Ansteuerung der beiden zur Verfügung stehenden Roboter und entlastet somit den DSP.

1.3 Verwendete Software und Hardware

Für die Realisierung der Aufgabenstellung ist eine umfangreiche Sammlung von Hard- und Software nötig. Diese wird in diesem Kapitel aufgelistet.

Für die Simulation der Spracherkennung wird folgende Software verwendet:
- *Matlab*® Version 6.5.0.196271 Release 13.0.1, 14.03.2003 des Herstellers *The MathWorks*™
- *Matlab*® Version 7.8.0.347 (R2009a), 12.02.2009 des Herstellers *The MathWorks*™

Die Entwicklung der Schaltpläne und des Platinenlayouts wird mit folgender Software durchgeführt:
- *Eagle* Version 5.4.0 des Herstellers *Cadsoft Computer GmbH*

Die Programmierung des Mikrocontrollers *ATmega 162* erfolgt mit:
- *Atmel® AVR® Studio* 4, Version 4.16 Build 638
- *WinAVR RC1* vom 06.03.2009
- *Atmel® AVR® ISP mk II*

Die Programmierung des digitalen Signalprozessors *TMS320C6713* erfolgt mit:
- *Code Composer Studio* 3.3.38.2 des Herstellers *Texas Instruments*
- *USB560m JTAG Emulator* des Herstellers *Blackhawk™*

Um das *ROBO Interface* von *fischertechnik®* zu programmieren, wird folgende Software verwendet:
- *ROBO Pro grafische Programmiersprache* V1.2.1.33 DEMO VERSION

Als Hardware kommen folgende Komponenten zum Einsatz:
- Ein *Robotino®* des Herstellers *Festo Didactic GmbH & Co. KG*
- Das *ROBO Mobile Set* des Herstellers *fischertechnik GmbH*
- Ein *D.Module.C6713* des Herstellers *D.SignT GmbH & Co. KG*
- Ein *D.Module.PCM3003* des Herstellers *D.SignT GmbH & Co. KG*

Messgeräte, Lötkolben und andere Werkzeuge werden nicht separat aufgelistet.

2 Die verwendeten Roboter

Ursprünglich war nur ein Roboter vorgesehen, der *Robotino*®. Im Laufe der Arbeit kam noch ein zweiter, kleinerer Roboter hinzu. Es handelt sich dabei um den Roboter, der aus dem *ROBO Mobile Set* von *fischertechnik*® gebaut werden kann. Jeder Roboter wird dabei unterschiedlich angesteuert, sowohl das Interface als auch das Protokoll der Datenübertragung sind verschieden. In diesem Kapitel werden die beiden unterschiedlichen Roboter erklärt.

2.1 Der *Robotino*®

Der *Robotino*® ist ein kompakter Roboter, der für Lehrzwecke konzipiert wurde. Im Lieferumfang befinden sich eine Ansteuerungssoftware sowie eine Sammlung von API-Klassen für eigene Programmierungen.

Abbildung 2-1: Der verwendete, modifizierte *Robotino*®

Der Roboter hat einen Durchmesser von ca. 35 cm. Die Steuerung erfolgt durch einen Einplatinen-PC (PC/104), auf dem eine Linux-Version läuft. Der Roboter kann kabellos von einem PC aus gesteuert werden. Hierzu ist in den *Robotino*® ein WLAN Access Point integriert. Über ein in den Deckel integriertes LCD und vier daneben angeordnete Tasten können Demo-Programme ausgewählt und abgefahren werden.
Für Bewegungen ist ein omnidirektionaler Antrieb mit drei Allseitenrädern – jeweils von einem separaten Motor angetrieben – integriert. Um Hindernisse zu erkennen, sind insgesamt neun Reflexionslichtschranken eingebaut.

Für die Spannungsversorgung sind zwei Akkumulatoren enthalten, damit der Roboter sich ohne eine Kabelverbindung frei bewegen kann.
Optional kann der Roboter mit zwei Lichtschranken zum Verfolgen einer Linie, einem induktiven Näherungssensor sowie einer Web-Kamera mit USB-Anschluss erweitert werden.

Um den *Robotino*® ohne den internen Einplatinen-PC ansteuern zu können, ist eine Modifikation nötig. Die serielle Schnittstelle, die für die Kommunikation zwischen dem Einplatinen-PC und dem Motortreiber verwendet wird (siehe 2.1.2), muss extern zugänglich gemacht werden. Ebenso muss eine Spannung von 5V DC aus dem Inneren des Roboters nach außen geführt werden.

2.1.1 Omnidirektionaler Antrieb

Durch den omnidirektionalen Antrieb ist es dem Roboter möglich, in jede beliebige Richtung zu fahren und sich dabei nicht zu drehen. So sind zum Beispiel seitliches Fahren oder Drehen auf der Stelle kein Problem. Allerdings müssen bei einer Bewegungsänderung alle Geschwindigkeiten und Drehrichtungen der drei Motoren neu berechnet werden.
Die Räder sind um 120° zueinander versetzt angeordnet. In der nachfolgenden Grafik ist ein Funktionsbeispiel dargestellt.

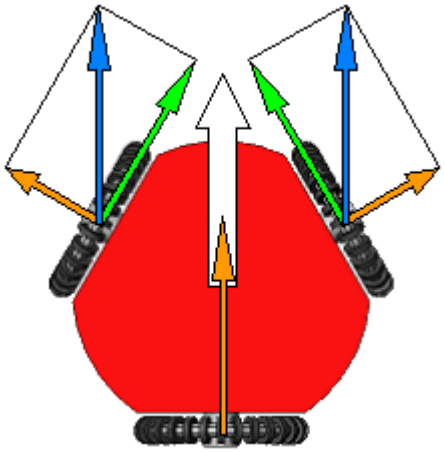

Abbildung 2-2: Funktion des omnidirektionalen Antriebes (aus [3])

Der weiße Pfeil stellt die Bewegungsrichtung des Roboters dar. Die orangefarbenen Pfeile deuten die Laufkräfte der Walzen der Allseitenräder an (Seitenkräfte), während die grünen Pfeile die Drehrichtung der Motoren und somit der Allseitenräder selbst darstellen. Es ist zu erkennen, dass das untere Rad nicht angetrieben wird und somit nur seitlich überrollt. Die beiden oberen Räder werden zwar angetrieben, drehen aber auch in gewissem Maße seitlich über. Hierbei ergibt sich die zusammengesetzte Bewegungsrichtung eines jeden Rades, dargestellt durch blaue Pfeile, aus dessen Antriebskraft und der Kraft des seitlichen Überrollens.
Ermöglicht wird das seitliche Überrollen durch den Einsatz von Allseitenrädern. Die bei dem *Robotino*® eingebauten Allseitenräder haben einen Durchmesser von ca. 8 cm und verfügen über sechs parallel zur Laufrichtung angeordnete Walzen. Der Motor befindet sich jeweils oberhalb des Rades parallel zu dessen Achse. Die Kraftübertragung erfolgt über einen Zahnriemen. Die nachfolgende Abbildung zeigt ein solches Rad im eingebauten Zustand.

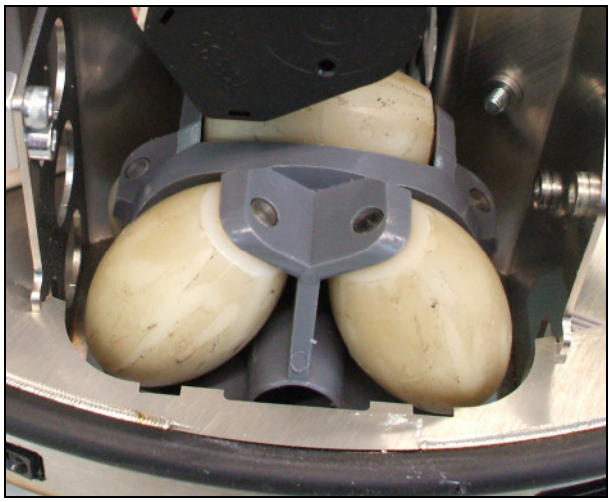

Abbildung 2-3: Allseitenrad des omnidirektionalen Antriebes

Die Funktionsweise dieser Allseitenräder ist einfach: Wird das Rad durch einen Motor angetrieben, so übertragen die Walzen diese Drehbewegung direkt auf den Untergrund. Wird das Rad zusätzlich seitlich geschoben (z. B. durch zwei andere Allseitenräder, welche den Roboter mit bewegen), so rollen die Walzen seitlich und ermöglichen somit die zusammengesetzte Seitwärtsbewegung bei reduziertem Reibungswiderstand.

2.1.2 Interne Hardware

Die interne Hardware des *Robotino*® besteht hauptsächlich aus zwei Komponenten, der Motortreiberplatine und einem Einplatinen-PC (*MOPSlcdSE*).

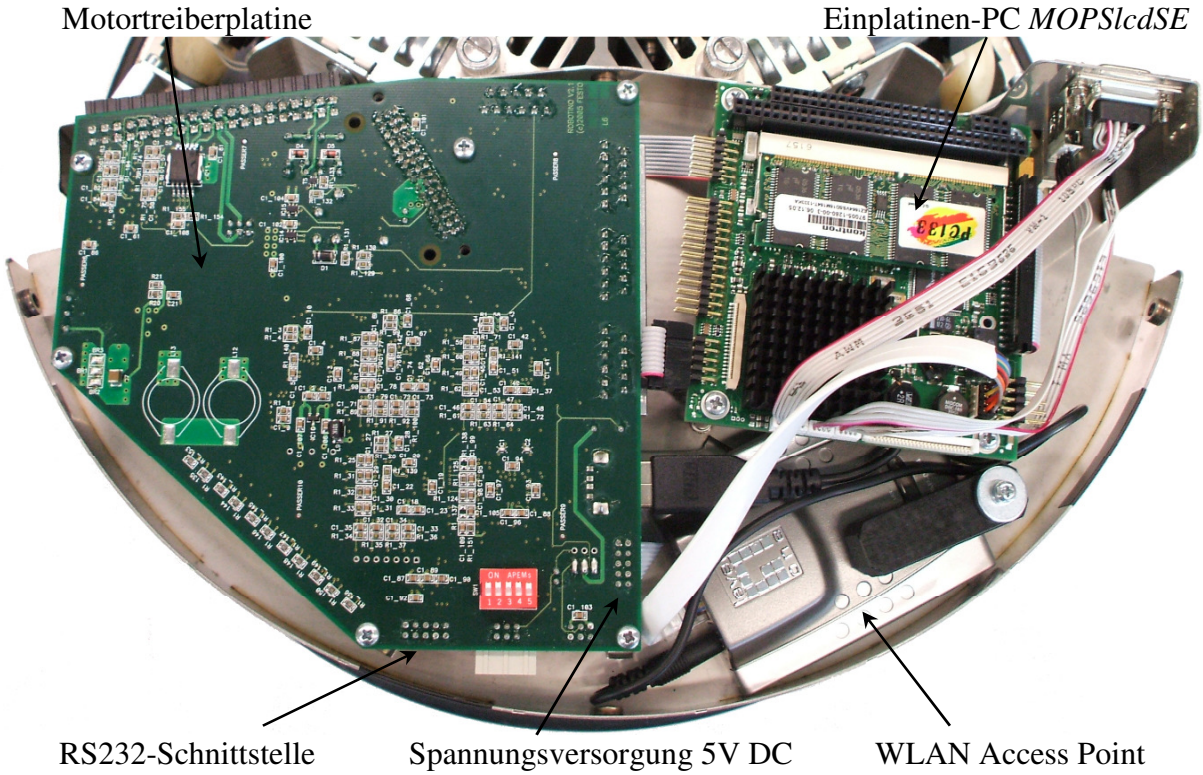

Abbildung 2-4: Interne Hardware des *Robotino*®

Die Motortreiberplatine enthält die komplette Elektronik zum Steuern der Motoren. Die Lichtschranken zur Hinderniserkennung sowie die optionalen Lichtschranken zum Verfolgen einer Linie und ein induktiver Näherungssensor werden ebenfalls an diese Platine angeschlossen. Außerdem sind die Spannungsversorgung und die Elektronik zum Ein- und Ausschalten des Roboters auf dieser Platine untergebracht.
Von dieser Motortreiberplatine führen zwei zehnpolige Flachbandkabel zum Einplatinen-PC, eins für die Verbindung der RS232-Schnittstelle und eins für die Spannungsversorgung des Einplatinen-PCs. Die Power-Taste in dem Gehäusedeckel ist die einzige Taste, die an die Motortreiberplatine angeschlossen ist.
An den Einplatinen-PC sind die vier Taster und das LCD im Gehäusedeckel angeschlossen. Für die Kommunikation mit einem externen PC ist ein WLAN Access Point vorhanden, der intern durch ein kurzes Ethernet-Kabel mit dem Einplatinen-PC verbunden ist. Unterhalb des Einplatinen-PCs ist ein Kartenleser für Speicherkarten (Compact Flash) eingebaut. Über diesen wird das auf einer Compact Flash Karte gespeicherte Betriebssystem (Linux) gebootet.
Zwei USB-Anschlüsse und ein VGA-Anschluss werden nach außen geführt. Hier können ein Monitor, eine Tastatur und, falls nötig, eine Maus angeschlossen werden, um den Einplatinen-PC wie einen „normalen“ PC zu bedienen.

2.1.3 Kommunikation über RS232

Die Kommunikation mit dem *Robotino*® erfolgt nach einem zeitkritischen Protokoll. Der Einplatinen-PC sendet ein 47 Byte großes Datenpaket an die Motortreiberplatine. Es enthält alle Steuerparameter für die Motoren. Die Motortreiberplatine antwortet mit einem 101 Byte großen Datenpaket, das alle Statusinformationen der Motortreiberplatine enthält (Steuerparameter der Motoren, Lichtschrankendaten u.Ä.). Bricht die Kommunikation ab, so werden alle Motoren nach einer Sekunde aus Sicherheitsgründen angehalten.
In der nachfolgenden Tabelle sind die für die Ansteuerung der Motoren verwendeten Bytes aufgelistet. Die anderen, nicht aufgeführten Bytes werden dauerhaft auf Null gesetzt.

Byte	Bit(s)	Bedeutung
0	0..7	Erstes Startbyte, R
1	0..7	Zweites Startbyte, E
2	0..7	Drittes Startbyte, C
3	0	Nothalt: 0 = stopp, 1 = ok
3	1	Power: 0 = an, 1 = aus
4	0	Motor 0 Bremse: 0 = an, 1 = aus
5	0	Modus: 0 = velocity, 1 = position
5	1	Richtung des Motors: 0 = negativ, 1 = positiv
6	0..7	Motorgeschwindigkeit
11	0..7	Regelparameter kp, Standardwert 0xFF
12	0..7	Regelparameter ki, Standardwert 0xFF
13	0..7	Regelparameter kd, Standardwert 0xFF
14-23		Wie Byte 4 – 13, nur für Motor 1
24-33		Wie Byte 4 – 13, nur für Motor 2
34-43		Wie Byte 4 – 13, nur für Motor 3
44	0..7	Erstes Stoppbyte, r
45	0..7	Zweites Stoppbyte, e
46	0..7	Drittes Stoppbyte, c

Tabelle 2-1: Aufbau des Steuerpaketes zum Motortreiber (nach [4])

Die einzigen Werte, die während des laufenden Betriebes geändert werden, sind die Geschwindigkeiten und die Drehrichtungen der drei Motoren. Alle anderen Werte bleiben unverändert. Dieser Roboter verwendet die Motoren 0 bis 2. Ein vierter Motor ist nicht vorhanden, obwohl die Motortreiberplatine dafür ausgelegt ist.

In der nachfolgenden Tabelle sind die vom Antwortpaket ausgewerteten Bytes aufgelistet. Die anderen, nicht aufgeführten Bytes werden ignoriert.

Byte	Bit(s)	Bedeutung
0	0..7	Erstes Startbyte, R
1	0..7	Zweites Startbyte, E
2	0..7	Drittes Startbyte, C
15	0..7	Analoger Eingang ADC1 (Bits 2..9)
16	0..7	Analoger Eingang ADC2 (Bits 2..9)
21	0..7	Analoger Eingang ADC7 (Bits 2..9)
35-55		Wie Byte 14 – 34, nur für Motor 1
56-76		Wie Byte 14 – 34, nur für Motor 2
77-97		Wie Byte 14 – 34, nur für Motor 3
98	0..7	Erstes Stoppbyte, r
99	0..7	Zweites Stoppbyte, e
100	0..7	Drittes Stoppbyte, c

Tabelle 2-2: Aufbau des Antwortpaketes vom Motortreiber (nach [4])

Die analogen Eingänge stehen mit einer Genauigkeit von 10 Bit zur Verfügung. Da jedoch für die Auswertung der an einigen der analogen Eingänge angeschlossenen Lichtschranken zur Hinderniserkennung keine große Genauigkeit erforderlich ist, werden nur die acht MSBs (engl.: Most Significant Bit) ausgewertet und die Genauigkeit der Verarbeitung somit auf acht Bit reduziert.
Der *Robotino*® ist mit insgesamt neun Reflexlichtschranken ausgestattet, die rings um den *Robotino*® angeordnet sind. Somit können Hindernisse in einer Entfernung von ca.10 – 15 cm erkannt werden.
Die Lichtschranken sind intern an die analogen Eingänge 1, 2 und 7 der drei Motortreiber angeschlossen. Ist kein Hindernis vor der entsprechenden Lichtschranke, beträgt der zugehörige Wert im Antwortpaket 0. Befindet sich das Hindernis unmittelbar vor der Lichtschranke, steigt der zugehörige Wert auf 255. Bei einem Abstand von ca. 10 cm beträgt der Rückgabewert 100. Dieser Wert wird bei der neu zu entwickelnden Kommunikationsplatine als kleinster zulässiger Abstand wie folgt implementiert (siehe Kapitel 7.2): Überschreitet der Rückgabewert von mindestens einer Lichtschranke den Wert 100, so wird der *Robotino*® sofort angehalten.

Außer den neun Rückgabewerten der Lichtschranken werden keine weiteren Daten des Antwortpaketes ausgewertet.

2.2 Der Roboter von *fischertechnik*®

Aus dem *ROBO Mobile Set* von *fischertechnik*® kann ein kleiner Roboter gebaut werden. Als Steuerung dient ein einfach anzuschließendes *ROBO Interface*, ebenfalls von *fischertechnik*®. Dabei handelt es sich um ein in ein Plexiglasgehäuse eingebautes Steuermodul, das per PC programmiert werden kann. Für Steuerungsanwendungen sind an dem *ROBO Interface* analoge und digitale Eingänge sowie digitale Ausgänge vorhanden.
Im Rahmen der Spracherkennung zur Robotersteuerung werden nur zwei Motoren angeschlossen. Die Befehle werden über vier digitale Eingänge übertragen.

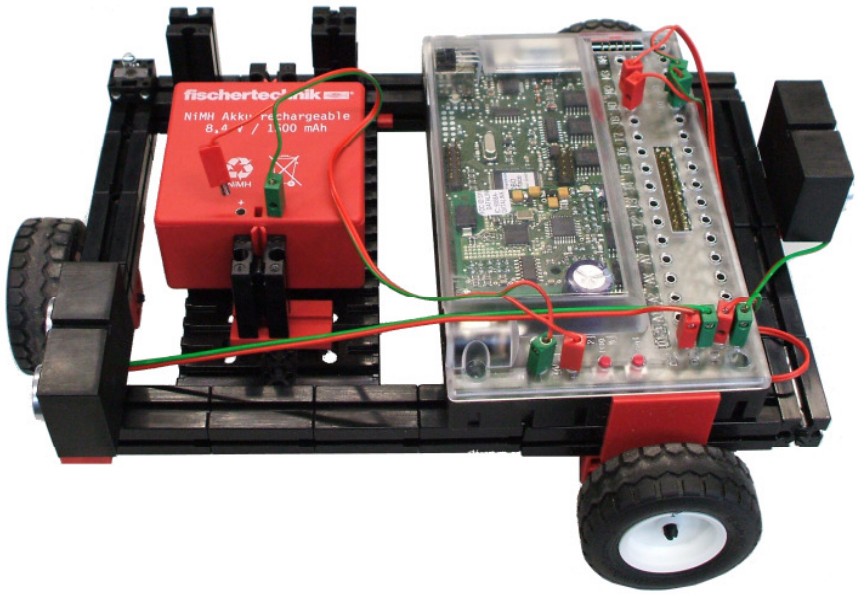

Abbildung 2-5: Der aus Teilen des *ROBO Mobile Set* von *fischertechnik*® gebaute Roboter

Der Roboter ist mit drei Rädern ausgestattet. Zwei davon werden direkt von je einem Motor unabhängig voneinander angetrieben. Das dritte Rad befindet sich an der Vorderseite quer zur Fahrtrichtung. Fährt der Roboter vorwärts oder rückwärts, steht dieses Rad still und rutscht seitlich. Im Gegensatz zum *Robotino*® verfügt dieser Roboter nicht über einen omnidirektionalen Antrieb und kann daher auch nicht seitlich fahren. Er verhält sich stattdessen wie ein Auto. Um abzubiegen müssen die beiden Antriebsräder mit unterschiedlichen Geschwindigkeiten angetrieben werden. Um sich zu drehen, müssen beide Antriebsräder mit derselben Geschwindigkeit jedoch mit entgegengesetzten Drehrichtungen angetrieben werden.
An den Roboter sind zwei Abstandssensoren, die mit Ultraschall arbeiten, montiert. Diese werden momentan nicht verwendet.

2.2.1 Kommunikation mit dem *ROBO Interface*

Das *ROBO Interface* ist mit einer 26-poligen Stiftleiste ausgestattet, auf die ein 26-poliger Pfostenstecker mit Crimpanschluss für ein Flachbandkabel aufgesteckt werden kann. An diesem Stecker sind alle Ein- und Ausgänge des *ROBO Interface* verfügbar. Die komplette Hardware, die für die Verbindung des *ROBO Interface* mit der Kommunikationsplatine erforderlich ist, wird in Kapitel 6.6 beschrieben.
Die Befehle werden von der Kommunikationsplatine zum *ROBO Interface* über eine parallele Verbindung mit vier Datenleitungen übertragen. Dabei wird der aktuelle Befehl vom

Mikroprozessor der Kommunikationsplatine an die vier Datenleitungen angelegt und somit an den vier verwendeten digitalen Eingängen des *ROBO Interface* empfangen. Solange ein Befehl an dieser Schnittstelle anliegt, soll dieser ausgeführt werden. Es handelt sich hierbei um eine unidirektionale Verbindung. Eine Antwort vom *ROBO Interface* ist nicht vorgesehen. Die Kodierung der Befehle, die der folgenden Tabelle entnommen werden kann, muss auf beiden Seiten implementiert sein, sowohl im Mikroprozessor der Kommunikationsplatine als auch im *ROBO Interface*.

Wert		Bedeutung / Funktion	In *const.h* definierte Konstante
dezimal	binär		
0	0000	halt	RI_STOP
1	0001	vorwärts	RI_FORWARD
2	0010	rückwärts	RI_BACKWARD
3	0011	links	RI_LEFT
4	0100	rechts	RI_RIGHT
5	0101	drehe (auf der Stelle)	RI_ROTATE

Tabelle 2-3: Kodierung der Befehle für die Steuerung des *ROBO Interface*

2.2.2 Software für das *ROBO Interface*

Das *ROBO Interface* muss programmiert werden. Dieses geschieht über eine speziell für Kinder und Jugendliche ausgelegte „Programmiersprache“ mit einer Demoversion der *ROBO Pro grafische Programmiersprache*.

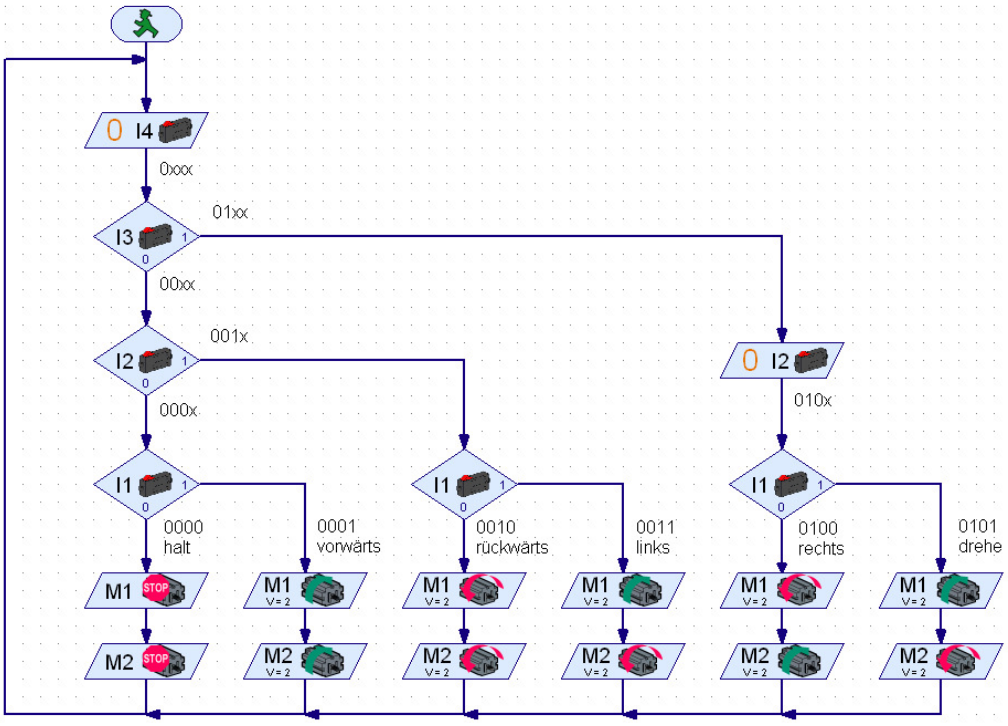

Abbildung 2-6: Das grafische Programm für das *ROBO Interface*

In diesem Programm werden die vier für die Kommunikation verwendeten digitalen Eingänge laufend abgefragt und die Geschwindigkeiten sowie die Drehrichtungen der beiden Motoren entsprechend eingestellt. Das Programm liegt der Masterarbeit auf CD-ROM bei (siehe 12.4).

3 Konzept des Gesamtsystems

In diesem Kapitel wird das Konzept des Gesamtsystems, bestehend aus den folgenden vier Hauptkomponenten, beschrieben:
- Mikrofon mit einem passenden Mikrofonverstärker
- DSP-Modul (*D.Module.C6713* mit *D.Module.PCM3003*, beide von *D.SignT GmbH & Co. KG*)
- Kommunikationsplatine (Eigenentwicklung)
- Roboter (*Robotino*® oder *fischertechnik*® *ROBO Mobile Set*)

Da der *Robotino*® zum Ansteuern der Motortreiberplatine ein zeitkritisches Protokoll verwendet (siehe 2.1.3) und der DSP mit der Spracherkennung bereits ausgelastet ist, soll dieser nicht auch noch durch die Überwachung und Aufrechterhaltung der Kommunikation und die Steuerung des *Robotino*® belastet werden.
Der Roboter von *fischertechnik*® wird über vier Datenleitungen angesteuert. Auch diese Steuerungsaufgabe soll nicht vom DSP übernommen werden. Aus diesem Grunde ist eine zusätzliche Platine vorgesehen, die Kommunikationsplatine. An die Kommunikationsplatine soll ein Display angeschlossen werden, um Statusinformationen des DSPs auszugeben. Des Weiteren sollen hierdurch auch bei einem möglichen Abbruch der Kommunikation mit dem *Robotino*® die vom DSP erkannten Worte angezeigt werden.

Zusammengesetzt ergibt sich somit folgendes System:

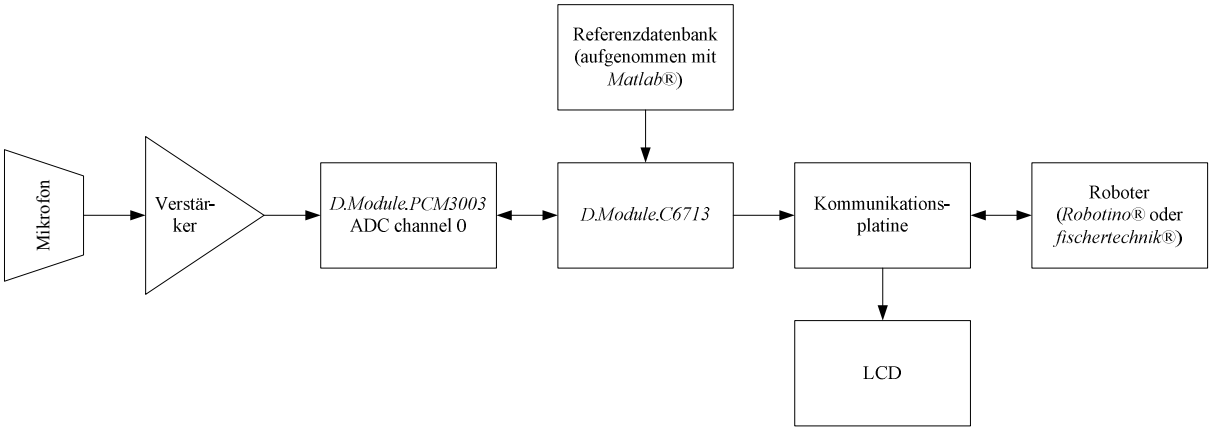

Abbildung 3-1: Überblick über das Gesamtsystem

Als Mikrofon kommt eine kleine Platine zum Einsatz, auf der sowohl eine Mikrofonkapsel als auch ein kleiner Mikrofonverstärker untergebracht sind. Dieser benötigt eine Versorgungsspannung von ca. 3,3V, welche direkt von der Kommunikationsplatine abgegriffen werden kann.

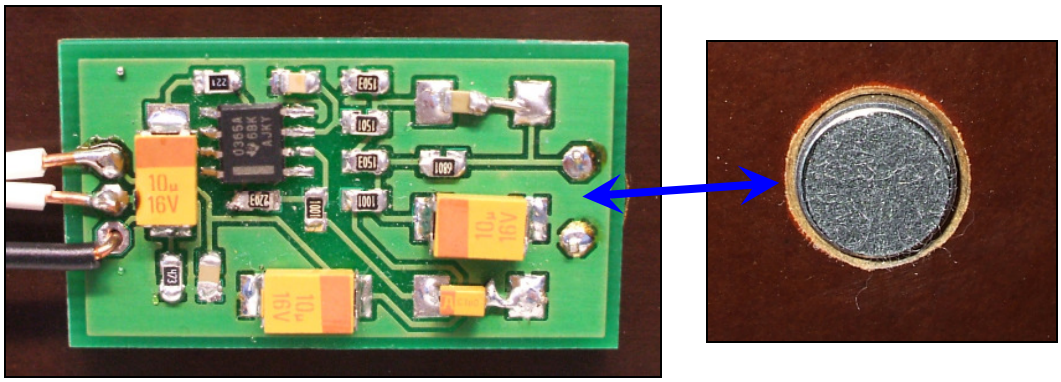

Abbildung 3-2: Mikrofonverstärker und montierte Mikrofonkapsel

Das verstärkte Mikrofonsignal wird direkt an den Eingangskanal 0 eines *PCM3003*-Codecs geführt, welcher auf dem *D.Module.PCM3003* vierfach eingesetzt wird [6]. Das Modul ist für mögliche spätere Erweiterungen ausgewählt und verfügt über acht Ein- und Ausgänge für Audiosignale. Es arbeitet mit einer festen Abtastfrequenz von momentan 8 KHz.

Auf dem *D.Module.C6713* arbeitet ein *TMS320C6713* des Herstellers *Texas Instruments*, der mit einer Taktfrequenz von 300 MHz arbeitet [7]. Dieser kommuniziert mit dem *D.Module.PCM3003* über eine I^2S-Schnittstelle und empfängt somit das digitalisierte, verstärkte Mikrofonsignal. Der DSP übernimmt die komplette Vorverarbeitung und die Spracherkennung.
Die Referenzdatenbank wird nicht mit dem DSP aufgenommen. Dieses geschieht mit dem Simulationsmodell der Spracherkennung in *Matlab*® (siehe Kapitel 4.5). Anschließend wird die Referenzdatenbank in eine *txt*-Datei exportiert, die in den Quellcode der DSP-Firmware eingebunden wird.
Hat der DSP ein Wort erkannt, wird dieses über die RS232-Schnittstelle an die Kommunikationsplatine ausgegeben. Der DSP überträgt für jede Information lediglich ein einzelnes Byte, um seine Rechenleistung möglichst komplett der Sprachverarbeitung und -erkennung zukommen zu lassen.

Die Kommunikationsplatine sorgt für eine ununterbrochene Kommunikation mit dem *Robotino*®. Wird ein Byte vom DSP empfangen, wertet der auf der Kommunikationsplatine eingesetzte Mikroprozessor dieses Byte aus und gibt bei Bedarf eine Mitteilung auf dem an ihn angeschlossenen LCD aus. Handelt es sich bei dem Byte um ein erkanntes Wort, so berechnet der Mikrocontroller die Geschwindigkeiten der drei Motoren des *Robotino*® neu und setzt die neuen Werte fortan bei der Kommunikation mit dem *Robotino*® ein. Sollte eine der Lichtschranken ein Hindernis signalisieren, sorgt der Mikroprozessor dafür, dass der Roboter stehen bleibt. Neben der aufwendigen Kommunikation mit dem *Robotino*® setzt der Mikroprozessor das erkannte Wort entsprechend codiert an den vier Datenleitungen des *ROBO Interface* an.

4 Entwicklung eines Simulationsmodells mit *Matlab*®

In diesem Kapitel wird die Theorie der Spracherkennung erläutert und mit Simulationen untersucht. Parallel dazu wird ein komplettes Simulationsmodell in *Matlab*® entwickelt. Dieses Modell wird optimiert und ausgiebig getestet.
Testergebnisse der einzelnen Abschnitte und Zwischensimulationen werden direkt in dem entsprechenden Unterkapitel aufgeführt. Das komplette Simulationsmodell wird in Kapitel 5 ausgiebig getestet.
Da das Simulationsmodell sowohl unter der Version 6 als auch unter der Version 7 von *Matlab*® funktionsfähig sein soll, müssen für beide Versionen separate *mat*-Dateien angelegt werden. Die Versionsnummer steht dabei im Dateinamen durch einen Tiefstrich abgetrennt an letzter Stelle vor dem Punkt der Dateiendung.

4.1 Auswahl des Verfahrens

Es stehen zwei verschiedene Algorithmen für die Spracherkennung zur Auswahl:
1. Das Hidden-Markov-Modell HMM und
2. die dynamische Zeitverzerrung DTW (engl.: Dynamic Time Warping).

Die Vorverarbeitung und die Merkmalsextraktion sind bei beiden Verfahren identisch. Es handelt sich hierbei lediglich um verschiedene Vorgehensweisen bei der Referenzablage sowie bei der eigentlichen Erkennung. Beide Algorithmen können auf einem DSP implementiert werden.

	HMM	DTW
Training	aufwändige Optimierung	Abspeichern der Referenzen
Erkennung	höchste Wahrscheinlichkeit	geringster Abstand
Bewertung	Vergleich mit einem Modell	Vergleich mit N Referenzen und Kombination der Einzelwerte
Aufwand Training	hoch	niedrig
Aufwand Erkennung	mittel (linear mit Anzahl der Modelle)	niedrig (linear mit Anzahl der Referenzen)
Modellierung	benötigt mehr Trainingsdaten, kompakte Repräsentation in einem Modell	kann mit sehr wenigen Referenzen arbeiten, Generalisierung schwierig
Erweiterung	einfache Kombination von kleineren Einheiten	Kombination zu größeren Einheiten schwierig

Tabelle 4-1: Vergleich zwischen HMM und DTW [8]

Das HMM ist für größere Wortschätze und – je nach Implementierung – begrenzt für kontinuierliche Spracherkennung geeignet. Es eignet sich auch für die Erkennung zusammengesetzter Wörter, die aus mehreren bekannten Wörtern bestehen. Aber auch die Erkennung anhand aneinander gereihter Phoneme ist möglich.
Das DTW kann hingegen nur für kleinere Wortschätze und Einzelworterkennung sinnvoll eingesetzt werden, da für jedes einzelne Wort eine Referenz angelegt und während der Erkennungsphase verglichen werden muss. Mit wachsenden Wortschätzen steigt die benötigte Zeit für die Erkennung linear an. DTW ist aber wesentlich einfacher zu implementieren als HMM.

Da die Aufgabenstellung einen kleinen Wortschatz und eine Einzelworterkennung erfordert, wird der DTW-Algorithmus ausgewählt. Eine genauere Beschreibung erfolgt in Kapitel 4.6. Das HMM wird, da es nicht zur Anwendung kommt, nicht weiter beschrieben.

4.2 Gesamtmodell der Spracherkennung

Für das Anlegen einer Referenzdatenbank mit Sprachmustern von einem oder mehreren Sprechern und der eigentlichen Spracherkennung werden zwei verschiedene Simulationsmodelle entworfen. Die Vorverarbeitung ist bei beiden identisch. Bei beiden in diesem Kapitel dargestellten Modellen handelt es sich um die Grundkonzepte ohne Optimierung. Optimierungen werden direkt in den einzelnen Unterkapiteln eingesetzt und erläutert.
Da mit *Matlab*® keine kontinuierliche Sprachaufnahme möglich ist, wird nur ein kurzes Intervall mit einer Dauer von zwei Sekunden aufgenommen und dieses im Anschluss untersucht. Für die Aufnahme wird die Funktion `wavrecord` verwendet. Aufgenommen wird in Mono mit einer Samplefrequenz von 8 KHz. Bei einer Aufnahmedauer von zwei Sekunden ergeben sich somit insgesamt 16000 Werte. Das Datenformat wird auf `double` eingestellt, die verwendete Auflösung beträgt 16 Bit.

Das folgende Diagramm zeigt den Aufbau des Simulationsmodells, mit welchem die Referenzdatenbank angelegt und in einer *mat*-Datei gespeichert wird. Diese wird später während der eigentlichen Spracherkennung importiert.

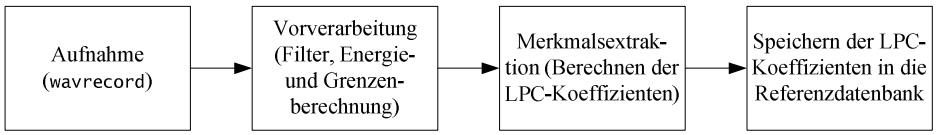

Abbildung 4-1: Simulationsmodell für das Anlegen der Referenzdatenbank

Das folgende Diagramm zeigt den Aufbau des Modells für die Simulation der Spracherkennung. Die Referenzdatenbank wird für das DTW aus der entsprechenden, vorher angelegten *mat*-Datei importiert.

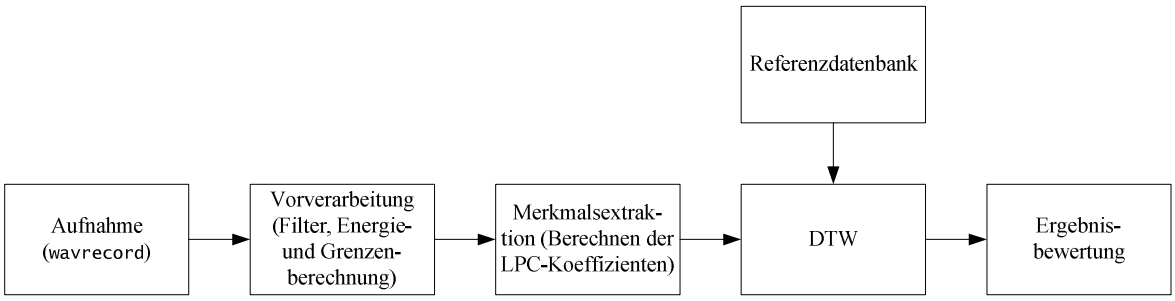

Abbildung 4-2: Simulationsmodell der Spracherkennung

Die einzelnen Verarbeitungsstufen werden im weiteren Verlauf des Kapitels genauer erläutert und analysiert.
Im Unterkapitel 4.9 werden die Bedienung und der Ablauf der beiden Simulationsmodelle dargestellt und erläutert.

4.3 Vorverarbeitung

Die Vorverarbeitung ist eine sehr wichtige Phase der Spracherkennung. Ein ungünstig gewähltes Filter, eine falsche Blockgröße oder eine unzureichende Überlappung der Blöcke können verheerende Folgen haben und die Zuverlässigkeit des ganzen Spracherkennungsmodells stark beeinflussen oder komplett unmöglich machen. Daher ist hier besondere Sorgfalt gefordert. Auch muss bei einem nicht gut funktionierenden Modell kontrolliert werden, ob die Qualität der Vorverarbeitung optimal ist.
In den verschiedenen Stufen der Vorverarbeitung, die das Sprachsignal der Reihe nach durchläuft, wird die Sprache gefiltert und auf ein enthaltenes Wort hin untersucht.

Abbildung 4-3: Ablauf der Vorverarbeitung

4.3.1 Bandpassfilter

Die erste Verarbeitungsstufe, die die digitalisierten Daten durchlaufen, ist ein Filter. Als Filtertyp wird ein FIR-Filter mit Bandpass-Charakteristik gewählt.
Die Grundfrequenz der normalen Sprache liegt bei Frauen bei ca. 180 Hz und bei Männern bei ca. 100 Hz [8]. Die Grundfrequenz soll nicht zu stark gedämpft werden. Auf der anderen Seite sollen tiefe Frequenzen unterhalb von 50 Hz möglichst keinen Einfluss auf die Spracherkennung haben. Solche Frequenzen können z. B. durch Fahrgeräusche des Roboters und die dadurch hervorgerufenen Vibrationen entstehen, aber auch durch Trittschall oder andere Störquellen. Da jedoch mit zunehmender Steilheit des Übergangsbereiches die Filterordnung und der damit verbundene Rechenaufwand schnell ansteigt, ist ein Kompromiss, der sowohl tieffrequente Störgeräusche gut unterdrückt als auch die Grundfrequenzen der Sprache nicht zu stark dämpft, zu finden.
Der obere Übergangsbereich des Bandpassfilters ist weniger kritisch, da dieser nicht so steil verlaufen muss. Er dient nur als großzügiges Anti-Alias-Filter.
Da *Matlab*® beim Design eines Bandpassfilters nur bei nahezu gleichen Breiten der Übergangsbereiche vernünftige Filter ohne starke Überschwingungen im Amplitudengang berechnen kann, wird der obere Übergangsbereich entsprechend angepasst.
Die obere Grenzfrequenz wird empirisch ermittelt und letztendlich auf 3,15 KHz festgelegt. Alle gewählten Parameter für das Design des Bandpass-Filters sind in der folgenden Tabelle zusammengefasst.

Bezeichnung	Wert
1. Stoppfrequenz	50 Hz
1. Grenzfrequenz	150 Hz
2. Grenzfrequenz	3,15 KHz
2. Stoppfrequenz	3,3 KHz
Abtastfrequenz	8 KHz
Rippel im Durchlassbereich	40 dB
Dämpfung im Sperrbereich	40 dB

Tabelle 4-2: Parameter für das Bandpassfilter

Das Filterdesign erfolgt mit der *Matlab*®-Funktion `remezord` und `remez`. `remezord` ermittelt eine Filterordnung von 156. Diese wird um zwei nach oben korrigiert, um eine Sperrdämpfung von 40 dB zu gewährleisten. Das so designte Bandpassfilter hat jedoch bei ca. 3,25 KHz einen Sprung im Amplitudengang. Um diesen Sprung zu entfernen, aber nach wie vor eine Sperrdämpfung von mindestens 40 dB zu erhalten, wird die Filterordnung experimentell erhöht. Bei einer Filterordnung von 170 entspricht das Bandpassfilter den in Tabelle 4-2 aufgestellten Bedingungen.
Das beschriebene Filterdesign wird mit der Datei *bandgap.m* durchgeführt. Der Amplitudengang des Filters wird wie folgt grafisch ausgegeben:

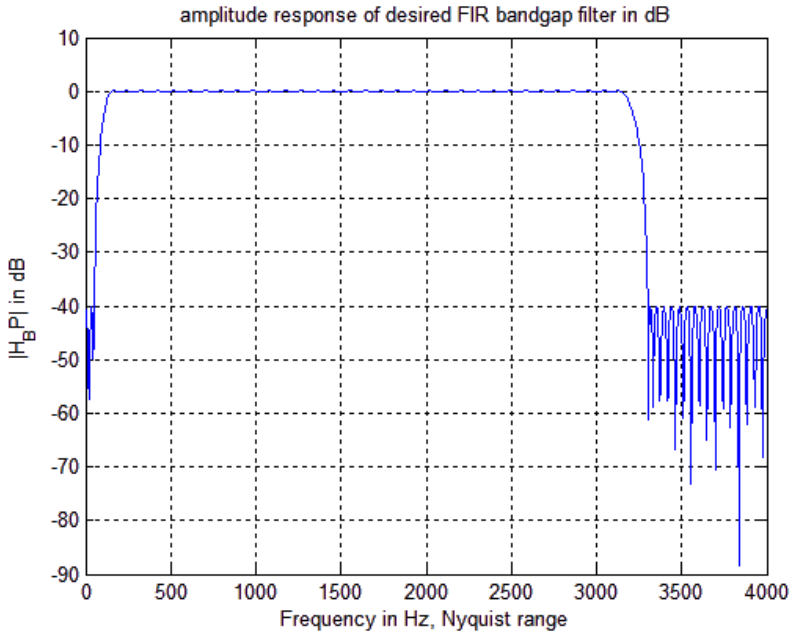

Abbildung 4-4: Amplitudengang des Bandpassfilters bei der *Matlab*®-Simulation

Die Filterkoeffizienten werden im `double`-Format in eine *mat*-Datei gespeichert (*bandgap_coefficients_6.mat* bzw. *bandgap_coefficients_7.mat*). Um eine direkte Implementierung der Spracherkennung auf dem DSP zu ermöglichen, werden die Filterkoeffizienten im `short`-Format in die Header-Datei *bandgap.h* exportiert.

Zu Beginn der beiden Simulationsmodelle werden die Filterkoeffizienten aus der *mat*-Datei importiert. Das Filter wird in die Vorverarbeitung mit der in *Matlab*® enthaltenen Funktion `filter` eingebaut.

4.3.2 Blockbildung

Die Blockbildung ist keine separate Verarbeitungsstufe. Um Zeit zu sparen und den Quellcode des Simulationsmodells übersichtlich zu halten, werden die Blöcke in jedem Verarbeitungsschritt durch entsprechende Berechnungen des Anfangs- und Endsamples des aktuellen Blockes aus dem aufgenommenen und gefilterten Datensatz entnommen.
Die Filterung mit dem Bandpassfilter ist die einzige Verarbeitungsstufe, die nicht blockbasiert arbeitet. Hier werden alle Samples der Reihe nach gefiltert. Alle weiteren Verarbeitungsstufen arbeiten blockbasiert. Vorgegeben wird diese Arbeitsweise durch das Berechnen der LPC-Koeffizienten (siehe 4.4.1). Diese werden für jeweils einen Datenblock berechnet. Ein Block

umfasst 240 Samples. Hieraus ergibt sich bei der verwendeten Samplefrequenz von 8 KHz eine Zeitdauer von 30 ms.
Kann mit den letzten Samples am Ende des Datensatzes kein ganzer Block mehr gefüllt werden, fallen diese weg, da sie ohnehin nur für einen ganz bestimmten Sonderfall gebraucht würden. Dieser Sonderfall besteht darin, dass ganz am Ende der Aufnahme ein Wort gesprochen wird. Da dieses höchstwahrscheinlich unvollständig sein würde und die Spracherkennung somit ohnehin ein falsches oder gar kein Ergebnis liefern würde, wird dieser Sonderfall nicht berücksichtigt.
Die Bestimmung der Wortgrenzen geschieht ebenfalls blockbasiert. Um die Wortgrenzen möglichst präzise bestimmen zu können, werden die Blöcke überlappend angeordnet. Die Überlappung beträgt 80 Samples, was einer Zeitdauer von 10 ms entspricht. Die Verschiebung ist in der folgenden Grafik dargestellt:

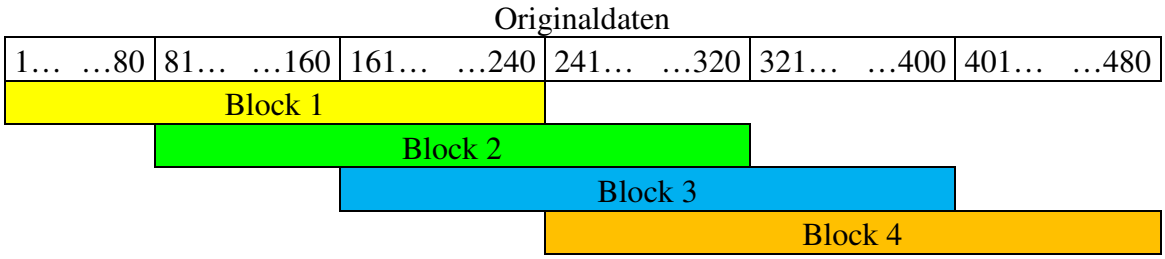

Abbildung 4-5: Blockbildung beim Simulationsmodell in *Matlab*®

In der Abbildung wird die Zählung des Feldindizes bei eins begonnen, wie es von *Matlab*® vorgegeben wird.
Jedes Sample ist somit an drei Blöcken beteiligt. Eine Ausnahme hiervon bilden die Samples am Anfang und am Ende des aufgenommenen Datensatzes.

4.3.3 Energieberechnung

Bevor eine Spracherkennung mit den aufgenommenen Daten durchgeführt werden kann, muss untersucht werden, ob der Datensatz überhaupt ein Wort enthält. Das Wort muss im Datensatz lokalisiert werden. Zusätzlich ist als Ausschlusskriterium zu prüfen, ob die Länge des erkannten Wortes die eines typischen Wortes ist.
Die Lokalisierung erfolgt über die Untersuchung der Signalenergie ([2], [8]). Hierzu wird für jeden der im vorherigen Unterkapitel beschriebenen Block ein separater Energiewert mit der folgenden Gleichung berechnet. Dabei sind *e* der berechnete Energiewert, *s(t)* die Samples des Datenblocks und *ta* bzw. *te* die Nummer des ersten und des letzten Samples.

$$e = \sum_{t=ta}^{te} s(t)^2$$

Gleichung 4-1: Berechnung der Energie eines Blocks [8]

Aus dem Verlauf der Energiewerte kann ein Wort nur sehr ungenau erkannt werden. Schwellwerte für eine automatische Erkennung der Grenzen sind praktisch nicht zu ermitteln. Um das Wort deutlicher hervorzuheben, werden die einzelnen Energiewerte der Reihe nach logarithmiert. Dabei sind *e* die einzelnen Energiewerte und e_l die logarithmierten Energiewerte.

$$e_l = \log_{10}(e)$$

Gleichung 4-2: Logarithmieren der Blockenergie [8]

In der folgenden Abbildung sind beispielhaft die zeitlichen Verläufe der Amplitude, der quadratischen Blockenergie und der logarithmierten quadratischen Blockenergie des Wortes „rückwärts“ dargestellt.

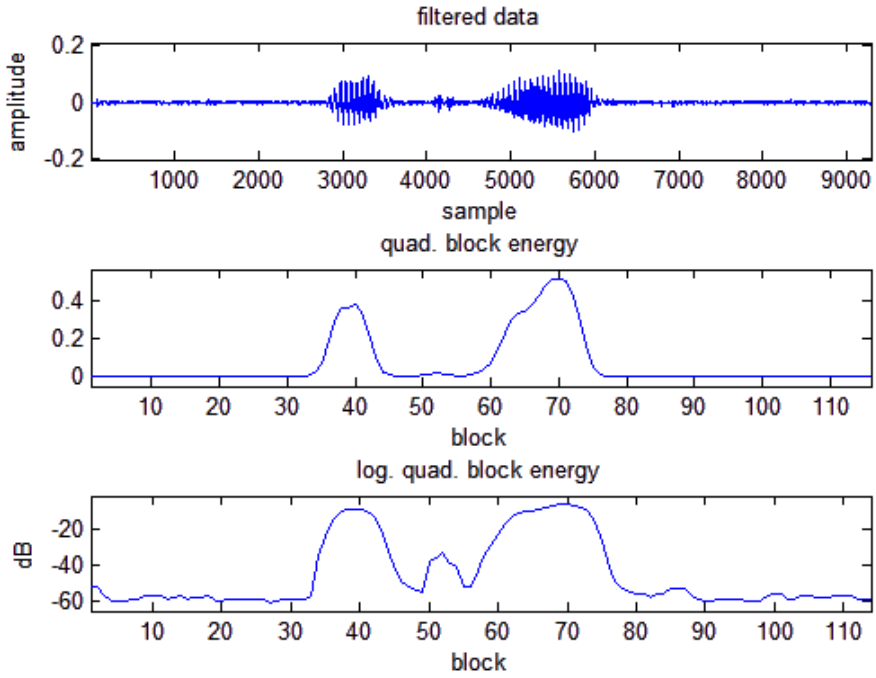

Abbildung 4-6: Amplitude und Energieverläufe des Wortes „rückwärts“

Das Wort „rückwärts“ besteht aus insgesamt vier markanten Teilen: Dem stimmhaften „rü“, einer stimmlosen Pause mit dem „ck“, einem stimmhaften „wär“ und einem stimmlosen Schluss mit „ts“.
Die beiden stimmhaften Wortteile sind in der oberen Grafik, die den zeitlichen Verlauf der aufgenommenen Samples zeigt, deutlich erkennbar. Das „ck“ ist als kleine Erhebung ebenfalls erkennbar, während das „ts“ am Wortende sich kaum von dem Umgebungsrauschen abhebt.
Der Verlauf der quadratischen Energie in der mittleren Grafik lässt eher darauf schließen, dass es sich um zwei getrennte Wörter handelt als um ein aus mehreren Silben bestehendes Wort. Der stimmlose Abschnitt, in dem das „ck“ gesprochen wird, hebt sich nur wenig von der Ruhephase (Umgebungsgeräusche) ab. Das stimmlose „ts“ am Wortende ist nicht erkennbar. Schwellwerte für eine automatische Erkennung der Grenzen sind nicht zuverlässig ermittelbar.
Aus dem Verlauf der logarithmierten quadratischen Energie in der unteren Grafik ist das zusammengesetzte Wort deutlich erkennbar. Das „ck“ in dem stimmlosen Abschnitt in der Wortmitte ist ebenfalls sehr gut sichtbar. Nur das „ts“ am Wortende, das sich bereits im zeitlichen Verlauf der aufgenommenen Samples kaum vom Umgebungsrauschen abhebt, unterscheidet sich auch hier kaum von der Ruhephase. Für die Einzelworterkennung mit einem kleinen Wortschatz kann dieses jedoch vernachlässigt werden.
Für die Erkennung der Wortgrenzen zur Isolierung des Wortes wird die logarithmierte quadratische Energie verwendet, da hier die benötigten Schwellwerte für eine automatische Erkennung der Grenzen sehr gut bestimmt werden können.

4.3.4 Isolierung des Wortes

Die theoretische Grundidee für die Isolierung des Wortes wird aus [2] übernommen. Allerdings wird das *Matlab*®-Skript nicht übernommen, sondern wegen der vielen Modifikationen und Anpassungen komplett neu programmiert.
Verwendet wird hier die unter Kapitel 4.3.3 beschriebene logarithmierte quadratische Energie.

Für die Beschreibung der Vorgehensweise werden die in *Matlab*® verwendeten Variablennamen verwendet:
- `status`: aktueller Status der Wortgrenzenerkennung
- `n`: Laufindex für die Blocknummerierung
- `logarithmical_energy(n)`: Verlauf der logarithmierten quadratischen Energie
- `start_of_word_block`: Nummer des ersten Blockes des gefundenen Wortes
- `end_of_word_block`: Nummer des letzten Blockes des gefundenen Wortes
- `gap_in_1_start`: Nummer des ersten Blockes der Energielücke (Wortpause) in Zone 1
- `gap_in_2_start`: Nummer des ersten Blockes der Energielücke (Wortpause) in Zone 2
- `upper_threshold`: oberer Schwellwert
- `lower_threshold`: unterer Schwellwert
- `maximum_slope_length`: maximale Blockanzahl für die Anstiegszeit der Energie von Zone 1 bis Zone 3
- `maximum_break_length`: maximale Blockanzahl der Energielücke (Wortpause)
- `minimum_word_length`: minimale Blockanzahl eines Wortes

Die Erkennung der Wortgrenzen geschieht mit Hilfe eines Zustandsautomaten mit fünf Zuständen. Dieser Zustandsautomat ist in der folgenden Grafik dargestellt. Gezeigt werden nur die Bedingungen, bei denen ein Zustand in einen anderen übergeht. Werden nicht alle Bedingungen für den Wechsel vom aktuellen Zustand in einen anderen Zustand erfüllt, bleibt der aktuelle Zustand bestehen.
Aus Übersichtsgründen werden die genauen Inhalte der einzelnen Zustände nicht mit abgebildet. Es werden die in *Matlab*® verwendeten Variablen- und Zustandsnamen übernommen.

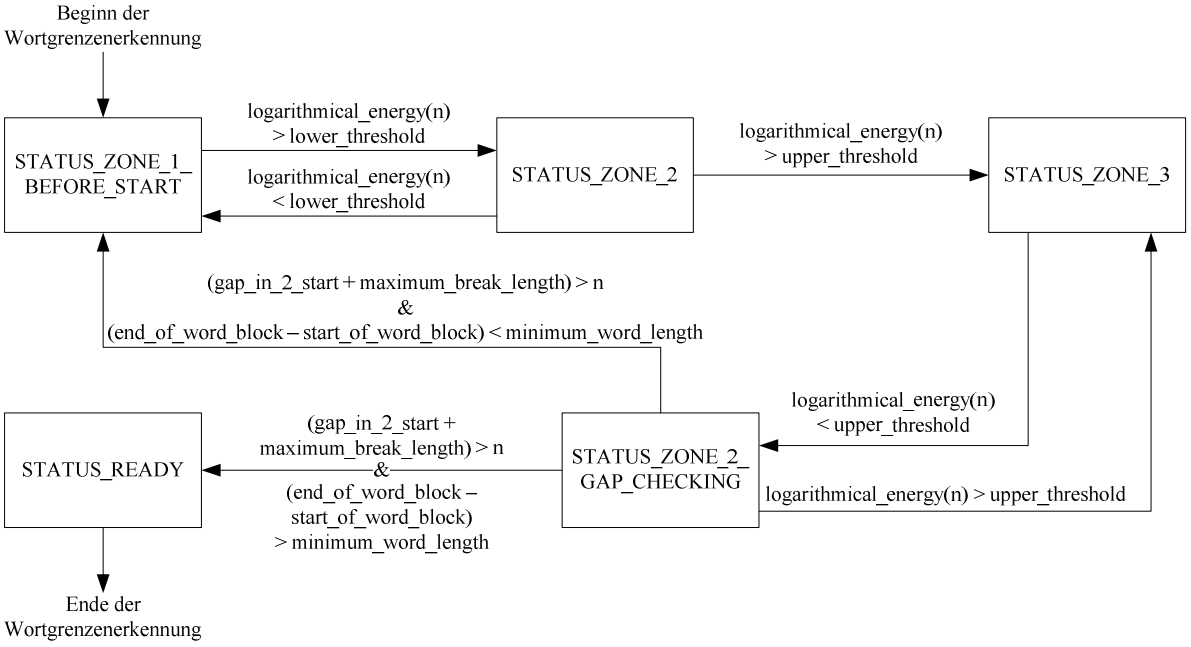

Abbildung 4-7: Zustandsautomat der Wortgrenzenerkennung

Die folgende Grafik zeigt den Verlauf der Energie des Wortes „rückwärts“. Die Grafik wird während der Spracherkennung mit *Matlab*® automatisch generiert und angezeigt. Die beiden Schwellwerte und die beiden erkannten Wortgrenzen werden ebenfalls mit in die Grafik eingezeichnet.

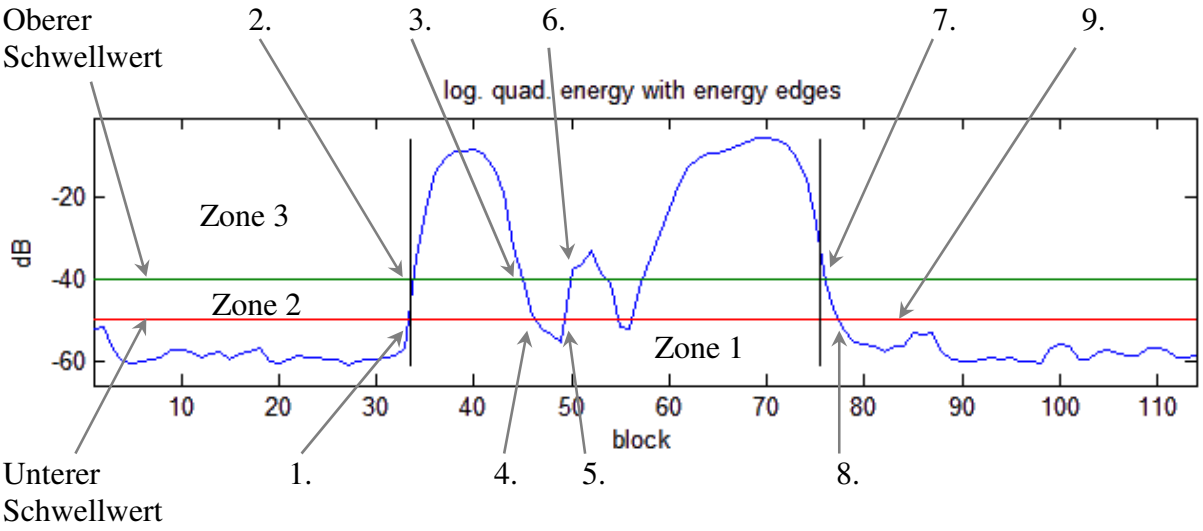

Abbildung 4-8: Funktionsweise der Wortgrenzenerkennung am Beispiel „rückwärts“

Durch die beiden Schwellwerte wird die Grafik in drei Zonen eingeteilt:

- Zone 1 liegt unterhalb des unteren Schwellwertes. In dieser Zone befindet sich die Energie, die bei Stille (Umgebungsrauschen, sehr leise Geräusche) vorliegt.
- Zone 2 liegt zwischen dem unteren und dem oberen Schwellwert. In dieser Zone befindet sich die Energie, wenn der Sprecher sehr leise oder weit vom Mikrofon entfernt mit normaler Lautstärke spricht. Auch Atemgeräusche haben eine Energie, die in diese Zone vordringen kann.
- Zone 3 liegt oberhalb des oberen Schwellwertes. In diesem Bereich liegt die Energie von normal gesprochenen Wörtern. Auch lautere Störgeräusche dringen in diese Zone ein.

Die beiden Schwellwerte für die Bestimmung der Wortgrenzen werden empirisch ermittelt und fest eingestellt. Sie werden nicht automatisch angepasst, da dieses auch bei der späteren Implementierung auf einem DSP durch die kontinuierliche Aufnahme problematisch werden kann. Der untere Schwellwert wird auf einen Wert von -50 dB eingestellt. Dieser ist in der *Matlab*®-Grafik durch eine rote Linie dargestellt. Der obere Schwellwert wird auf einen Wert von -40 dB eingestellt. Dieser ist in der *Matlab*®-Grafik durch eine grüne Linie dargestellt.

Die Funktion der Wortgrenzenerkennung wird anhand der Abbildung 4-8 erläutert. Die Nummern bezeichnen die markanten Stellen des Energieverlaufes. Bei diesem Wort handelt es sich um ein normal gesprochenes Wort ohne besondere Störungen.
Zu Beginn der Wortgrenzenerkennung muss die Energie in Zone 1 (Stille) liegen. Ist dies nicht der Fall, wird die Suche mit einer Fehlermeldung abgebrochen. Der aktuelle Status ist `STATUS_ZONE_1_BEFORE_START`. Der Laufindex `n` wird hochgezählt.

1. Die Energie `logarithmical_energy(n)` wird erstmals größer als `lower_threshold`. Die aktuelle Blocknummer `n` wird als möglicher Startblock des Wortes in `start_of_word_block` gespeichert. Der Zustandsautomat geht in den Status `STATUS_ZONE_2` über.

2. Die Energie `logarithmical_energy(n)` wird größer als `upper_threshold`. Anhand der aktuellen Blocknummer `n` wird geprüft, ob die Anstiegsdauer nicht zu lange gedauert hat. Hierzu wird geprüft, ob `start_of_word_block + maximum_slope_length < n` ist. Dies ist der Fall, `start_of_word_block` bleibt unverändert und gilt als Startblock des Wortes. Der Zustandsautomat geht in den Status `STATUS_ZONE_3` über.

3. Die Energie `logarithmical_energy(n)` fällt unter `upper_threshold`. Die aktuelle Blocknummer wird in `gap_in_2_start` als möglicher Endblock des Wortes gespeichert. Der Zustandsautomat geht in den Status `STATUS_ZONE_2_GAP_CHECKING` über.

4. Während des Zustandes `STATUS_ZONE_2_GAP_CHECKING` fällt die Energie `logarithmical_energy(n)` unter `lower_threshold`. Die in diesem Zustand laufende Prüfung der bisherigen Länge der Energielücke (Wortpause) mithilfe des Blockindizes `n` durch `gap_in_2_start + maximum_break_length < n` ergibt, dass es sich um eine normale Pause zwischen verschiedenen Teilen eines Wortes handelt. Der Zustand bleibt deshalb unverändert.

5. Während des Zustandes `STATUS_ZONE_2_GAP_CHECKING` steigt die Energie `logarithmical_energy(n)` erneut über `lower_threshold`.
   Die laufende Prüfung der Energielücke (siehe 4.) ergibt, dass es sich um eine normale Pause zwischen verschiedenen Teilen eines Wortes handelt. Der Zustand bleibt deshalb unverändert.

6. Die Energie `logarithmical_energy(n)` wird größer als `upper_threshold`. Die laufende Prüfung der Energielücke (siehe 4.) ergibt, dass die maximale Länge der Pause noch nicht erreicht wurde. `gap_in_2_start` wird zurückgesetzt. Der Zustandsautomat geht in den Status `STATUS_ZONE_3` über.

Zwischen 6 und 7 wiederholt sich der Ablauf von 3 bis 6 bei einer zweiten Lücke im Energieverlauf.

7. Die Energie `logarithmical_energy(n)` fällt unter `upper_threshold`. Die aktuelle Blocknummer wird in `gap_in_2_start` als möglicher Endblock des Wortes gespeichert. Der Zustandsautomat geht in den Status `STATUS_ZONE_2_GAP_CHECKING` über.

8. Während des Zustandes `STATUS_ZONE_2_GAP_CHECKING` fällt die Energie `logarithmical_energy(n)` unter `lower_threshold`. Die laufende Prüfung der Energielücke (siehe 4.) ergibt, dass es sich um eine normale Pause zwischen verschiedenen Teilen eines Wortes handelt. Der Zustand bleibt deshalb unverändert.

9. Die Energie steigt während des Zustandes `STATUS_ZONE_2_GAP_CHECKING` nicht erneut über `upper_threshold`. Die laufende Prüfung der Energielücke (siehe 4.) ergibt, dass die maximale Pausenlänge erreicht wird. `gap_in_2_start - 1` wird als `end_of_word_block` gespeichert. Die Wortgrenzen gelten vorerst als erkannt. Abschließend wird die Länge des Wortes mit `(end_of_word_block - start_of_word_block) < minimum_word_length` geprüft. Ist das Wort lang genug, geht der Zustand in `STATUS_READY` über, die Wortgrenzen sind somit erkannt. Die weiteren Schritte der Spracherkennung können nun durchgeführt werden.

An drei Stellen kann ein abweichender Ablauf der Wortgrenzenerkennung eintreten. Dieses beruht entweder auf einem längeren, weniger lauten Störgeräusch (z. B. knisterndes Papier),

welches direkt in die Sprache übergeht, einem zu leise oder vom Mikrofon weggedreht gesprochenen Wortanfang oder einem sehr kurzen, lauteren Störgeräusch (z. B. Türknall). Diese besonderen Zustandswechsel sind in Abbildung 4-8 nicht dargestellt. Hiervon betroffen sind die markanten Stellen 2, 8 und 9.

2. Die Energie `logarithmical_energy(n)` wird größer als `upper_threshold`. Anhand der aktuellen Blocknummer `n` wird geprüft, ob die Anstiegsdauer nicht zu lang ist. Hierzu wird geprüft, ob `start_of_word_block + maximum_slope_length < n` ist. Sollte der Energieanstieg von Zone 1 bis Zone 3 durch einen sehr langsamen Anstieg der Energie zu viele Blöcke benötigt haben, wird `start_of_word_block` angepasst und auf `n - maximum_slope_length` gesetzt. Der Zustandsautomat geht in den Status `STATUS_ZONE_3` über.

8. Die Energie `logarithmical_energy(n)` fällt unter `lower_threshold` ohne `upper_threshold` jemals zuvor überschritten zu haben. Der Zustandsautomat geht in den Zustand `STATUS_ZONE_1_BEFORE_START` über. Die Wortgrenzenerkennung wird ab dem aktuellen Block neu gestartet und der Zustandsautomat geht in den Zustand `STATUS_ZONE_1_BEFORE_START` über.

9. Die Energie steigt während des Zustandes `STATUS_ZONE_2_GAP_CHECKING` nicht erneut über `upper_threshold`. Die laufende Prüfung der Energielücke (siehe 4.) ergibt, dass die maximale Pausenlänge erreicht wird. `gap_in_2_start - 1` wird als `end_of_word_block` gespeichert. Die Wortgrenzen gelten vorerst als erkannt. Als Letztes wird die Länge des Wortes mit `(end_of_word_block - start_of_word_block) < minimum_word_length` geprüft. Ist das Wort zu kurz, werden alle Variablen zurückgesetzt. Die Wortgrenzenerkennung wird ab dem aktuellen Block neu gestartet und der Zustandsautomat geht in den Zustand `STATUS_ZONE_1_BEFORE_START` über.

Eine Besonderheit der umgesetzten Wortgrenzenerkennung ist, dass der erste Block eines Wortes eine Energie haben kann, die nur unmittelbar oberhalb des unteren Schwellwertes liegt, während der letzte Block eines Wortes einen Energiewert haben muss, der sich oberhalb des oberen Schwellwertes befindet. Ausführliche Tests haben ergeben, dass durch diese Vorgehensweise die Erkennungsrate des Gesamtsystems etwas gesteigert wird.

Nur wenn in dem aufgenommenen Datenblock ein Wort mit gültiger Länge gefunden wird, kann die Spracherkennung oder Anlage der Referenzdatenbank fortgesetzt werden. Das Wort befindet sich in den Blöcken mit den Indizes `start_of_word_block` bis `end_of_word_block`.

4.4 Merkmalsextraktion

Da es nicht möglich ist, zwei Wörter direkt im Zeitbereich Sample für Sample miteinander zu vergleichen und somit eine Übereinstimmung festzustellen, wird ein anderer Ansatz verwendet. Für jeden Block des isolierten Wortes werden Merkmale berechnet, die den jeweiligen Block charakterisieren. Diese Merkmale werden dann mit den gespeicherten Merkmalen der antrainierten Wörter verglichen.
Es stehen verschiedene Arten von Merkmalen zur Verfügung:
- Autokorrelationskoeffizienten: Koeffizienten, die direkt durch eine Autokorrelation bestimmt werden. Dieser Koeffiziententyp wird in [2] verwendet.

- Linear Predictive Coding (LPC-Koeffizienten): Bei diesen Koeffizienten kommt als erster Schritt der Berechnung ebenfalls eine Autokorrelation zum Einsatz. Es folgen jedoch weitere Rechenschritte. LPC-Koeffizienten sind stabiler und zuverlässiger als die reinen Autokorrelationskoeffizienten [2].
- Cepstrum Koeffizienten: Diese Koeffizienten werden aus den zuvor bestimmten LPC-Koeffizienten berechnet. Sie sind robuster als die LPC-Koeffizienten [9].

In einem direkten Vergleichstest zwischen LPC-Koeffizienten und Cepstrum Koeffizienten hat sich keine Veränderung der Qualität der Spracherkennung gezeigt. Daher werden in dieser Arbeit die LPC-Koeffizienten verwendet und der zusätzliche Rechenaufwand für die Bestimmung der Cepstrum Koeffizienten eingespart.

4.4.1 LPC-Koeffizienten

Die lineare Prädiktion basiert auf dem Quelle-Filter-Modell ([8], [9], [10]). Bei diesem Modell wird versucht, das Filter der Signalquelle nachzubilden. Bei der Spracherkennung handelt es sich dabei um die Sprachquelle, bestehend aus dem Vokaltrakt und der Schallabstrahlung am Mund.
Der Ansatz der Berechnung der LPC-Koeffizienten besteht darin, aus einer Reihe bekannter Signalwerte den nächsten, unbekannten Signalwert im Voraus zu schätzen [10]. Hierzu wird als sogenanntes Prädiktionsfilter ein FIR-Filter mit der Ordnung $N = p\text{-}1$ eingesetzt, wobei p die Anzahl der LPC-Koeffizienten ist. Die folgende Grafik zeigt einen FIR-Prädiktor N-ter Ordnung.

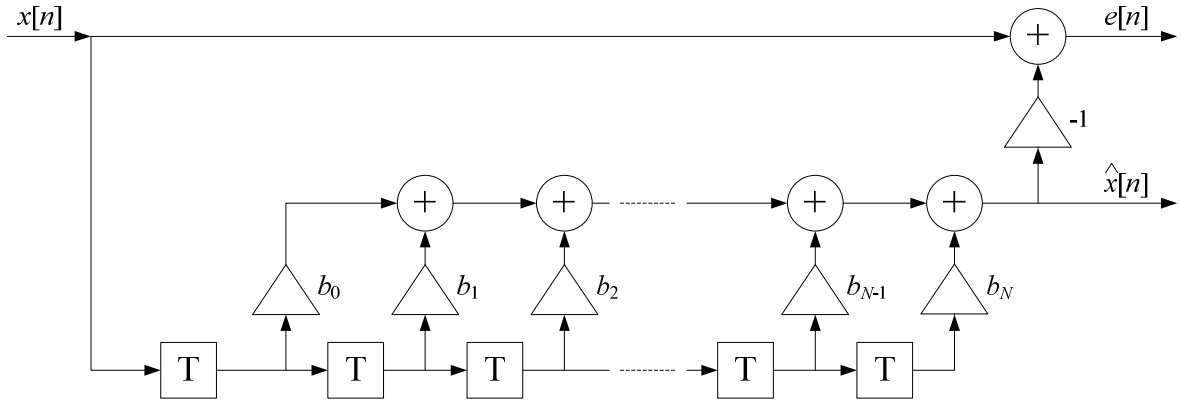

Abbildung 4-9: FIR-Prädiktor N-ter Ordnung in transversaler Struktur (nach [10])

$$\hat{x}[n] = \sum_{k=0}^{N} b_k * x[n-1-k]$$

Gleichung 4-3: Berechnung des Schätzwertes der linearen Prädiktion (nach [8], [9], [10])

Um die LPC-Koeffizienten b_k zu berechnen, wird der Prädiktionsfehler $e[n]$, der den Abstand zwischen tatsächlichem Wert $x[n]$ und geschätztem Wert $\hat{x}[n]$ angibt, eingesetzt.

$$e[n] = x[n] - \hat{x}[n] = x[n] - \sum_{k=0}^{N} b_k * x[n-1-k]$$

Gleichung 4-4: Bestimmungsgleichung für den Prädiktionsfehler (nach [8], [9], [10])

Die Berechnung der LPC-Koeffizienten für einen Datenblock benötigt die über diesen Datenblock berechneten Koeffizienten der Autokorrelation (Kurzzeit-Autokorrelation, [8]). Diese werden mit der nachfolgenden Gleichung berechnet. Dabei ist $x[n]$ ein Datenblock mit $M = 240$ Samples und r_l die berechneten $p+1$ Koeffizienten der Autokorrelation.

$$r_l = \sum_{n=0}^{M-1-l} x[n+l] * x[n] \qquad \text{für } l = 0, \ldots, p$$

Gleichung 4-5: Berechnung der Autokorrelationskoeffizienten (nach [8], [9], [10])

Die so berechneten Autokorrelationskoeffizienten werden in das folgende Gleichungssystem eingesetzt. Die Herleitung dieses Gleichungssystems aus den Gleichungen 4-3 und 4-4 ist umfangreich und kann in [10] nachgelesen werden.

$$\begin{pmatrix} r_0 & r_1 & \cdots & r_N \\ r_1 & r_0 & & r_{N-1} \\ \vdots & & \ddots & \vdots \\ r_N & & \cdots & r_0 \end{pmatrix} * \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_N \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_{N+1} \end{pmatrix}$$

$$R * b = r$$

Gleichung 4-6: Gleichungssystem der LPC-Koeffizienten-Berechnung (nach [8], [9], [10])

Zum direkten Berechnen der LPC-Koeffizienten muss das Gleichungssystem umgestellt werden:

$$b = R^{-1} * r$$

Gleichung 4-7: Direkte Berechnung der LPC-Koeffizienten (nach [8])

In *Matlab*® kann die direkte Berechnung der LPC-Koeffizienten so einfach durchgeführt werden, da *Matlab*® die Matrixinversion und –multiplikation beherrscht.
Bei der späteren Umsetzung auf dem DSP ist diese direkte Berechnung aber nur sehr umständlich zu implementieren. Außerdem ist sie mit einem hohen Rechenaufwand verbunden. Daher wird hier ein anderer Weg zur Berechnung der LPC-Koeffizienten gewählt, der Levinson-Durbin-Algorithmus. Dieser nutzt die besonderen Eigenschaften der Toeplitzmatrix R (Gleichung 4-6) aus. Bei der rekursiven Berechnung der LPC-Koeffizienten werden vier Hilfsgrößen eingesetzt, die drei Vektoren E, k, und q sowie die Matrix *alpha*. Die Herleitung des Levinson-Durbin-Algorithmus ist umfangreich und kann in [10] nachgelesen werden. An dieser Stelle werden nur die tatsächlich verwendeten Gleichungen aufgeführt.

Die Reihenfolge der aufgeführten Gleichungen entspricht der umgesetzten Reihenfolge. Diese muss eingehalten werden.
Vor der rekursiven Berechnung müssen einige Hilfswerte separat berechnet werden:

$$alpha_{1,1} = -\frac{r_1}{r_0}$$

Gleichung 4-8: Vorabberechnung von $alpha_{1,1}$ vor der rekursiven Berechnung [10]

$$E_1 = r_0 + alpha_{1,1} * r_1$$

Gleichung 4-9: Vorabberechnung von E_1 vor der rekursiven Berechnung [10]

$$q_1 = alpha_{1,1} * r_1 + r_2$$

Gleichung 4-10: Vorabberechnung von q_1 vor der rekursiven Berechnung [10]

Der Hilfswert k_1 wird nicht benötigt und daher auch nicht berechnet. Alle Voraussetzungen für die rekursive Berechnung der LPC-Koeffizienten sind nunmehr gegeben. Die folgenden Gleichungen werden in einer großen Schleife der Reihe nach eingesetzt. Erst nachdem jede Gleichung einmal berechnet wurde, darf der Laufindex i der Schleife inkrementiert werden.
Für eine bessere Veranschaulichung wird die Reihenfolge des rekursiven Teils des Levinson-Durbin-Algorithmus' in einem Flussdiagramm dargestellt:

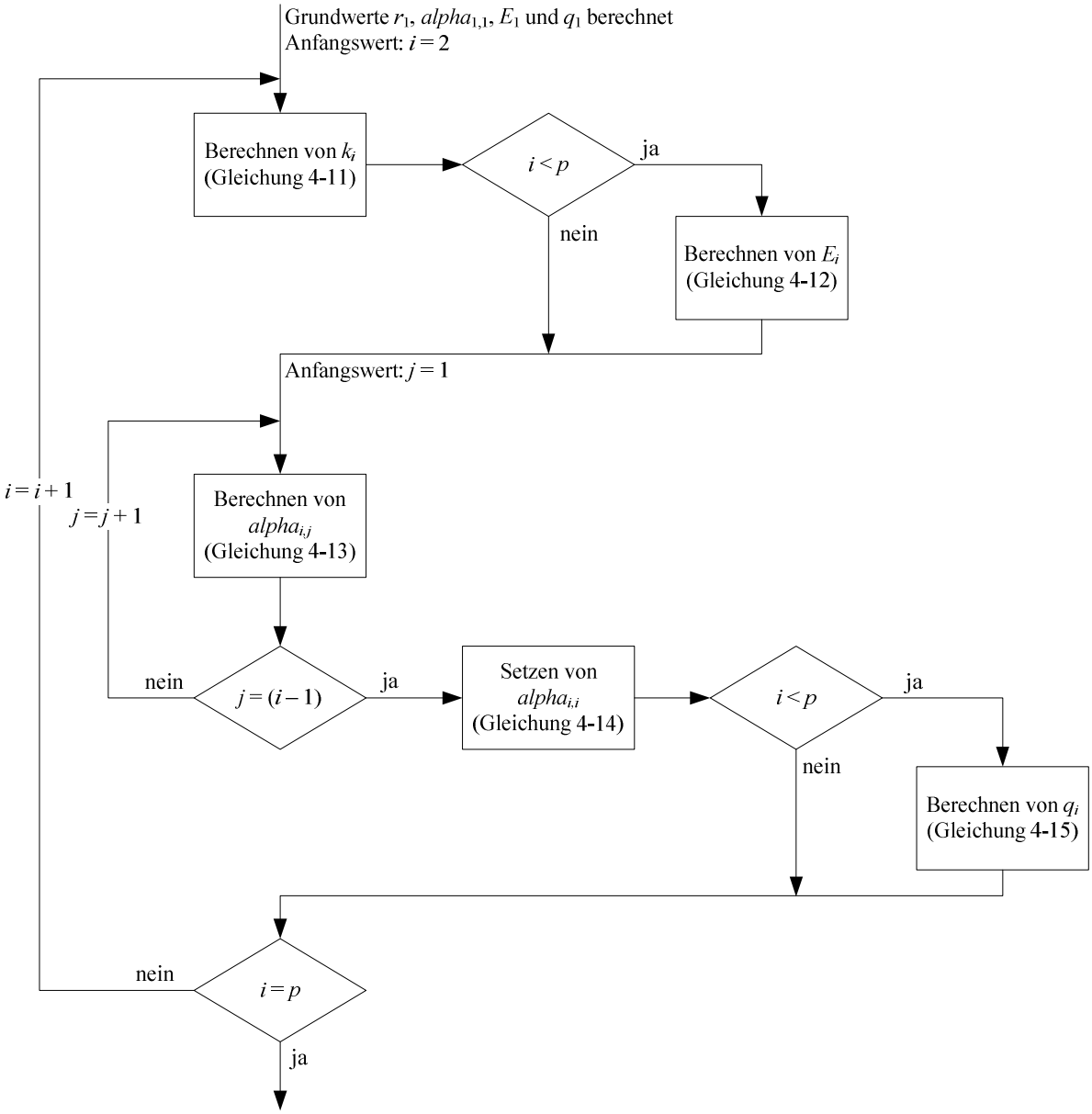

Abbildung 4-10: Rekursive Schleife des Levinson-Durbin-Algorithmus

$$k_i = -\frac{q_{i-1}}{E_{i-1}}$$

Gleichung 4-11: Berechnung von k_i innerhalb der Rekursionsschleife (nach [10])

$$E_i = E_{i-1} * \left(1 - k_i^{\,2}\right)$$

Gleichung 4-12: Berechnung von E_i innerhalb der Rekursionsschleife (nach [10])

E_i wird im letzten Durchgang der Rekursionsschleife nicht mehr berechnet.

$$alpha_{i,j} = alpha_{i-1,j} + k_i * alpha_{i-1,i-j} \qquad \text{für } j = 1, \ldots, i\text{-}1$$

Gleichung 4-13: Berechnung von $alpha_{i,j}$ innerhalb der Rekursionsschleife (nach [10])

$$alpha_{i,i} = k_i$$

Gleichung 4-14: Setzen von $alpha_{i,i}$ innerhalb der Rekursionsschleife (nach [10])

$$q_i = r_{i+1} + \sum_{j=1}^{i} alpha_{i,j} * r_{i-j+1}$$

Gleichung 4-15: Berechnung von q_i innerhalb der Rekursionsschleife (nach [10])

q_i wird im letzten Durchgang der Rekursionsschleife nicht mehr berechnet.

Nach Abschluss der Rekursionsschleife werden die berechneten Werte aus der letzten Zeile der Matrix *alpha* mit Hilfe einer Schleife in den LPC-Koeffizientenvektor *a* kopiert. Eine Ausnahme stellt der Koeffizient a_0 dar. Dieser wird immer separat auf 1 gesetzt.

$$a_i = alpha_{p,i} \qquad \text{für } i = 1, \ldots, p$$

Gleichung 4-16: Kopieren der LPC-Koeffizienten a_i aus der Matrix *alpha* (nach [10])

Die mit dem Levinson-Durbin-Algorithmus berechneten LPC-Koeffizienten a_i unterscheiden sich von den Filterkoeffizienten des Prädiktionsfilters b_k. Ein Zusammenhang in Bezug auf Abbildung 4-9 ist durch folgende Übertragungsfunktion zwischen dem Eingangswert $x[n]$ und dem Ausgangswert $e[n]$ gegeben:

$$H_e(z) = 1 - z^{-1} * \sum_{k=0}^{N} b_k * z^{-k} = \sum_{i=0}^{N+1} a_i * z^{-i} \qquad \text{mit } N = p - 1$$

Gleichung 4-17: Übertragungsfunktion für den Prädiktionsfehler in Abbildung 4-9 (nach [10])

Hieraus ergibt sich der folgende Zusammenhang:

$$a_i = -b_{i-1} \qquad \text{für } i = 1, \ldots, p$$

Gleichung 4-18: Zusammenhang zwischen a_i und b_i (nach [10])

Für die Umsetzung der Spracherkennung werden die LPC-Koeffizienten a_i verwendet. Diese Koeffizienten stimmen mit der *Matlab*®-Funktion `lpc` überein.
Da die Berechnung der LPC-Koeffizienten blockbasiert arbeitet, verändern sich die LPC-Koeffizienten bei Lautänderungen im Verlauf des Sprechens eines Wortes schnell. Pro Block werden p LPC-Koeffizienten berechnet, wobei a_0 hier nicht mitgezählt wird.
Bevor aus einem Datenblock die LPC-Koeffizienten berechnet werden, wird dieser mit dem Hanning-Fenster bearbeitet, um Störeinflüsse durch Sprungstellen zu vermeiden.
Die Anzahl p der zu berechnenden LPC-Koeffizienten wird fest auf 10 eingestellt. Dabei handelt es sich um den bei einer Samplefrequenz von 8 KHz typischen Wert [9]. Experimente mit einer größeren Anzahl führen zu keiner Verbesserung der Erkennungsrate.

4.5 Referenzdatenbank

Um ein unbekanntes Wort identifizieren zu können, muss eine Referenzdatenbank vorhanden sein, die für jedes Wort, das verstanden werden soll, mindestens einen Referenzdatensatz enthält. Zudem muss die Referenzdatenbank flexibel aufgebaut sein. Es sollen sowohl neue Wörter als auch komplette neue Benutzer antrainiert werden können.
Im Simulationsmodell mit *Matlab*® wird hierzu ein vierdimensionales Feld eingesetzt.
Die erste Dimension dieses Feldes unterteilt die verschiedenen Sprecher, die zweite die verschiedenen Wörter der Sprecher, die dritte die verschiedenen Koeffizientenblöcke der Wörter, während die vierte Dimension die einzelnen LPC-Koeffizienten der Koeffizientenblöcke unterteilt. Durch diese Strukturierung ist ein Zugriff auf jeden einzelnen LPC-Koeffizienten innerhalb der kompletten Referenzdatenbank möglich.

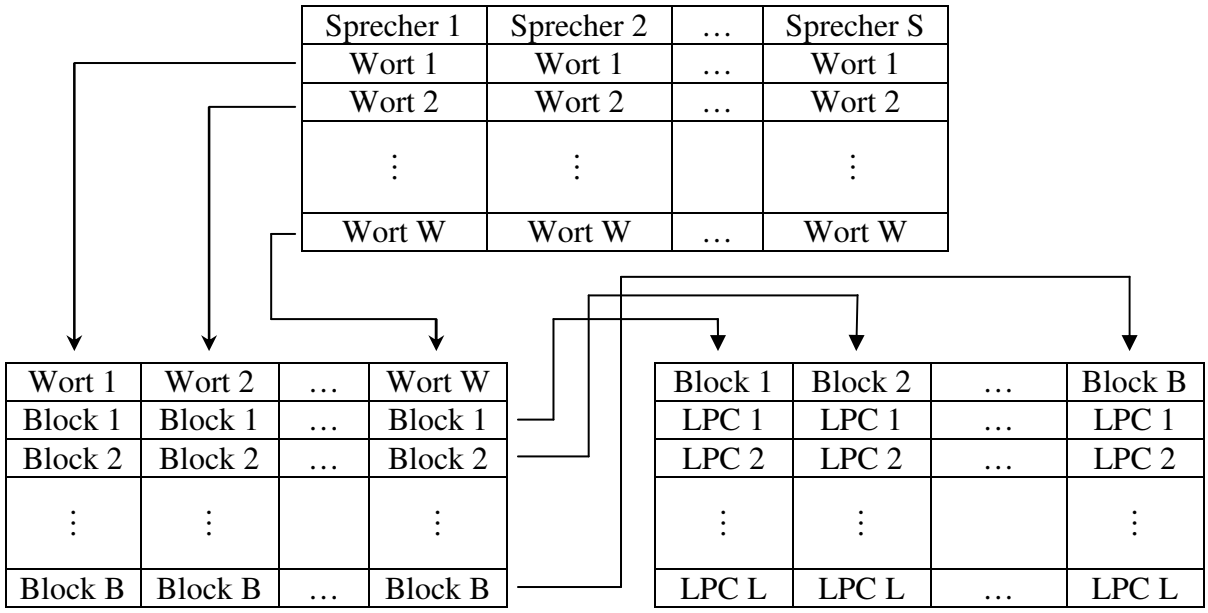

Abbildung 4-11: Strukturierung der Referenzdatenbank

Soll ein neuer Sprecher angelegt werden, wird die Datenbank in der ersten Dimension erweitert. Kommt ein neues Wort hinzu, wird sie in der zweiten Dimension erweitert. Soll einem Sprecher ein in der Datenbank bei einem anderen Sprecher bereits vorhandenes Wort antrainiert werden, muss die Datenbank nur erweitert werden, wenn das neue Wort länger als das längste Wort der Datenbank ist. Diese Erweiterung findet in der dritten Dimension statt. Wörter, die kürzer sind als das längste in der Datenbank enthaltene Wort, werden mit Nullen am Ende aufgefüllt.

Die Größe der vierten Dimension ist fest auf die Anzahl der verwendeten LPC-Koeffizienten eingestellt und kann nicht verändert werden.
Zusätzlich werden die Namen der Sprecher und die Wörter in Klartext abgespeichert.

4.6 Mustervergleich mit DTW

Um ein gesprochenes Wort zu identifizieren, muss es mit den einzelnen Wörtern der Referenzdatenbank verglichen werden. Dieser Vergleich wird nicht zwischen den einzelnen Samples des unbekannten Wortes und der einzelnen Referenzwörter durchgeführt, sondern mit deren LPC-Koeffizienten. Dabei gibt es zwei Probleme, die einen besonderen Vergleichsalgorithmus erfordern:

1. Die beiden miteinander zu vergleichenden Wörter sind unterschiedlich lang.
2. Die Länge der einzelnen Wortbestandteile (Laute) unterscheidet sich. Zum Beispiel kann das Wort „rückwärts“ in der Referenzdatenbank mit einem kurzen „ü“ und einer längeren Pause zwischen „ck“ und „w“ gespeichert sein, während das zu einem anderen Zeitpunkt gesprochene und zu identifizierende „rückwärts“ z. B. ein langes „ü“ und keine merkbare Pause zwischen „ck“ und „w“ haben kann. Daher reicht eine Anpassung der beiden Wortlängen allein nicht aus.

Die dynamische Zeitverzerrung (DTW, engl.: Dynamic Time Warping) ist ein Verfahren zum direkten Vergleichen zweier Muster, das die beiden genannten Probleme bewältigen kann. Sie bildet in dieser Arbeit das Kernstück der Spracherkennung.
Die dynamische Zeitverzerrung berechnet den Abstand zwischen zwei Wörtern und arbeitet dazu mit einem zweidimensionalen Feld. Die Anzahl N der Zeilen entspricht dabei der Anzahl der Blöcke (Länge) des Referenzwortes $x_{n,i}$, mit welchem der Vergleich durchgeführt werden soll. Die Anzahl M der Spalten entspricht der Anzahl der Blöcke (Länge) des zu identifizierenden Wortes $y_{m,i}$. Jeder Block wird durch p LPC-Koeffizienten repräsentiert, wobei nur die Koeffizienten a_1 bis a_p verwendet werden. a_0 ist immer 1 und wird nicht mit berücksichtigt. n bzw. m geben die Blocknummer und i die Nummer des LPC-Koeffizienten innerhalb des ausgewählten Blockes an.
Um nicht 10 (p) zweidimensionale Felder $D(x,y)$ berechnen zu müssen und bei der Auswertung nicht 10 (p) Ergebnisse pro Wortvergleich bewerten zu müssen, wird eine Vektor-Quantisierung durchgeführt. Hierdurch wird die Datenmenge so reduziert, dass nur ein zweidimensionales Feld $D(x,y)$ pro Vergleich zwischen zwei Wörtern berechnet werden muss. Dabei wird ein Abstandswert d zwischen einem Block des Wortes x und einem Block des Wortes y berechnet. Alle 10 (p) LPC-Koeffizienten der beiden Blöcke werden in die Vektor-Quantisierung mit einbezogen.
Es existieren zwei gebräuchliche Möglichkeiten, den Abstand d zwischen zwei Vektoren zu berechnen. Dabei handelt es sich um den Betragsabstand und den quadratischen Euklidischen Abstand:

$$d_{n,m}(x_n, y_m) = \sum_{i=1}^{p} \left| x_{n,i} - y_{m,i} \right| \qquad \text{für } n = 1, \ldots, N \text{ und } m = 1, \ldots, M$$

Gleichung 4-19: Vektor-Quantisierung mit Betragsabstand (nach [8])

$$d_{n,m}(x_n, y_m) = \sum_{i=1}^{p} \left(x_{n,i} - y_{m,i} \right)^2 \qquad \text{für } n = 1, \ldots, N \text{ und } m = 1, \ldots, M$$

Gleichung 4-20: Vektor-Quantisierung mit quadriertem Euklidischen Abstand (nach [8])

Tests ergeben, dass die Vektor-Quantisierung mit quadriertem Euklidischen Abstand zu einer gesteigerten Erkennungsrate der gesprochenen Wörter führt. Daher wird dieser Algorithmus ausgewählt.
Der Aufbau des zweidimensionalen Feldes $D(x,y)$ ist in der folgenden Tabelle dargestellt:

$n = N$	$D_{N,1}$	$D_{N,2}$	$D_{N,3}$	…	$D_{N,M-1}$	$D_{N,M}$
$n = N-1$	$D_{N-1,1}$	$D_{N-1,2}$	$D_{N-1,3}$	…	$D_{N-1,M-1}$	$D_{N-1,M}$
…	…	…	…	…	…	…
$n = 3$	$D_{3,1}$	$D_{3,2}$	$D_{3,3}$	…	$D_{3,M-1}$	$D_{3,M}$
$n = 2$	$D_{2,1}$	$D_{2,2}$	$D_{2,3}$	…	$D_{2,M-1}$	$D_{2,M}$
$n = 1$	$D_{1,1}$	$D_{1,2}$	$D_{1,3}$	…	$D_{1,M-1}$	$D_{1,M}$
	$m = 1$	$m = 2$	$m = 3$	…	$m = M-1$	$m = M$

Tabelle 4-3: Aufbau des zweidimensionalen Feldes D (nach [8])

Ein Feldelement $D_{n,m}$ wird aus einer Kombination aus drei vorherigen Feldelementen $D_{n-1,m}$ $D_{n,m-1}$ und $D_{n-1,m-1}$ und dem – dem Feld durch die beiden Indizes n und m zugeordneten – quantisierten Vektorabstand $d_{n,m}$ (x_n, y_m) ermittelt. Dieses Vorgehen wird „lokale Pfadeinschränkung“ genannt.

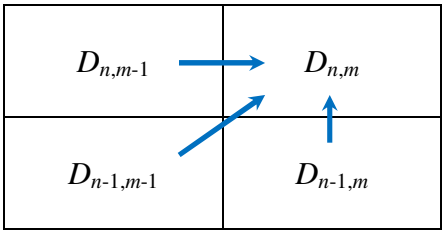

Tabelle 4-4: Berücksichtigung der vorherigen Werte für die Berechnung von $D_{n,m}$ (nach [8])

Zu dem kleinsten Wert (Minimum) der drei vorhergehenden Felder wird der zu dem Feldelement $D_{n,m}$ zugehörige Vektorabstand $d_{n,m}$ (x_n, y_m) addiert.

$$D_{n,m} = \min\left(D_{n-1,m}, D_{n-1,m-1}, D_{n,m-1}\right) + d_{n,m} \qquad \text{für } n = 1, \ldots, N \text{ und } m = 1, \ldots, M$$

Gleichung 4-21: Berechnen der einzelnen Feldelemente $D_{n,m}$ (nach [8])

Dabei ergibt sich ein Randproblem, welches durch drei Anpassungen der Gleichung 4-21 gelöst werden muss:

1. Dem ersten Feldelement $D_{1,1}$ wird, da hier keine vorherigen Werte existieren, direkt das Ergebnis der Vektor-Quantisierung $d_{1,1}$ zugewiesen.

$$D_{1,1} = d_{1,1}$$

Gleichung 4-22: Zuweisung des ersten Feldelementes $D_{1,1}$ (nach [8])

2. Zu den Feldelementen $D_{n,1}$ der ersten Spalte mit $m = 1$ und $n > 1$ existiert nur ein vorheriger Wert, da die anderen beiden vorherigen Werte jenseits des Randes liegen und somit nicht existieren. Hierzu wird folgende angepasste Gleichung verwendet:

$$D_{n,1} = D_{n-1,1} + d_{n,1} \qquad \text{für } n = 2, \ldots, N$$

Gleichung 4-23: Berechnen der einzelnen Feldelemente $D_{n,1}$ der ersten Spalte (nach [8])

3. Zu den Feldelementen $D_{1,m}$ der ersten Zeile mit $n = 1$ und $m > 1$ existiert nur ein vorheriger Wert, da die anderen beiden vorherigen Werte jenseits des Randes liegen und somit nicht existieren. Hierzu wird folgende angepasste Gleichung verwendet:

$$D_{1,m} = D_{1,m-1} + d_{1,m} \qquad \text{für } m = 2, \ldots, M$$

Gleichung 4-24: Berechnen der einzelnen Feldelemente $D_{1,m}$ der ersten Zeile (nach [8])

Das folgende Flussdiagramm zeigt den programmierten Ablauf der dynamischen Zeitverzerrung:

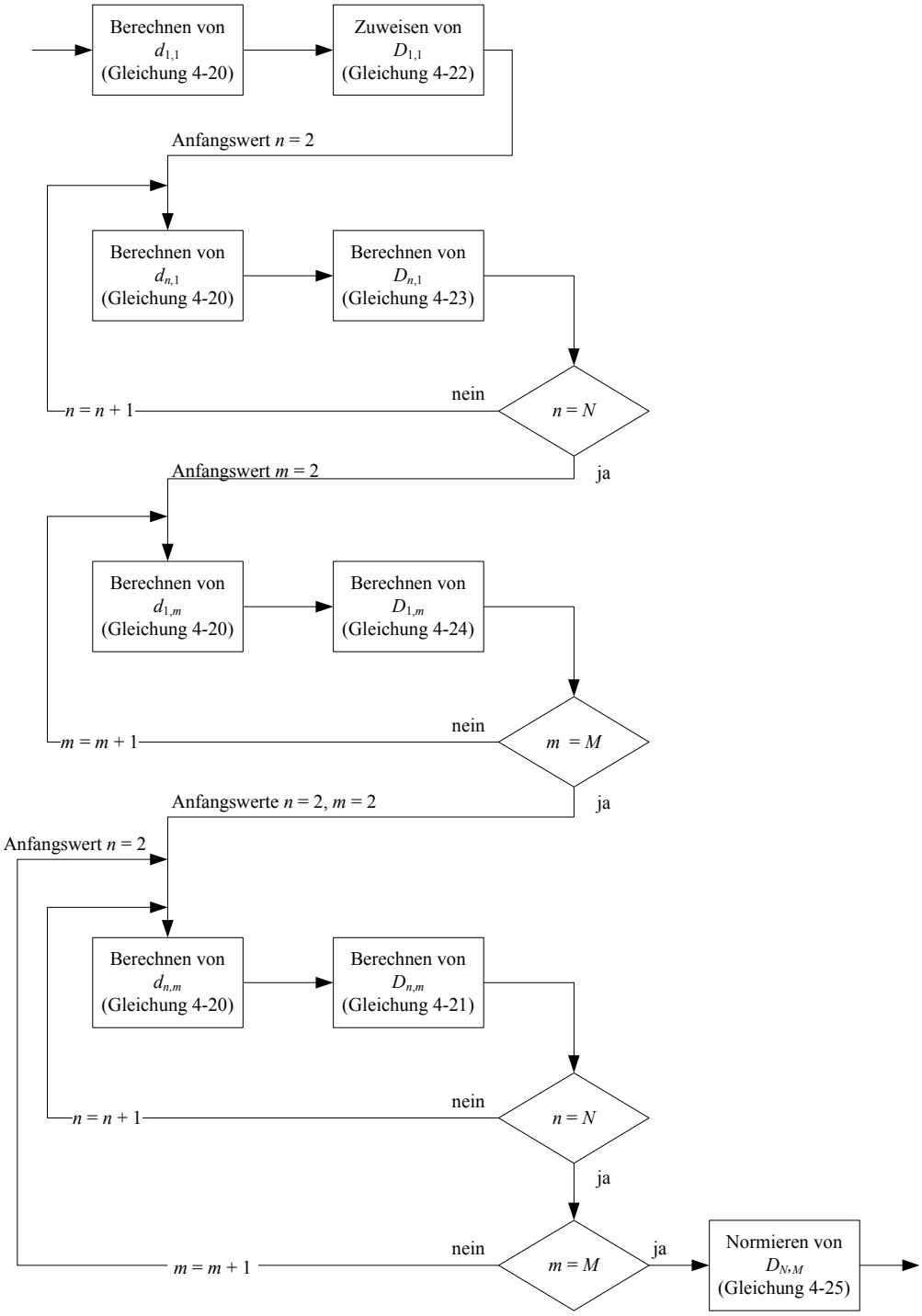

Abbildung 4-12: Ablauf der DTW-Berechnung

Der zuletzt berechnete Wert $D_{N,M}$ gibt den Abstand der beiden miteinander verglichenen Wörter an. Er wird mit der folgenden Gleichung normiert:

$$D_{N,M,normiert} = \frac{D_{N,M}}{N+M}$$

Gleichung 4-25: Normieren von $D_{N,M}$ (nach [2])

Nur das normierte Ergebnis wird verwendet. Der Vergleich mit DTW ist abgeschlossen.

4.6.1 Bewertungskriterien für die DTW-Ergebnisse

Die Bewertungskriterien werden nur bei der eigentlichen Spracherkennung und nicht bei der Anlage oder Erweiterung der Referenzdatenbank angewendet.

Bevor der Datensatz des unbekannten Wortes per DTW mit einem Datensatz eines Referenzwortes verglichen wird, erfolgt eine Wortlängenkontrolle. Ist der Datensatz des unbekannten Wortes deutlich länger oder kürzer als die des Referenzwortes, findet kein Mustervergleich mit DTW statt. Das aktuelle Referenzwort wird ausgeschlossen. Die Grenze liegt bei einer Längendifferenz von 30 Blöcken. Nur wenn die Längendifferenz kleiner ist, wird ein Vergleich durchgeführt.
Nach Abschluss des Vergleichsvorganges zweier Datensätze wird das normierte Ergebnis des DTW mit dem zum Datensatz des Referenzwortes gehörenden Referenz-DTW-Ergebnis verglichen. Das neu berechnete Ergebnis muss sich innerhalb zweier Toleranzgrenzen um das Referenz-DTW-Ergebnis befinden, ansonsten wird das aktuelle Referenzwort ausgeschlossen. Diese Vorgehensweise dient dem verbesserten Ablehnen von nicht in der Datenbank enthaltenen Wörtern.
Nachdem der Datensatz des unbekannten Wortes mit jedem Datensatz aus der Referenzdatenbank verglichen wurde, kann das Wort identifiziert werden. Hierzu wird das kleinste der gültigen DTW-Ergebnisse ausgewählt. Das unbekannte Wort gilt als identifiziert. Es handelt sich somit um das Wort, dessen Referenzdatensatz an dem Vergleich mit dem kleinsten gültigen DTW-Ergebnis beteiligt ist.

4.7 Wichtige Systemparameter

Das komplette Simulationsmodell der Spracherkennung verfügt über mehrere wichtige Parameter, die sowohl beim Anlegen oder Ändern der Referenzdatenbank als auch bei der Spracherkennung identisch sein müssen. Diese Parameter werden direkt zu Beginn von *manage_reference_database.m* (Anlegen der Referenzdatenbank) und *word_recognition.m* (Spracherkennung) definiert. Die meisten Parameter werden bei beiden Programmteilen des Simulationsmodells eingesetzt.
Der Dateiname der Referenzdatenbank ist fest in die beiden Programmteile eingebaut. Dieser muss ohne die Dateiendung „*.mat*“ und ohne die Versionserweiterung des Dateinamens „*_6*“ bzw. „*_7*“ angegeben werden. Soll eine andere, vorhandene Referenzdatenbank verwendet werden, muss diese im jeweiligen Programmteil eingetragen werden. Die Referenzdatenbank muss nur dann, wenn die Datei noch nicht existiert oder leer ist, mit *manage_reference_database.m* neu erstellt werden. Ansonsten reicht ein Neustart des jeweiligen Programmteiles des Simulationsmodells aus.

Folgende, während der Laufzeit nicht veränderbare Parameter werden definiert:
- `p = 10`: Anzahl der verwendeten LPC-Koeffizienten
- `block_length = 240`: Anzahl der Samples pro Block, entspricht 30 ms
- `shift = 80`: Anzahl der Samples, um die der folgende Block verschoben wird, entspricht 10 ms
- `FS = 8000`: Samplefrequenz in Hertz

Werden diese Parameter verändert, muss auf jeden Fall eine neue Referenzdatenbank angelegt werden.

Alle im Folgenden aufgeführten Parameter können während der Laufzeit verändert werden, ohne dass die Simulation neu gestartet oder eine neue Referenzdatenbank angelegt werden muss.
- `upper_threshold = -2.0`: oberer Schwellwert der Wortgrenzenerkennung in dB/20
- `lower_threshold = -2.5`: unterer Schwellwert der Wortgrenzenerkennung in dB/20
- `minimum_word_duration = 0.21`: minimale Wortlänge in Sekunden, entspricht 19 Datenblöcken
- `maximum_break_duration = 0.13`: maximale Länge der Energielücke (Pause) in Sekunden, entspricht 11 Datenblöcken
- `maximum_slope_duration = 0.11`: maximale Länge der Anstiegszeit am Wortanfang bzw. der Abfallzeit am Wortende in Sekunden, entspricht 9 Datenblöcken
- `recording_time = 2`: Länge des Aufnahmezeitraumes in Sekunden
- `set_first_1000_values_to_zero = 'n'`: ermöglicht das Setzen der ersten 1000 aufgenommenen Samples auf den Wert 0
- `play_wav = 'y'`: aktiviert oder deaktiviert die direkt im Anschluss an die Aufnahme stattfindende Audioausgabe des neu aufgenommenen Datensatzes
- `wav_option = 'n'`: schaltet die Sprachquelle zwischen der Aufnahme per Mikrofon (`'n'`) und einer vorhandenen *wav*-Datei (`'y'`) um

Die Spracherkennung verwendet fünf weitere Parameter, die bei der Verwaltung der Referenzdatenbank nicht benötigt werden und daher dort auch nicht integriert sind.
- `maximum_length_difference = 30`: maximale Längendifferenz für die Längenprüfung vor dem Vergleich zweier Wörter per DTW
- `upper_dtw_tolerance = 0.3`: obere Toleranzgrenze für die Kontrolle des DTW-Ergebnisses mit dem in der Referenzdatenbank gespeicherten DTW-Wert
- `lower_dtw_tolerance = 0.4`: untere Toleranzgrenze für die Kontrolle des DTW-Ergebnisses mit dem in der Referenzdatenbank gespeicherten DTW-Wert
- `use_preselection = 'n'`: aktiviert oder deaktiviert die Möglichkeit, die komplette Referenzdatenbank oder nur einen bestimmten Sprecher für die Identifikation zu verwenden
- `preselect_speaker = 0`: gibt die Nummer des für die Identifikation zu verwendenden Sprechers in der Referenzdatenbank an

4.8 Programmierte Funktionen für die Simulation mit *Matlab*®

In diesem Kapitel werden die programmierten Funktionen für die Simulation mit *Matlab*® aufgelistet und beschrieben. Die beiden Hauptprogramme *manage_reference_database.m* und *word_recognition.m* werden in Kapitel 4.9 genauer beschrieben.
Jede Funktion ist hier in einer separaten *m*-Datei untergebracht. Funktions- und Dateiname sind – wie es bei *Matlab*® typisch ist – identisch.
Zwei Funktionen werden während dieser Arbeit nicht selbst programmiert. Es handelt sich dabei um `inputs` und `write_coeff`. Diese Funktionen wurden von Herrn Prof. Dr.-Ing. Ulrich Sauvagerd programmiert und für die Praktika im Rahmen der Vorlesungen zur digitalen Signalverarbeitung zur Verfügung gestellt.

Während der Durchführung einer Simulation werden die folgenden acht Funktionen aufgerufen:

- `calculate_energy`
  Diese Funktion berechnet den kompletten Verlauf der Energie über den gesamten Datenbereich. Für jeden Block wird ein Energiewert berechnet. Dabei werden die Grundlagen der Kapitel 4.3.2 und 4.3.3 umgesetzt.
  Die Funktion erwartet als Übergabeparameter den mit dem FIR-Bandpassfilter bearbeiteten Datensatz `wav_filtered_data`, die Blocklänge `block_length` und den Blockverschiebungswert `shift`.
  Als Rückgabewert wird der Verlauf der blockweise berechneten logarithmierten Energie `logarithmical_energy` zurückgegeben.

- `calculate_energy_boundaries`
  Diese Funktion ermittelt die Grenzen eines in dem Datensatz enthaltenen Wortes. Dabei werden die Grundlagen des Kapitels 4.3.4 umgesetzt.
  Die Funktion erwartet als Übergabeparameter den Verlauf der blockweise berechneten logarithmierten Energie `logarithmical_energy`, den unteren Energieschwellwert `lower_threshold`, den oberen Energieschwellwert `upper_threshold`, die minimale Wortlänge `minimum_word_length`, die maximale Länge der Energielücke (Pause) `maximum_break_length`, die maximale Anstiegs- bzw. Abfallzeit `maximum_slope_length`, die Blocklänge `block_length` und den Blockverschiebungswert `shift`.
  Als Rückgabewerte werden die Nummer des ersten bzw. letzten Blockes des gefundenen Wortes `start_of_word_block` bzw. `end_of_word_block` sowie die Nummer des ersten bzw. letzten Samples des gefundenen Wortes `start_of_word_sample` bzw. `end_of_word_sample` zurückgegeben.

- `calculate_lpc_coefficients`
  Diese Funktion berechnet die LPC-Koeffizienten eines Datenblockes. Dabei werden die Grundlagen des Kapitels 4.4.1 umgesetzt.
  Die Funktion erwartet als Übergabeparameter einen mit dem Hanning-Fenster bearbeiteten Datenblock `data` und die Anzahl der zu berechnenden LPC-Koeffizienten `p`.
  Als Rückgabewerte werden `p`+1 LPC-Koeffizienten zurückgegeben, wobei $a_0 = 1$ enthalten ist.

- `wav_to_lpc`
  Diese Funktion führt alle benötigten Schritte aus, um aus den aufgenommenen Audiodaten die LPC-Koeffizienten des darin enthaltenen Wortes zu berechnen.

Der komplette Datensatz wird mit dem FIR-Bandpassfilter bearbeitet. Sofern die Wiedergabeoption durch den globalen Systemparameter `wav_play` aktiviert ist, wird der gefilterte Datensatz über die Soundkarte des PCs wiedergegeben.
Im Anschluss wird mit dem Ausführen der Funktion `calculate_energy` der Verlauf der Energie über den gesamten Datensatz berechnet und mit dem anschließenden Ausführen der Funktion `calculate_energy_boundaries` nach einem Wort gesucht.
Ist ein Wort in dem Datensatz vorhanden, werden in einer Schleife blockweise die LPC-Koeffizienten durch die Fensterung des aktuellen Datenblockes mit einem Hanning-Fenster und anschließender Ausführung der Funktion `calculate_lpc_coefficients` berechnet. Der aufgenommene, gefilterte Datensatz und der Verlauf der Energie werden in einem grafischen Fenster dargestellt. Die Wortgrenzen werden markiert. Die LPC-Koeffizienten werden ohne den Koeffizienten $a_0 = 1$ gespeichert.
Ist in dem Datenbereich kein Wort enthalten, werden die LPC-Koeffizienten nicht berechnet. Alle Rückgabeparameter werden auf Null gesetzt. Der gefilterte Datenbereich und der Verlauf der Energie werden auch hier in einem grafischen Fenster dargestellt. Da keine Grenzen gefunden wurden, können diese auch nicht markiert werden.
Zur besseren Veranschaulichung wird der interne Ablauf der Funktion `wav_to_lpc` in einem Flussdiagramm dargestellt:

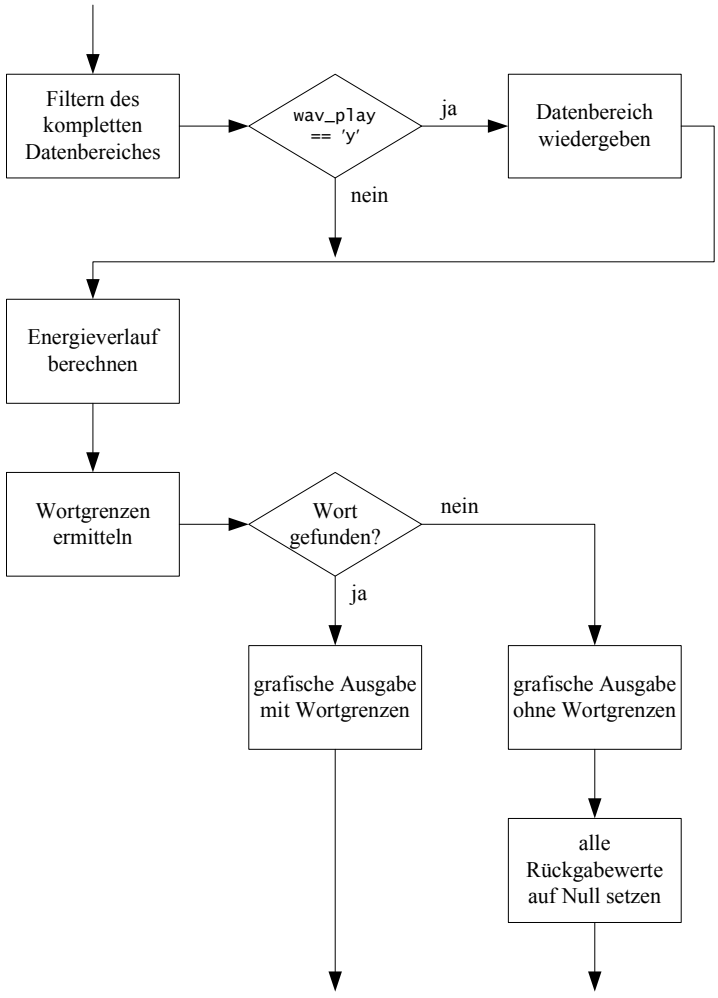

Abbildung 4-13: Flussdiagramm der Funktion `wav_to_lpc`

Die Funktion erwartet als Übergabeparameter den aufgenommenen, unbearbeiteten Datensatz `wav_original_data`, die Samplefrequenz `FS`, die Koeffizienten des Bandpass-

filters `b_FIR_BP`, die Blocklänge `block_length`, den Blockverschiebungswert `shift`, den oberen Energieschwellwert `upper_threshold` und den unteren Energieschwellwert `lower_threshold`. Zusätzlich werden die minimale Wortdauer `minimum_word_duration`, die maximale Dauer der Energielücke (Pause) `maximum_break_duration`, die maximale Anstiegs- bzw. Abfallzeit `maximum_slope_duration`, die Anzahl der zu berechnenden LPC-Koeffizienten `p` und der globale Systemparameter `play_wav` benötigt.
Als Rückgabewerte werden der Verlauf der blockweise berechneten logarithmierten Energie `logarithmical_energy` und die ebenfalls blockweise berechneten LPC-Koeffizienten `lpc_coefficients` zurückgegeben.

- `array_copy_2_of_3`
  Diese Funktion kopiert den Inhalt der zweiten und dritten Dimension eines dreidimensionalen Feldes in ein zweidimensionales Feld. Diese Hilfsfunktion wird benötigt, da *Matlab*® ein dreidimensionales Feld normalerweise nur in ein anderes, mindestens drei Dimensionen umfassendes Feld kopieren kann.
  Die Funktion erwartet als Übergabeparameter das dreidimensionale Feld `source`.
  Als Rückgabewert wird das zweidimensionale Feld `target` zurückgegeben.

- `get_length_of_coefficients`
  Die Funktion ermittelt die tatsächlich genutzte Länge eines Vektors. Dabei wird davon ausgegangen, dass die LPC-Koeffizienten immer von Null verschieden sind. LPC-Koeffizienten könnten theoretisch den Wert Null erreichen. Dieser Fall ist jedoch während der kompletten Programmier- und Testphase nicht aufgetreten.
  Als Ende des Vektors wird das erste Vorkommen einer Null interpretiert.
  Die Funktion erwartet als Übergabeparameter den Vektor unbekannter Länge `vector`.
  Als Rückgabewert wird die Länge des Vektors `vector_length` zurückgegeben.

- `get_length_of_string`
  Diese Funktion ermittelt die tatsächlich genutzte Länge eines Strings. Als Ende des Strings wird das erste Vorkommen des Leerzeichens mit einem Dezimalwert von 32 interpretiert.
  Die Funktion erwartet als Übergabeparameter den String unbekannter Länge `word`.
  Als Rückgabewert wird die Länge des Strings `word_length` zurückgegeben.

- `dtw`
  Diese Funktion berechnet den Abstand zweier Wörter zueinander mittels DTW. Dabei werden die Grundlagen des Kapitels 4.6 umgesetzt. Eine Bewertung des Ergebnisses findet innerhalb dieser Funktion nicht statt.
  Die Funktion erwartet als Übergabeparameter den kompletten Satz an LPC-Koeffizienten des Referenzwortes `reference` und des unbekannten, zu identifizierenden Wortes `current`.
  Als Rückgabewert wird der normierte Abstand `D_NM_scaled` zurückgegeben.

Zusätzlich werden drei weitere Funktionen programmiert, die während der Spracherkennung nicht ausgeführt werden:

- `bandgap`
  Mit dieser Funktion werden die Koeffizienten des in Kapitel 4.3.1 beschriebenen FIR-Bandpassfilters berechnet. Anschließend werden die Koeffizienten für eine Verwendung im Simulationsmodell in *bandgap_coefficients_6.mat* bzw. *bandgap_coefficients_7.mat* gespeichert. Der Dateiname hängt von der verwendeten Version von *Matlab*® ab.
  Zusätzlich werden die Filterkoeffizienten für die spätere Implementierung auf einem DSP in die Datei *bandgap.h* exportiert.

- `mat_to_txt`
  Diese Funktion wird für das Exportieren der Referenzdatenbank vom Simulationsmodell zum DSP benötigt. Eine ausführliche Beschreibung befindet sich in Kapitel 8.3.

- `window_to_h`
  Diese Funktion wird für das Exportieren des bei der Berechnung der LPC-Koeffizienten eingesetzten Hanning-Fensters benötigt. Die Funktion berechnet das Hanning-Fenster und speichert die Koeffizienten in *hanning_window.h* ab, welche anschließend in den Projektordner des DSP-Programmes kopiert werden muss.

4.9 Bedienung und Ablauf des Simulationsmodells

Das Simulationsmodell mit *Matlab*® wird in zwei Programme unterteilt. Diese Unterteilung wird gewählt, um die Verwaltung der Referenzdatenbank von der Spracherkennung zu trennen. Die Referenzdatenbank muss in der Regel nur einmal angelegt werden. Das Programm der Spracherkennung wird somit deutlich kürzer und übersichtlicher.
Beide Teilprogramme greifen dabei auf dieselben, im vorigen Kapitel erläuterten Funktionen und *mat*-Dateien zu, damit die Bedingungen für die Wortreferenzen dieselben sind wie bei der Erkennung der unbekannten Wörter.
Im Folgenden werden die Bedienung und die Abläufe der beiden Programmteile beschrieben.

4.9.1 Verwaltung der Referenzdatenbank mit *manage_reference_database.m*

Mit dem Programm *manage_reference_database.m* kann eine neue Referenzdatenbank angelegt oder eine bestehende Referenzdatenbank erweitert werden. Es können neue Sprecher oder neue Wörter hinzugefügt oder ein bereits existierendes Wort neu angelernt werden.
Zu Beginn des Programmablaufes werden alle globalen Systemparameter gesetzt. Wenn die angegebene Referenzdatenbank existiert, wird diese geladen. Ansonsten wird eine neue Referenzdatenbank erstellt.
Die Filterkoeffizienten für das FIR-Bandpassfilter werden geladen und die Hauptschleife des Programms gestartet.
Zu Beginn der Hauptschleife wird das Menü ausgegeben:

```
-- MAIN MENU --

1 apply a new speaker
2 apply a new word / renew a word
3 show the content of the database
4 view and change the system parameters
5 end program
Please enter your selection:
```

Abbildung 4-14: Hauptmenü von *manage_reference_database.m*

Die einzelnen Menüpunkte werden durch die Eingabe der jeweiligen Nummer, gefolgt von der Enter-Taste, aufgerufen.

Im Folgenden werden die einzelnen Menüpunkte erläutert:
1. Menüpunkt: `apply a new speaker`
   Mit der Auswahl dieses Menüpunktes wird ein neuer Sprecher zu der Referenzdatenbank hinzugefügt.

Im nachfolgenden Flussdiagramm sind die einzelnen Schritte, die hierzu durchgeführt werden müssen, dargestellt:

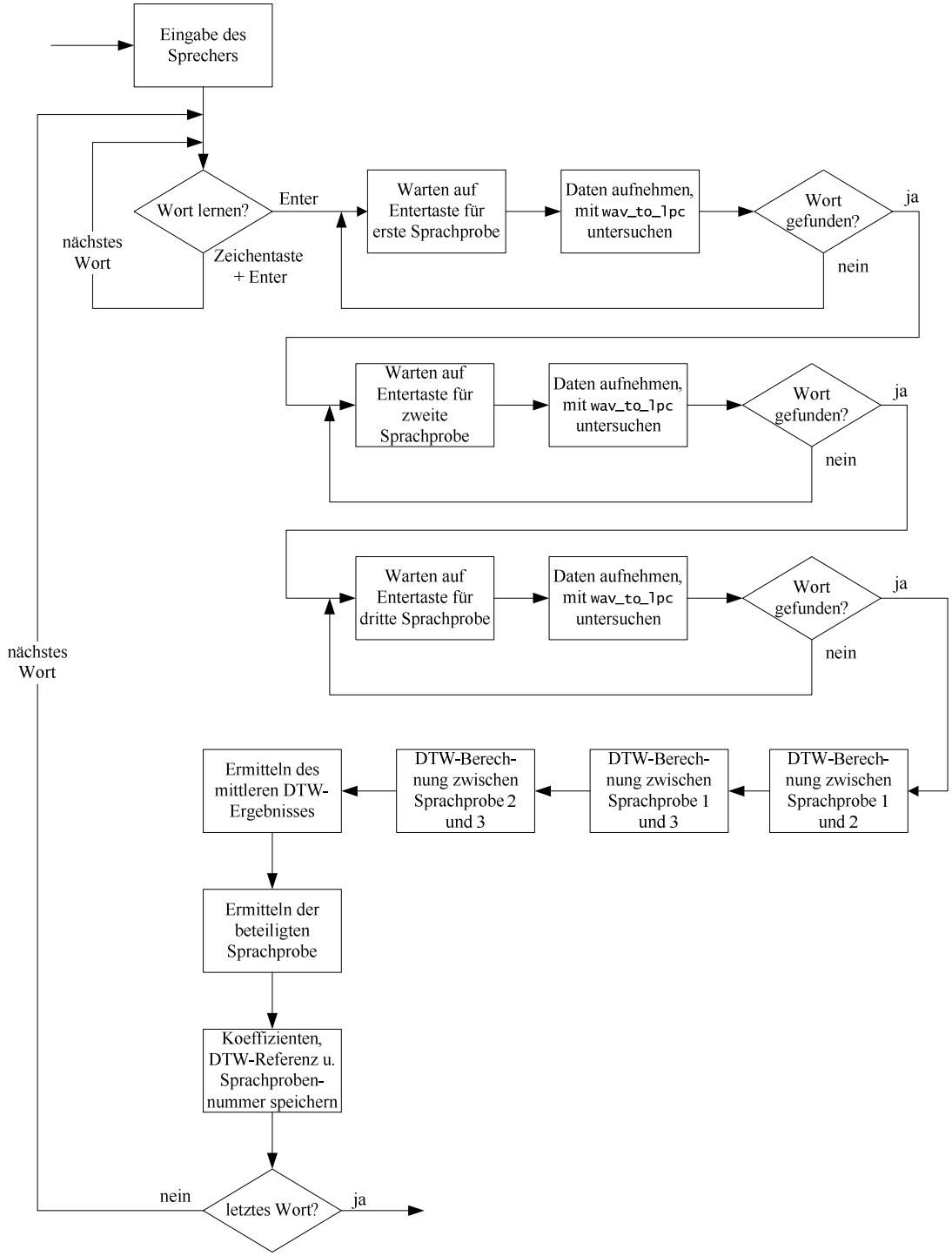

Abbildung 4-15: Flussdiagramm des Menüpunktes `apply a new speaker`

Als Erstes muss der Name des Sprechers eingegeben werden. Dieser wird mit in die *mat*-Datei der Referenzdatenbank gespeichert.
Im nächsten Schritt wird das zu sprechende Wort angezeigt. Durch Drücken einer beliebigen Zeichentaste, gefolgt von der Enter-Taste, wird das Wort übersprungen und mit dem nächsten Wort fortgefahren. Wird nur die Enter-Taste gedrückt, wird dem Anlegen

des Wortes zugestimmt und das Programm ist bereit für die erste Sprachprobe des neuen Wortes. Ein Druck auf die Enter-Taste startet die Aufnahme. Der voreingestellte Wert für die Dauer der Aufnahme beträgt zwei Sekunden. Nach erfolgter Aufnahme wird die Sprachprobe mit der Funktion `wav_to_lpc` bearbeitet. Falls in dem aufgenommenen Bereich kein gültiges Wort vorhanden ist, wird die Aufnahme wiederholt.
Es folgen eine zweite und eine dritte Aufnahme desselben Wortes, da drei gültige Sprachproben eines Wortes zur Anlage eines Referenzdatensatzes benötigt werden.
Im Anschluss werden entsprechend drei Abstandsberechnungen per DTW durchgeführt, nämlich zwischen der ersten und zweiten, zwischen der ersten und dritten sowie zwischen der zweiten und dritten Sprachprobe. Die drei DTW-Ergebnisse werden untersucht und der mittlere Wert als DTW-Referenzwert für das Wort gespeichert.
Im letzten Schritt wird aus den beiden übrig gebliebenen Ergebnissen dasjenige ermittelt, das am dichtesten an dem vorher bestimmten DTW-Referenzwert liegt. Von der Sprachprobe, die an diesem DTW-Ergebnis und dem DTW-Referenzwert beteiligt ist, werden die LPC-Koeffizienten als Referenz für das Wort abgespeichert.
Das Programm wird mit dem nächsten Wort fortgesetzt. Sind alle Wörter angelernt, kehrt das Programm zum Hauptmenü zurück.

2. Menüpunkt: `apply a new word / renew a word`
   Der Ablauf dieses Menüpunktes ähnelt dem ersten Menüpunkt.

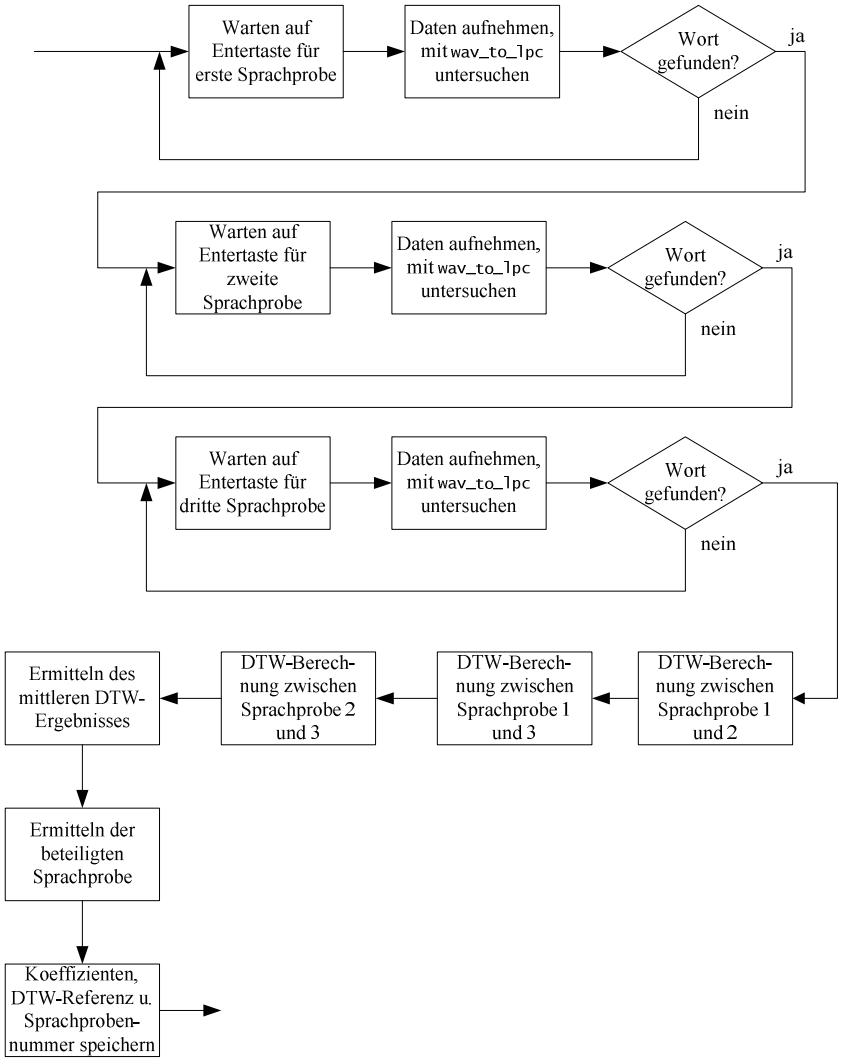

Abbildung 4-16: Flussdiagramm des Menüpunktes `apply a new word / renew a word`

Als Erstes wird eine Liste aller in der Referenzdatenbank enthaltenen Sprecher angefertigt, aus der der Sprecher, bei dem ein Wort ergänzt oder ersetzt werden soll, ausgewählt werden muss. Anschließend werden alle in der Referenzdatenbank enthaltenen Wörter aufgelistet. Aus dieser Liste muss das Wort, welches ersetzt oder neu angelernt werden soll, ausgewählt werden. Soll ein neues Wort hinzugefügt werden, muss hierbei eine Null eingegeben werden.
Wird ein Wort hinzugefügt, muss dieses im Klartext eingegeben werden. Im Anschluss daran wird die Datenbank erweitert; das neue Wort wird im Klartext gespeichert.
Soll ein vorhandenes Wort erneut gesprochen werden, muss dieses nicht neu eingegeben werden; die Datenbank wird nicht erweitert.
In beiden Fällen läuft die Lernphase gleich ab. Diese entspricht der Lernphase aus Menüpunkt 1, jedoch beschränkt auf ein Wort. Demzufolge wird auf eine erneute Beschreibung dieses Programmablaufes verzichtet.

3. Menüpunkt: `show the content of the database`
   Unter diesem Menüpunkt wird der Inhalt der Referenzdatenbank aufgelistet. Der Name des ersten Sprechers wird im Klartext ausgegeben. Darunter werden alle von dem Sprecher angelernten Wörter im Klartext zusammen mit der Länge des Wortes und dem zugehörigen DTW-Referenzwert ausgegeben. Danach wird entsprechend mit den nächsten Sprechern fortgesetzt.
   Bei der Ausgabe der Liste wird eine einfache Qualitätskontrolle der in der Referenzdatenbank enthaltenen Datensätze vorgenommen. Dieser Test wird während der Ausgabe bei jedem einzelnen Wort durchgeführt. Dabei wird geprüft, ob der DTW-Referenzwert eines Referenzwortes kleiner als dessen Länge, zu der acht hinzuaddiert und anschließend durch 100 dividiert wird, ist. Diese Berechnung wurde empirisch im Laufe vieler Tests ermittelt. Ist diese Bedingung erfüllt, wird hinter dem aufgelisteten Wort ein „OK“ ausgegeben. Es hat sich gezeigt, dass Wörter, die diese Bedingung erfüllen, bei der Spracherkennung besser erkannt werden. Außerdem werden sie seltener nicht in der Datenbank enthaltenen Wörtern zugeordnet.
   Nachdem der komplette Inhalt ausgegeben wurde, kehrt das Programm zum Hauptmenü zurück.

4. Menüpunkt: `view and change the system parameters`
   Unter diesem Menüpunkt werden alle während der Laufzeit veränderbaren Parameter aufgelistet.
   Im Anschluss kann jedem einzelnen Parameter der Reihe nach ein neuer Wert zugewiesen werden. Soll der alte Wert erhalten bleiben, muss die entsprechende Abfrage mit der Enter-Taste übersprungen werden.
   Nachdem alle Parameter abgefragt wurden, kehrt das Programm zum Hauptmenü zurück.

5. Menüpunkt: `end program`
   Über diesen Menüpunkt wird das Programm beendet. Wurden zuvor Änderungen an der Referenzdatenbank vorgenommen, wird gefragt, ob die bearbeitete Referenzdatenbank gespeichert werden soll. Dieses geschieht mit der Eingabe des Buchstabens „s“, gefolgt von der Enter-Taste. Um die Datenbank nicht zu speichern, muss eine beliebige Zeichentaste, gefolgt von der Enter-Taste, gedrückt werden. Erst danach wird das Programm beendet. Wurden keine Änderungen vorgenommen, wird die Referenzdatenbank nicht gespeichert und das Programm ohne Abfrage sofort beendet.

4.9.2 Spracherkennung mit *word_recognition.m*

Mit dem Programm *word_recognition.m* wird die Erkennung gesprochener Wörter durchgeführt. Voraussetzung hierfür ist eine existierende Referenzdatenbank mit mindestens einem angelernten Sprecher.
Zu Beginn des Programmablaufes werden alle globalen Systemparameter gesetzt. Die angegebene Referenzdatenbank und die Filterkoeffizienten für das FIR-Bandpassfilter werden geladen, die Hauptschleife des Programms gestartet.
Zu Beginn der Hauptschleife wird das Menü ausgegeben:

```
-- MAIN MENU --

1 word recognition
3 show the content of the database
4 view and change the system parameters
5 end program
Please enter your selection:
```

Abbildung 4-17: Hauptmenü von *word_recognition.m*

Die einzelnen Menüpunkte werden durch die Eingabe der jeweiligen Nummer, gefolgt von der Enter-Taste, aufgerufen.

Im Folgenden werden die einzelnen Menüpunkte erläutert:

1. Menüpunkt: `word recognition`
   Hinter diesem Menüpunkt verbirgt sich die Spracherkennung, deren Ablauf im folgenden Flussdiagramm dargestellt ist.

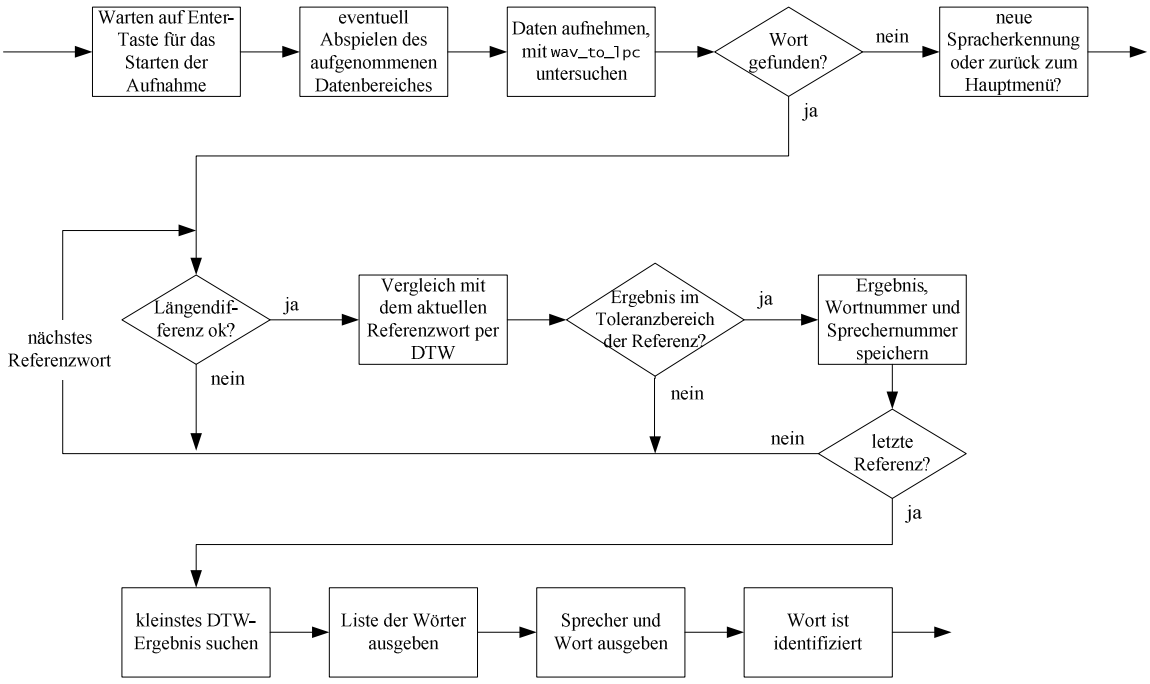

Abbildung 4-18: Flussdiagramm des Menüpunktes `word recognition`

Das Programm ist bereit für eine Aufnahme, die mit der Enter-Taste gestartet wird. Nach Abschluss der Aufnahme wird diese mit einem aus Datum und Uhrzeit bestehenden Dateinamen in eine *wav*-Datei gespeichert und, falls die Option aktiviert ist, über die

Soundkarte ausgegeben. Nach erfolgter Aufnahme wird der Datensatz mit der Funktion `wav_to_lpc` bearbeitet.
Wird in dem aufgenommenen Datensatz kein Wort gefunden, wird eine entsprechende Mitteilung ausgegeben. Durch Betätigen der Enter-Taste wird eine neue Spracherkennung gestartet, während das Programm durch die Eingabe eines beliebigen Zeichens, gefolgt von der Enter-Taste, zum Hauptmenü zurückkehrt.
Befindet sich ein Wort in dem aufgenommenen Datensatz, wird ein Vergleich per DTW durchgeführt. Dieses Wort wird mit jedem Wort der Referenzdatenbank verglichen. Bevor aber ein Vergleich zwischen zwei Wörtern durchgeführt wird, erfolgt die Kontrolle deren Längendifferenz. Ist diese größer als der globale Systemparameter `maximum_length_difference` (siehe Kapitel 4.7), findet der Vergleich nicht statt. In diesem Fall wird mit dem nächsten Referenzwort fortgefahren.
Nach jedem Vergleich wird das Ergebnis mit dem in der Referenzdatenbank gespeicherten DTW-Referenzwert überprüft. Das neu berechnete DTW-Ergebnis muss dabei innerhalb der beiden Toleranzgrenzen oberhalb (`upper_dtw_tolerance`) bzw. unterhalb (`lower_dtw_tolerance`) des DTW-Referenzwertes liegen. Andernfalls kommt das Referenzwort für den aktuellen Identifikationsprozess nicht in Frage.
Alle möglichen Wörter, die für die Identifizierung des unbekannten Wortes zur Auswahl stehen, werden in einer Tabelle ausgegeben. Dabei wird das Wort mit dem kleinsten DTW-Ergebnis markiert. Dieses ist die von der automatischen Spracherkennung getroffene Zuordnung des gesprochenen Wortes. Im Anschluss werden der Name des Sprechers und das Wort im Klartext ausgegeben; die Spracherkennung ist abgeschlossen.
Durch Betätigen der Enter-Taste wird eine neue Spracherkennung gestartet, während das Programm durch die Eingabe eines beliebigen Zeichens, gefolgt von der Enter-Taste, zum Hauptmenü zurückkehrt.

Die Menüpunkte 3 (`show the content of the database`) und 4 (`view and change the system parameters`) sind bei beiden Programmteilen des Simulationsmodells identisch und werden bereits beim Programm *manage_reference_database.m* (siehe Kapitel 4.9.1) erläutert. Daher werden sie hier nicht erneut beschrieben.

5. Menüpunkt: `end program`
   Über diesen Menüpunkt wird das Programm beendet. Da die Referenzdatenbank bei dem Programm *word_recognition.m* nicht verändert werden kann, muss sie hier auch nicht gespeichert werden.

5 Testergebnisse des Simulationsmodells

Jede einzelne programmierte Funktion des Simulationsmodells wurde während der Programmierung ausführlich getestet, um diese als Fehlerquelle so weit wie möglich ausschließen zu können. Ein zusammenhängender Funktionstest wurde jedoch erst möglich, nachdem sowohl eine Referenzdatenbank angelegt als auch eine Spracherkennung durchgeführt werden konnten.
Bei allen Tests lag eine gefüllte Referenzdatenbank zugrunde.
Während des Testablaufes sprach ein Sprecher der Reihe nach einzeln die Worte „links“, „rechts“, „vorwärts“, „rückwärts“, „schneller“, „langsamer“, „halt“ und „drehe“. Diese Reihenfolge wurde insgesamt zehnmal wiederholt. Durch die insgesamt 80 gesprochenen Wörter ist das Ergebnis aussagekräftig. Wichtig dabei war, das nicht jedes Wort zehnmal hintereinander gesprochen und dann mit dem nächsten Wort fortgefahren wurde, sondern dass das nächste Wort sich vom vorherigen unterschied.

Für das Ermitteln der Abweisungsrate nicht in der Referenzdatenbank abgelegter Wörter wurden folgende Worte gesprochen: „Haus“, „Baum“, „Auto“, „Schild“, „sollte“, „jetzt“, „nicht“, „erkennen“, „Sprache“, „wenn“, „dann“, „können“, „das“, „ja“ und „nein“. Auch diese Reihenfolge wurde zehnmal wiederholt.

In der ersten Programmversion wurden die Wörter in der Referenzdatenbank aus nur einer Sprachprobe erstellt. Eine Optimierung war nicht vorgesehen. Das Entscheidungskriterium nach Abschluss der Spracherkennung war die Suche nach dem kleinsten DTW-Ergebnis. Die Rate der richtig zugeordneten Erkennungen lag bei einem Wert von 70%, jedoch erfolgte bei den restlichen 30% eine falsche Zuordnung; sie wurden nicht abgewiesen. Nahezu jedes beliebige Wort wurde über ein Referenzwort fälschlicherweise identifiziert. Die Ausnahme bildeten zu kurze Wörter. Diese scheiterten an der Wortgrenzenbestimmung mit `calculate_energy_boundaries`.

Daraufhin wurden zwei Kriterien für die Bewertung des DTW-Ergebnisses eingesetzt.
Das erste Kriterium war eine Kontrolle der Längendifferenz vor der Durchführung eines Vergleiches zwischen zwei Wörtern per DTW. Unterschieden deren Längen sich zu stark voneinander, wurde der Vergleich nicht durchgeführt. Das Referenzwort kam für das unbekannte Wort nicht in Frage.
Das zweite Kriterium beruhte darauf, dass das Ergebnis eines Vergleiches per DTW zwischen dem unbekannten Wort und dem Referenzwort nicht größer als die beiden zusammenaddierten und durch 100 geteilten Längen sein durfte.
Zusätzlich wurde eine Fensterung der Datenblöcke unmittelbar vor der Berechnung der LPC-Koeffizienten eingebaut. Hierzu wurde ein Hanning-Fenster eingesetzt. Mit dieser Ergänzung wurde eine neue Referenzdatenbank erstellt.
Durch diese drei Maßnahmen konnte die Erkennungsrate bekannter Wörter auf ca. 75% gesteigert werden. 5% der bekannten Wörter wurden abgewiesen, die anderen 20% falsch zugeordnet.
Die Abweisungsrate der oben aufgeführten unbekannten Worte lag bei ca. 40%.
Sowohl die Erkennungsrate als auch die Abweisungsrate waren somit noch nicht zufriedenstellend.

Experimente mit zusätzlichen Merkmalen wie z. B. der Nulldurchgangsrate des mit dem Bandpass gefilterten Sprachsignals pro Block oder der Einsatz dynamischer Merkmale, die über mehrere Blöcke der LPC-Koeffizienten berechnet werden, brachten keine nennenswerten zusätzlichen Verbesserungen. Daher wurden diese nicht übernommen, sondern verworfen.

In der abschließenden, optimierten Programmversion bleibt die Kontrolle der Längendifferenz vor dem Durchführen der DTW erhalten. Eine weitere Verbesserung betrifft die Optimierung der Referenzdatenbank durch ein erweitertes Trainingsverfahren, bei dem aus drei gültigen Sprachproben eines Wortes die beste in der Referenzdatenbank abgelegt wird. Ebenso wird ein DTW-Referenzwert zu jedem Wort der Referenzdatenbank gespeichert (siehe Kapitel 4.9.1).
Während der Spracherkennung wird nun auch zusätzlich auf die DTW-Referenzwerte zurückgegriffen. Nach einem Vergleich zwischen dem zu identifizierenden Wort und einem Referenzwort wird das so berechnete DTW-Ergebnis mit dem zu dem Referenzwort gehörenden DTW-Wert verglichen. Dabei muss sich das neu berechnete Ergebnis innerhalb der beiden Toleranzgrenzen oberhalb (`upper_dtw_tolerance`) bzw. unterhalb (`lower_dtw_tolerance`) des Referenzwertes befinden (siehe Kapitel 4.9.2). Ansonsten wird das Ergebnis verworfen und das Referenzwort steht für die Identifikation des unbekannten Wortes nicht zur Auswahl.

Das abschließende Testergebnis der so optimierten Spracherkennung mit bekannten Wörtern zur Ermittlung der Erkennungsrate wird in der nachfolgenden Tabelle dokumentiert.
Eine grüne Zahl steht für eine richtige Erkennung des Wortes, während eine rote Zahl die falsche Zuordnung des Wortes anzeigt.
Als Referenzdatenbank wird die Datei *reference_database_7.mat* vom 19.11.2009, 11:55h verwendet. In dieser ist nur ein Sprecher enthalten.

Durchlauf	links	rechts	vorwärts	rückwärts	schneller	langsamer	halt	drehe
1	3344	3402	3415	3426	3439	3452	3504	3513
2	3608	3618	3628	3639	3650	3700	3717	3729
3	3740	3749	3759	3809	3819	3828	3838	3847
4	3856	3906	3915	3924	3933	3942	3952	4001
5	4010	4018	4035	4050	4100	4109	4118	4127
6	4137	4145	4154	4204	4212	4221	4230	4250
7	4300	4309	4318	4327	4336	4345	4355	4403
8	4412	4421	4429	4438	4447	4456	4504	4513
9	4523	4531	4539	4551	4600	4608	4617	4625
10	4637	4646	4701	4711	4719	4728	4749	4809

Tabelle 5-1: Abschließendes Testergebnis der Spracherkennung bei bekannten Wörtern

Die Zahlen in der Tabelle sind Bestandteile der Dateinamen der jeweiligen, im Unterordner *recorded_files* des Simulationsmodells abgespeicherten *wav*-Dateien, die direkt und ohne Bearbeitung nach der Aufnahme einer Sprachprobe gespeichert werden. Der Dateiname setzt sich aus Uhrzeit und Datum zusammen: *20091119T12xxxx_word_recognition.wav*, wobei *xxxx* durch die vier Ziffern des jeweiligen Tabellenfeldes zu ersetzen ist. Somit kann der Test jederzeit durch Aktivieren der wav-Option durch den globalen Systemparameter `wav_option` wiederholt werden. Dabei könnten Systemparameter verändert werden, um deren Einfluss auf die Spracherkennung zu untersuchen.

Wie aus der Tabelle ersichtlich beträgt die ermittelte Erkennungsrate bekannter Wörter somit ca. 95%. Ein weiterer Test ergab, dass die Abweisungsrate bei den oben erwähnten, nicht in der Referenzdatenbank enthaltenen Worte auf ca. 75% steigt.
Mit diesem äußerst zufriedenstellenden Ergebnis ist die Entwicklung des Simulationsmodells zur Spracherkennung mit *Matlab*® abgeschlossen.

6 Hardware der Kommunikationsplatine

Da das zentrale Bauteil der Kommunikationsplatine ein programmierbarer Mikroprozessor ist, gehören die beiden Kapitel 6 (Hardware der Kommunikationsplatine) und 7 (Software der Kommunikationsplatine) eng zusammen. Teilweise werden in diesem Kapitel Funktionsblöcke erwähnt, die erst in Kapitel 7 ausführlicher erläutert werden (z. B. die Abläufe beim Ansteuern des Displays). Würden diese beiden Kapitel in ihrer Reihenfolge vertauscht, so würde der reale Ablauf verfälscht: Zuerst werden grobe Züge der Hardware konstruiert, erst danach kann die Softwareentwicklung erfolgen.
In diesem Kapitel wird die Hardwareentwicklung erläutert, von den einzelnen besonderen Baugruppen bis hin zu den kompletten Schaltplänen und dem Platinenlayout.
Die unten abgebildete Übersicht der Hardware zeigt die zentrale Lage des Mikroprozessors. Die Funktionsblöcke sind alle mit ihm verbunden und können ohne ihn nicht miteinander kommunizieren. Damit die komplette Hardware zuverlässig und sicher ihre Funktion erfüllen kann, ist es wichtig, dass die Bauteile miteinander kompatibel sind. Sämtliche Bauteile werden so gewählt, dass sie mit einer Betriebsspannung von 5V DC arbeiten. Die einzelnen Funktionsblöcke werden im weiteren Verlauf des Kapitels beschrieben.

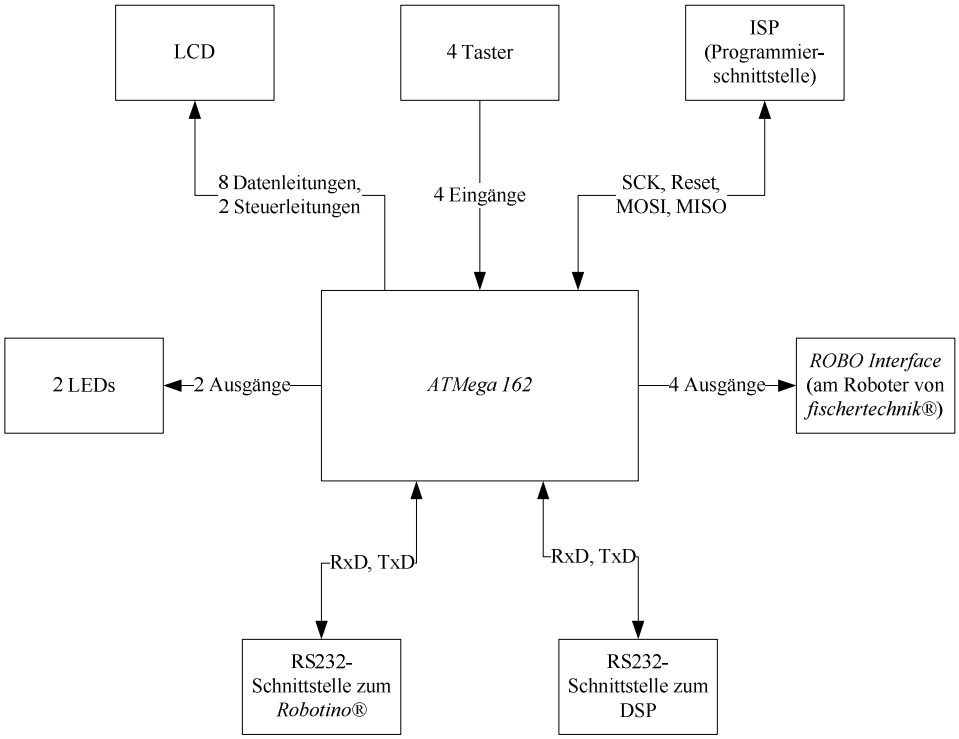

Abbildung 6-1: Übersicht über die einzelnen Hardwarekomponenten (Blockschaltbild)

6.1 Prozessorauswahl

Das wichtigste Kriterium zur Auswahl des Prozessors ist, dass er mit zwei echten USART-Schnittstellen (engl.: Universal Synchronous/Asynchronous Receiver Transmitter) ausgestattet sein muss, die unabhängig voneinander eingestellt werden können. Es besteht zwar auch die Möglichkeit, eine USART-Schnittstelle in Software zu implementieren. Diese würde jedoch – im Gegensatz zu einer in Hardware implementierten USART-Schnittstelle – deutlich mehr Prozessorzyklen benötigen und die aufwendige Kommunikation mit dem *Robotino*® somit erschweren, wenn nicht sogar unmöglich machen.

Der Hersteller *Atmel* bietet moderne, weit verbreitete 8-Bit-Mikroprozessoren mit RISC-Technologie (engl.: Reduced Instruction Set Computing) in verschiedenen Größenordnungen und mit verschiedenem Funktionsumfang an. Diese Prozessoren sind preiswert und sehr zuverlässig. Sie können sowohl in C als auch in Assembler programmiert werden.
Die Programmierung des Prozessors erfolgt im auf der Platine eingebauten Zustand durch einen ISP (engl.: In System Programmer) über zwei Datenleitungen, Taktsignal und Reset. Da der Mikroprozessor das zentrale Bauteil des gesamten Projektes ist und daher mit jeder Funktionsgruppe kommunizieren muss, sind folgende Schnittstellen unbedingt erforderlich:

- zwei echte USART-Schnittstellen zur Kommunikation mit dem DSP und dem *Robotino*®
- zehn freie Pins (Ausgänge) für das LCD
- vier freie Pins (Eingänge) für die optionalen Taster des *Robotino*®
- vier freie Pins (Ausgänge) für das *ROBO Interface*
- zwei freie Pins (Ausgänge) für die Status-LEDs
- Programmierschnittstelle für den ISP

Es stehen mehrere Prozessoren zur Auswahl. Aufgrund einer Vorabplanung über die Aufgaben des Prozessors wird der *Atmega162* [11] gewählt. Dieser verfügt über 16 KB Programmspeicher, 1 KB RAM, 2 USART-Schnittstellen sowie 28 freie I/O-Pins, die für das LCD, die LEDs, die Taster und das *ROBO Interface* benutzt werden können. Der Prozessor arbeitet mit 4,5 – 5,5V DC, die maximale Taktfrequenz beträgt 16 MHz. Er ist sowohl in einer 40-poligen PDIP-Variante (engl.: Plastic Dual Inline Package) als auch in einer 44-poligen TQFP-Variante (engl.: Thin Quad Flat Package) erhältlich.
In der folgenden Tabelle ist die verwendete Pinbelegung des Mikroprozessors in beiden Varianten aufgelistet. Diese Tabelle wird sowohl für die Hardwareentwicklung (Pins des Prozessors) als auch für die Softwareentwicklung (Port und Index) benötigt. Das *ROBO Interface* ist nur in der Version, die auf der Lochrasterplatine aufgebaut ist, vorhanden.

Port	Pin (PDIP)	Pin (TQFP)	Pinfunktion	Peripherie
A[0..7]	32..39	30..37	Output	LCD Data
B0, B1	1, 2	40, 41	I/O	*ROBO Interface*
B2, B3	3, 4	42, 43	RxD1, TxD1	RS232 (DSP)
B4	5	44	I/O	frei
B[5..7]	6..8	1..3	MISO, MOSI, SCK	ISP
C[0..2]	21..23	18..20	I/O	frei
C3	24	21	Output	rote LED (Error)
C4	25	22	Output	gelbe LED (Antwort)
C5	26	23	Output	LCD Beleuchtung
C6	27	24	Output	LCD RS
C7	28	25	Output	LCD E
D0, D1	10, 11	5, 7	RxD0, TxD0	RS232 (*Robotino*®)
D2, D3	12, 13	8, 9	I/O	*ROBO Interface*
D4	14	10	Input	Taster *AB*
D5	15	11	Input	Taster *AUF*
D6	16	12	Input	Taster *LINKS*
D7	17	13	Input	Taster *ENTER*
E[0..2]	29..31	26, 27, 29	I/O	frei

Tabelle 6-1: Pinbelegung des Mikroprozessors (Port, Pin und Funktion nach [11])

6.2 Spannungsversorgung

Es werden zwei verschiedene Gleichspannungen benötigt:
- +3,3V für das *D.Module.C6713* und das *D.Module.PCM3003*
- +5V für den Betrieb der Kommunikationsplatine und des LCDs

Da das gesamte System mobil und somit netzunabhängig arbeiten soll, müssen die Kommunikationsplatine und der DSP vom Roboter mit Spannung versorgt werden. Der *Robotino*® stellt 5V DC mit 2A [5] zur Verfügung. Diese 5V werden direkt zur Kommunikationsplatine geleitet. Um einen eventuellen Einbau in den *Robotino*® zu ermöglichen, werden das in ihm eingesetzte Steckersystem und dieselbe Pinbelegung verwendet. Als Steckverbindung kommen ein 10-poliger Pfostenstecker mit Crimpanschluss für Flachbandkabel und die passende Stiftleiste (Wannenleiste) auf der Platine zum Einsatz. Folgende Pinbelegung ist vorgegeben:

Spannung	Pin	Pin	Spannung
GND	1	2	+5V
BATT	3	4	+12V
-5V	5	6	-12V
GND	7	8	+5V
-	9	10	-

Tabelle 6-2: Pinbelegung des Steckers X14 für die Spannungsversorgung (nach [5])

Dabei sind am *Robotino*® nur die Pins 1, 2, 7 und 8 nach dieser Tabelle belegt. Die anderen Pins sind nicht belegt und werden daher auch nicht angeschlossen. Der auf der Kommunikationsplatine eingebaute Stecker X14 hat demnach auch nur vier Pins, die anderen sechs werden entfernt. Die entsprechenden Löcher am Pfostenstecker werden mit Sekundenkleber verstopft, um das unbeabsichtigte Einstecken auf der Stiftleiste X10 für die serielle Schnittstelle zu vermeiden, da hierdurch sowohl der *Robotino*® als auch die Kommunikationsplatine stark beschädigt werden könnten.
Eine Verpolungsschutzdiode ist nicht vorgesehen. Einerseits ist die Stiftleiste durch die umgebende Wanne verpolungssicher, andererseits würde die Spannung um ca. 0,7V auf ca. 4,3V sinken.
Das *D.Module.C6713* wird zusammen mit dem *D.Module.PCM3003* auf der Kommunikationsplatine montiert und über die vorhandenen Stiftleisten angeschlossen. Dabei werden für die Spannungsversorgung entweder die Pins A1 (Vcc, +3,3V) und B1 (GND) oder A32 (GND) und B32 (Vcc, +3,3V) benötigt [7]. Um Störungen durch Stromschleifen zu vermeiden, soll laut Datenblatt nur eines der beiden Paare verwendet werden. Um kürzere Leiterbahnen einzusetzen, wird die Spannungsversorgung auf die Pins A1 und B1 festgelegt.

Das *D.Module.C6713* und das dazugehörende *D.Module.PCM3003* benötigen eine Versorgungsspannung von 3,3V bei einem Spitzenstrom von ca. 1,6A (*D.Module.C6713* max. 1,5A [7], *D.Module.PCM3003* typ. 85mA [6]). Da an einem Linearregler eine zu hohe Verlustleistung abfallen würde und ein entsprechender Kühlkörper relativ groß wäre, wird hier ein Schaltregler eingesetzt, dessen Verlustleistung deutlich geringer ist.
Schaltregler, die von lediglich 5V auf 3,3V bei einem Ausgangsstrom von mindestens 1,6A umsetzen können, sind sehr aufwendig und entsprechend teuer. Gewählt wird ein *R-523.3PA* des Herstellers *RECOM Electronic GmbH*, der bei einer Eingangsspannung von 4,5 – 18V eine Ausgangsspannung von 3,3V und einen maximalen Ausgangsstrom von 2A zur Verfügung stellt [12]. Dieser Schaltregler (IC4) wird genau nach Datenblatt [12] mit den Kondensatoren C17 und C18 in unmittelbarer Nähe zu den Ein- bzw. Ausgangspins

beschaltet. Da die Ausgangsspannung nie abgeschaltet werden soll, bleibt Pin 1 (ON/OFF) offen. Die Ausgangsspannung muss nicht variiert werden. Daher bleibt Pin 12 (Vout-Adj.) ebenfalls offen. Die im Datenblatt benannten Widerstände R1 und R2 werden nicht bestückt. Der Schaltregler wird auf der Unterseite der Platine platziert, um durch die beiden Kupferlagen (GND) der Platine die Störeinflüsse auf die analogen Eingänge des *D.Module.PCM3003* zu reduzieren.
Bei der Version der Kommunikationsplatine, die auf der Lochrasterplatine aufgebaut wird und für die Steuerung des *fischertechnik*® Roboters vorgesehen ist, beträgt die Akkuspannung 9V. Für die Versorgungsspannung von 5V wird hier ein Linearregler (*7805*) eingesetzt. Für die Versorgungsspannung von 3,3V kann ebenfalls der oben genannte *R-523.3PA* verwendet werden. Dieser wird jedoch bisher noch nicht fest eingeplant.

6.3 Die beiden RS232-Schnittstellen

Da die USART-Schnittstellen am Prozessor mit TTL-Pegeln (0V / +5V) arbeiten, sind externe Bustreiber nötig, denn sowohl der *Robotino*® als auch der DSP arbeiten mit RS232-typischen Pegeln. Hierfür werden zwei *MAX202* des Herstellers *Maxim Integrated Products* verwendet. Diese sind sowohl in einer 16-poligen PDIP-Variante (engl.: Plastic Dual Inline Package) als auch in einer 16-poligen Wide SO-Variante (engl.: Wide Small Outline) erhältlich. Sie sind für einen Betrieb an 5V ausgelegt und benötigen nur fünf externe Kondensatoren.
Um mit den beiden vorgegebenen Baudraten möglichst ohne Kommunikationsfehler arbeiten zu können, wird ein Quarz mit 14,7456 MHz gewählt.

6.3.1 Verbindung mit dem DSP

Für die Verbindung mit dem DSP wird IC3 mit den Kondensatoren C9, C10, C11, C12 und C13 verwendet.
Das *D.Module.C6713* wird zusammen mit dem *D.Module.PCM3003* auf der Kommunikationsplatine montiert und über die vorhandenen Stiftleisten angeschlossen. Dabei werden für die serielle Schnittstelle nur die Pins A3 (TxD), A5 (RxD) und A6 (GND) benötigt [7]. Die Handshake-Signale an den Pins A2 (RTS) und A4 (CTS) werden nicht genutzt und daher nicht angeschlossen.

6.3.2 Verbindung mit dem *Robotino*®

Für die Verbindung mit dem *Robotino*® wird IC2 mit den Kondensatoren C5, C6, C7, C8 und C14 verwendet.
Um einen eventuellen Einbau in den *Robotino*® zu ermöglichen, werden wieder wie bei der Spannungsversorgung (siehe 6.2) das im Roboter eingesetzte Steckersystem und dieselbe Pinbelegung verwendet. Als Steckverbindung kommen auch hier ein 10-poliger Pfostenstecker mit Crimpanschluss für Flachbandkabel und die passende Stiftleiste (Wannenleiste) auf der Platine zum Einsatz. Beim Anschließen ist unbedingt darauf zu achten, dass die Stecker der Spannungsversorgung und der seriellen Schnittstelle nicht verwechselt werden, da ansonsten die Kommunikationsplatine und der *Robotino*® beschädigt werden können.

Folgende Pinbelegung ist vorgegeben:

Spannung	Pin	Pin	Spannung
DCD	1	2	DSR
RxD	3	4	RTS
TxD	5	6	CTS
DTR	7	8	RI
GND	9	10	+5V

Tabelle 6-3: Pinbelegung des Steckers X10 für die serielle Schnittstelle (nach [5])

Bei der Kommunikation wird kein Handshake-Verfahren eingesetzt. Daher werden nur die Signale RxD und TxD sowie GND angeschlossen. Alle anderen Pins bleiben offen.
Zur Unterscheidung von dem Steckverbinder für die Spannungsversorgung (X14, schwarz), der ebenfalls 10-polig ist, wird für X10 zur besseren Unterscheidbarkeit ein blauer Stecker für die Montage auf der Platine gewählt.

6.4 Display und Taster

Im Deckel des *Robotino*® ist ein Standard-LCD mit 4 Zeilen, 16 Zeichen pro Zeile, blauer Hintergrundbeleuchtung und weißer Schrift eingebaut. Das Display entspricht einem *EA W164B-NLW* [14] des Herstellers *ELECTRONIC ASSEMBLY GmbH*. Es ist zusammen mit einer kleinen Platine in den Deckel des *Robotino*® eingebaut. Die Ansteuerung erfolgt über einen 26-poligen Pfostenstecker. Über dieselbe Platine und dieselbe Steckverbindung sind auch die vier in den Deckel integrierten Taster angeschlossen. Da über die gesamte Elektronik keinerlei Unterlagen verfügbar sind, muss die Pinbelegung durch Verfolgung der Leiterbahnen und durch Messungen mit einem Durchgangsprüfer ermittelt werden. Der folgende Ausschnitt aus dem Schaltplan (siehe 12.2) zeigt die ermittelte Pinbelegung des Pfostensteckers:

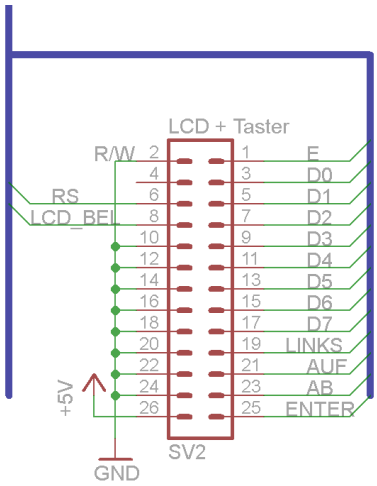

Abbildung 6-2: Anschlussbelegung des Pfostensteckers für das LCD und die vier Taster

Das Display benötigt nur eine Spannungsversorgung von 5V DC und kann direkt an den Mikroprozessor angeschlossen werden. Für die Kommunikation werden acht Datenleitungen, eine Enable-Leitung (E, positive Flanke zur Datenübernahme) und eine weitere Leitung (RS),

die angibt, ob die anstehenden Daten als Zeichen auf dem Display ausgegeben werden sollen oder ob es sich um einen Steuerbefehl für das Display handelt, benötigt.
Es können nicht nur Daten zum LCD gesendet, sondern auch aus ihm gelesen werden. Da hiervon kein Gebrauch gemacht wird, ist die Steuerleitung dieser Funktion (R/W, engl.: Read / Write, Pin 2 am Pfostenstecker, Pin 5 am LCD) fest mit Masse verbunden. Somit ist die Datenrichtung vom Mikroprozessor zum LCD dauerhaft eingestellt.
Die zeitlichen Signalabläufe, die zum Ansteuern des Displays unbedingt eingehalten werden müssen, werden im Kapitel 7.4.4 beschrieben.
Die vier Taster liefern im ungedrückten Zustand +5V und im gedrückten Zustand 0V. Somit können sie ebenfalls direkt an den Mikroprozessor angeschlossen werden. Es ist kein weiterer Schaltungsaufwand erforderlich.

6.5 LEDs

Auf der Platine kommen insgesamt vier LEDs zum Einsatz:
- LED1 (grün) mit Vorwiderstand R5 zur Anzeige der vorhandenen Spannung +5V
- LED2 (grün) mit Vorwiderstand R2 zur Anzeige der vorhandenen Spannung +3,3V
- LED3 (rot) mit Vorwiderstand R3 als Error-Anzeige an Port C3 des Mikroprozessors
- LED4 (gelb) mit Vorwiderstand R4 als Antwort-Anzeige an Port C4 des Mikroprozessors

Als LEDs sind normale, diffuse 3mm-LEDs vorgesehen. Die Vorwiderstände sind so gewählt, dass ein gut sichtbarer Betrieb ermöglicht wird.

6.6 Das *ROBO Interface*

Das *ROBO Interface* ist nicht Bestandteil der gefertigten Platine. Es wird nur bei der Version auf der Lochrasterplatine eingebaut. Um für beide Platinenversionen dieselbe Software verwenden zu können, werden die entsprechenden Pins bei der gefertigten Platine offen gelassen.
Das *ROBO Interface* verfügt über einen Anschluss für einen 26-poligen Pfostenstecker, dessen Pinbelegung der folgenden Tabelle entnommen werden kann:

Funktion	Pin	Pin	Funktion
+9V	1	2	GND für Messanschlüsse
analoger Widerstandseingang AX	3	4	analoger Widerstandseingang AY
analoger Eingang A1	5	6	analoger Eingang A2
Distanzsensor D1	7	8	Distanzsensor D2
Zähleingang	9	10	GND
Eingang I1	11	12	Eingang I2
Eingang I3	13	14	Eingang I4
Eingang I5	15	16	Eingang I6
Eingang I7	17	18	Eingang I8
Ausgang O1	19	20	Ausgang O2
Ausgang O3	21	22	Ausgang O4
Ausgang O5	23	24	Ausgang O6
Ausgang O7	25	26	Ausgang O8

Tabelle 6-4: Pinbelegung des 26-poligen Pfostensteckers am *ROBO Interface* [15]

Da nur die Pins 1, 10, 11, 12, 13 und 14 verwendet werden und eine Verwechslungsgefahr mit dem Anschluss für das Display ausgeschlossen werden soll, wird auf der Platine ein 14-poliger Pfostenstecker eingesetzt. Dadurch gelangen die Signale der Pins 15 – 26 des *ROBO Interface* nicht bis zur Lochrasterplatine.
Laut Bedienungsanleitung des *ROBO Interface* [15] arbeiten die Eingänge mit einer Spannung von 0V bzw. 9V. Eine Messung mit einem regelbaren Netzteil, angeschlossen am Eingang I1, ergibt eine Schaltschwelle von ca. 3,5V. Somit können die entsprechenden Ausgangspins des Mikrocontrollers ohne Adapterschaltung direkt an die Eingänge des *ROBO Interface* angeschlossen werden. Während mehrerer praktischer Tests traten keine Verbindungsfehler auf.

6.7 Testen der Hardware vor dem Platinenlayout

Die komplette Hardware wird zum Testen und Experimentieren vorerst auf einer Lochrasterplatine aufgebaut, damit alle Funktionsblöcke zur Sicherstellung der gewünschten Funktionen richtig dimensioniert werden können. Parallel dazu entsteht die erste Version der Software für den Mikroprozessor, so dass die Hardware in Betrieb genommen und das Zusammenspielen der Hardwarefunktionsgruppen verifiziert werden kann.
Während des Aufbauens und ausgiebigen Untersuchens der Testplatine entsteht auch nach und nach der Schaltplan. Nachdem alle Tests erfolgreich abgeschlossen sind, kann daraus das Platinenlayout erstellt werden.

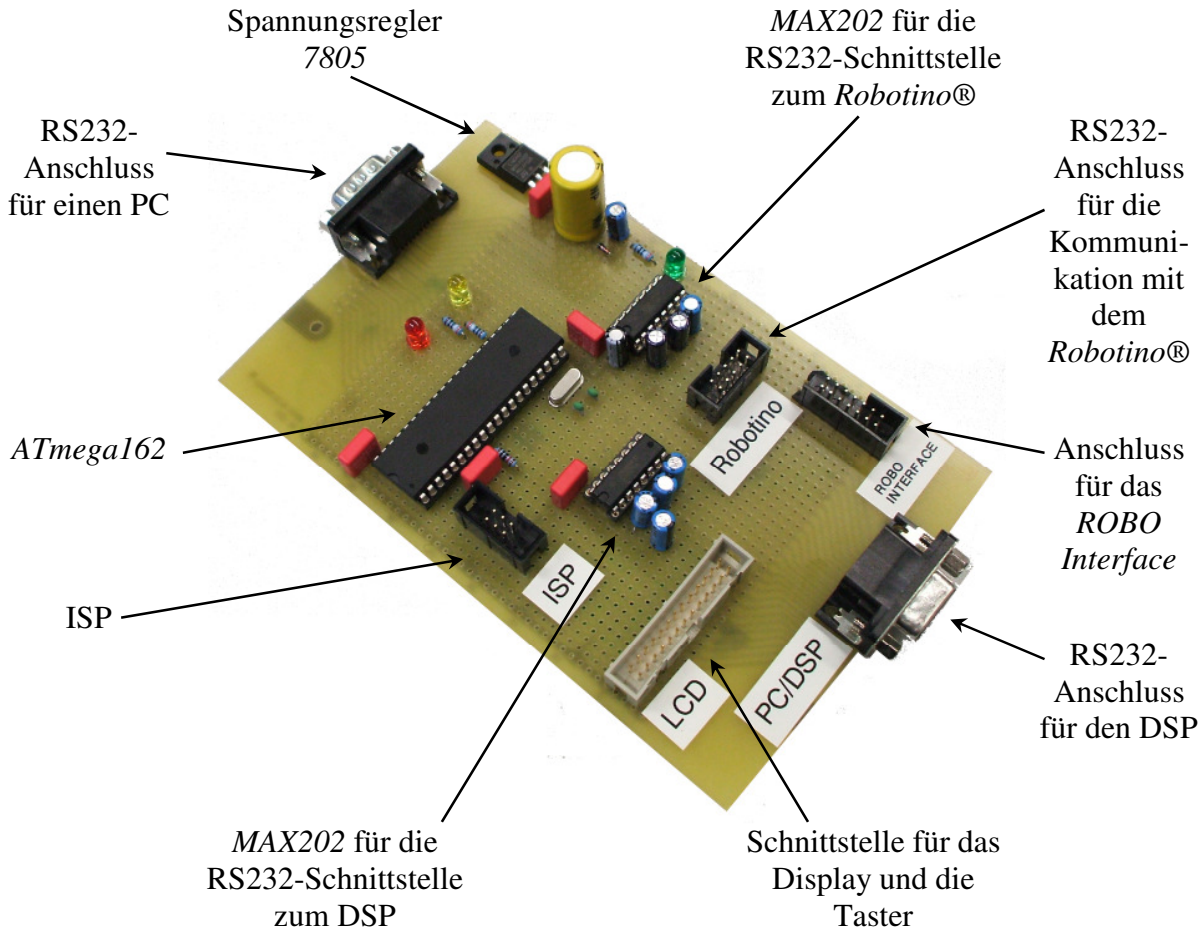

Abbildung 6-3: Aufbau der Hardware auf einer Lochrasterplatine

Auf der Lochrasterplatine werden alle Bauteile in bedrahteter Bauform eingesetzt. Die genauen Bezeichnungen der ICs lauten:

- IC1 (Mikroprozessor): *ATmega 162-16PU*
- IC2, IC3 (RS232-Pegelumsetzer): *MAX 202 CPE*

Die Version auf der Lochrasterplatine wird für das Ansteuern des Roboters von *fischertechnik®* verwendet. Daher ist die Platine für einen direkten Betrieb an 9V ausgelegt. Eine Verpolungsschutzdiode ist vorhanden. Die Spannung wird über die Verbindung zum *ROBO Interface* geliefert. Eine separate Leitung ist somit nicht erforderlich.
Das *D.Module.C6713* und das *D.Module.PCM3003* können momentan nicht mit auf die Platine aufgesteckt werden und müssen daher über ein normales RS232-Kabel an die Platine angeschlossen werden.

6.8 Fertige Hardware

Auf der gefertigten Platine werden alle ICs und Widerstände sowie die meisten Kondensatoren in SMD-Bauform eingesetzt, da der Platz und auch die mögliche Bauhöhe sehr begrenzt sind. Die genauen Bezeichnungen der ICs lauten:

- IC1 (Mikroprozessor): *ATmega 162-16AU*
- IC2, IC3 (RS232-Pegelumsetzer): *MAX 202 ECWE*

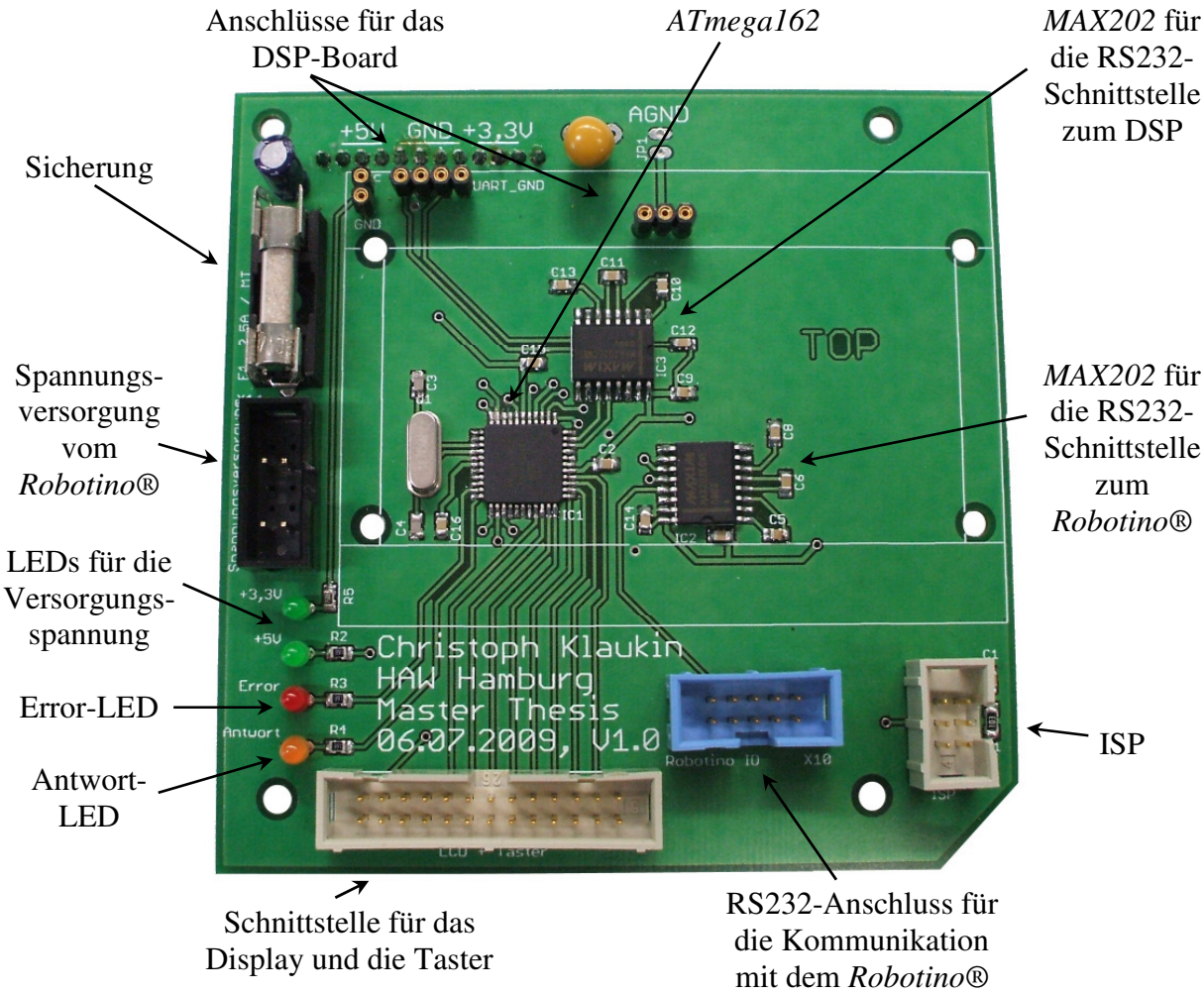

Abbildung 6-4: Gefertigte, bestückte Platine (Oberseite)

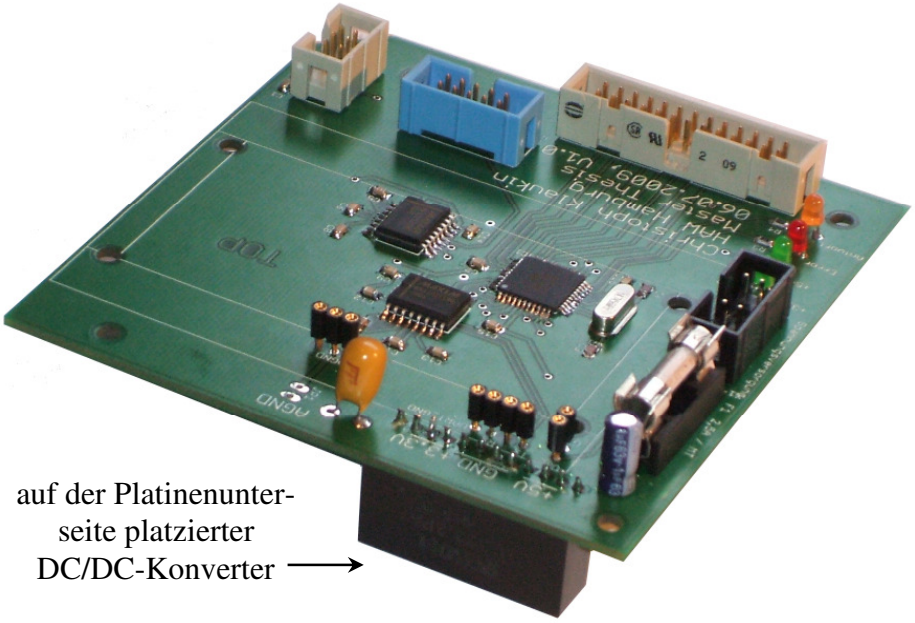

Abbildung 6-5: Gefertigte, bestückte Platine (Seitenansicht)

7 Software der Kommunikationsplatine

In diesem Kapitel werden die komplette Struktur, das Konzept sowie wichtige Hintergrundinformationen für das Verständnis der Software, die für den *Atmel*-Prozessor auf der Kommunikationsplatine benötigt wird, erläutert. Das komplexe Hauptprogramm wird zudem mit einem Flussdiagramm veranschaulicht. Bei allen anderen Funktionen ist ein solches Flussdiagramm nicht sinnvoll, da diese sehr übersichtlich aufgebaut sind.
Der komplette Quellcode umfasst 1323 Programm- und Kommentarzeilen. Er liegt der Masterarbeit auf CD-ROM bei (siehe 12.4).

7.1 Modulare Struktur und Funktionsgruppen

Zur Realisierung werden sämtliche Teilaufgaben in kompakten Funktionen programmiert und nach ihrer Zugehörigkeit in einzelnen Quellcodedateien abgelegt. Durch dieses Konzept reduziert sich der Programmieraufwand deutlich, da viele Funktionen somit nur einmal programmiert werden müssen. Anschließend kann an beliebig vielen Stellen im Programm durch einen klar verständlichen und kurzen Aufruf auf diese Funktionen zugegriffen werden. Die Lesbarkeit des Quellcodes wird dadurch ebenfalls erleichtert; der Aufwand an zusätzlichen Kommentaren reduziert sich.
Die Unterteilung in Funktionsgruppen bedeutet hauptsächlich das sinnvolle Abspeichern der einzelnen Funktionen in die verschiedenen Quellcodedateien. Diese sind nach Hardwareteilen oder Aufgaben benannt.
Alle Module werden so allgemein wie möglich programmiert, damit diese an verschiedenen Stellen für unterschiedliche Parameter eingesetzt werden können. Somit benötigen einige Funktionen neben einem Wert auch noch andere Parameter (z. B. Positionskoordinaten für das LCD). Durch diesen Ansatz werden eine hohe Flexibilität und eine einfache Erweiterung der Software ermöglicht.
In der folgenden Beschreibung der Software werden Variablen- und Funktionsnamen in quellcodetypischer Schrift dargestellt. Die hierfür gewählte Groß- und Kleinschreibung entspricht der im Quellcode verwendeten Schreibweise.

7.2 Struktur der *main.c*

Die *main.c* ist die Hauptdatei, in der das Hauptprogramm, das den genauen Ablauf vorgibt und steuert, gespeichert ist. In dieser Datei werden auch alle verwendeten Quellcodedateien und die Definitionsdatei *const.h* eingebunden. Die Definitionen für die Motorsteuerung befinden sich aus Gründen der Übersichtlichkeit in einer separat angelegten Definitionsdatei.
Das Hauptprogramm kann nur durch eine Unterbrechung der Spannungsversorgung beendet werden. Eine weitere Bedingung für ein Programmende wäre fatal, da der Roboter dann nicht mehr kontrollierbar wäre. Daher ist ein Programmende in der Software nicht vorgesehen.
Die sieben wesentlichen Bestandteile des Hauptprogrammes sind:
- die Initialisierung der Hardware durch Aufrufen der einzelnen Initialisierungsfunktionen
- das Abwarten von ca. vier Sekunden, in denen die Motortreiberplatine im *Robotino*® sich initialisiert
- das Empfangen und Interpretieren eines neuen Kommandos vom DSP
- das Übertragen der Kommandos an das *ROBO Interface* (siehe 2.2.1)
- die Ausgabe von Statusinformationen auf dem LCD
- die ständige Kommunikation mit dem *Robotino*®
- die Auswertung der Lichtschrankensignale des *Robotino*® (siehe 2.1.3)

Das folgende Flussdiagramm zeigt den Ablauf des Hauptprogrammes:

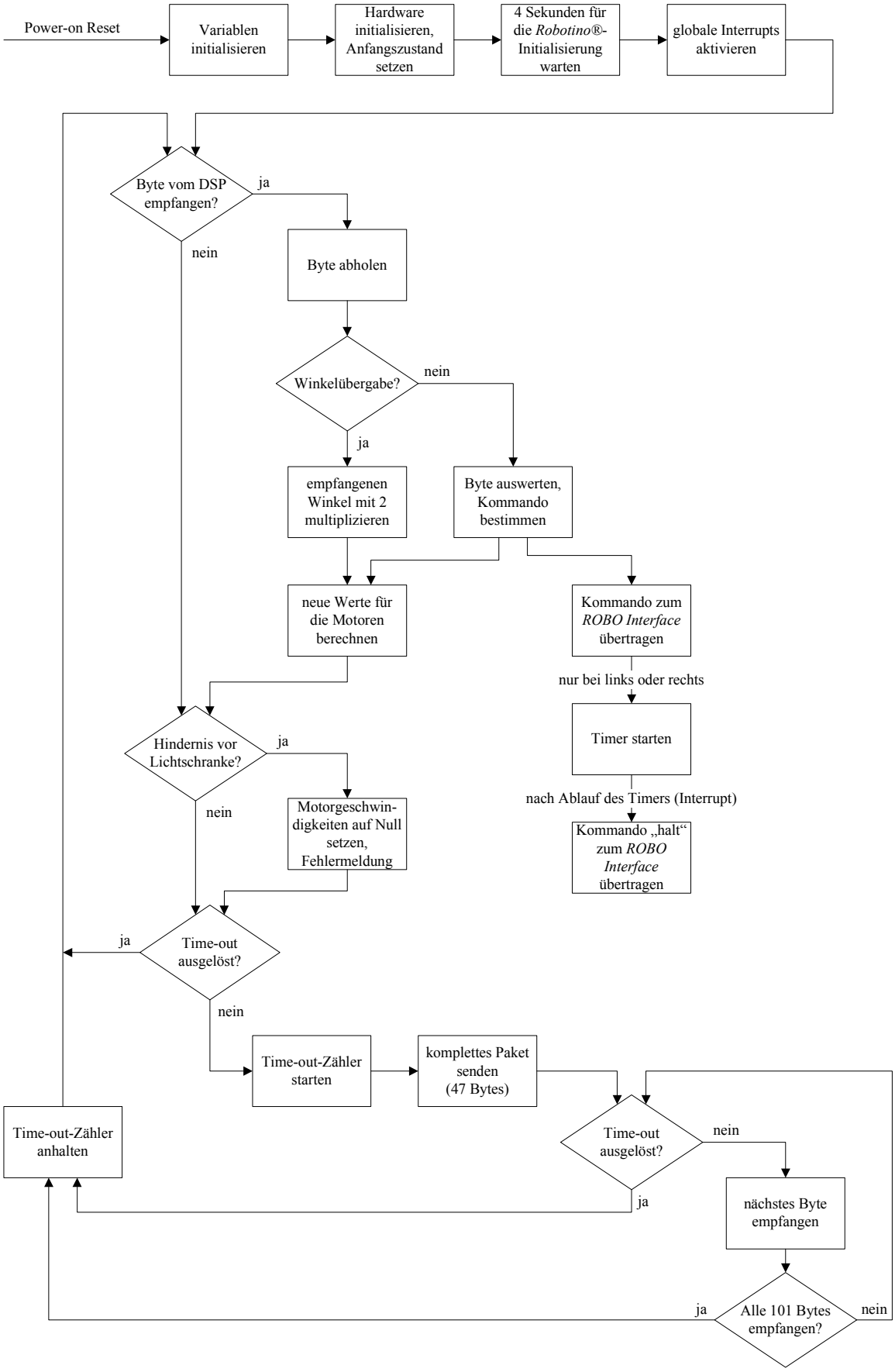

Abbildung 7-1: Flussdiagramm des Hauptprogrammes für den Mikroprozessor (*main.c*)

Das Interpretieren des vom DSP gesendeten Kommandobytes geschieht in der Hauptschleife der Funktion `main`. Dabei wird – wie auch innerhalb des DSPs – folgende Tabelle zugrunde gelegt:

Wert		Bedeutung / Funktion
hexadezimal	dezimal	
0x00 – 0xB3	0 - 179	Winkelübergabe, halber Winkelwert
0xC9	201	vorwärts, entspricht 0°
0xCA	202	rückwärts, entspricht 180°
0cCB	203	links, entspricht 270°
0xCC	204	rechts, entspricht 90°
0xCD	205	drehe (auf der Stelle)
0xCE	206	schneller
0xCF	207	langsamer
0xD0	208	halt
Statusmeldungen vom DSP, werden so auf dem Display ausgegeben		
0xFA	250	`DSP idle…`
0xFB	251	`DSP listening…`
0xFC	252	`DSP working…`
0xFD	253	`word too short`
0xFE	254	`unknown word`
0xFF	255	`UNKNOWN ERROR`

Tabelle 7-1: Interpretation des Befehlbytes

7.3 Definitionen in der *const.h*

In der Definitionsdatei *const.h* sind folgende allgemeine Definitionen vorhanden, die von verschiedenen Unterprogrammen verwendet werden:

- `NO = 0`
- `YES = 1`
- `OFF = 2`
- `ON = 3`

Für die seriellen Kommunikationen sowohl mit dem *Robotino*® als auch mit dem DSP werden je zwei Konstanten zum Berechnen der Taktteiler für das Setzen der Baudraten benötigt, wobei beide Berechnungen die `CPU_FREQUENCY` verwenden:

- `CPU_FREQUENCY = 14745600`
- `BAUDRATE_DSP = 9600`
- `BAUDRATE_ROBOTINO = 115200`

Einige Funktionen für die Ausgabe auf dem LCD arbeiten stringbasiert. Hierzu ist eine Längenangabe nötig. Diese entspricht der Länge einer Zeile des LCDs. Würde ein anderes LCD verwendet werden, müsste die Zeilenlänge angepasst werden. Ansonsten würde das Display nicht komplett genutzt oder ein Teil nicht angezeigt.

- `STRING_LENGHT = 16`

Neben dem *Robotino*® kann auch ein anderer Roboter angesteuert werden. Die Ansteuerung arbeitet nicht mit einer RS232-Schnittstelle. Hierbei werden die Kommandos binär codiert parallel an vier Pins angelegt. Die sechs codierten Kommandos werden ebenfalls als Konstanten definiert:

- `RI_STOP = 0`
- `RI_FORWARD = 1`
- `RI_BACKWARD = 2`
- `RI_LEFT = 3`
- `RI_RIGHT = 4`
- `RI_ROTATE = 5`

Die vier in den *Robotino*® eingebauten Tasten können von der Kommunikationsplatine ebenfalls ausgewertet werden. Hierzu werden sechs weitere Konstanten benötigt, die den aktuellen Zustand der Tasten beschreiben:
- `BUTTON_ENTER = 101`
- `BUTTON_LEFT = 102`
- `BUTTON_UP = 103`
- `BUTTON_DOWN = 104`
- `NO_BUTTON = 100`
- `MORE_BUTTONS = 105`

Neben den Konstanten wird die globale Variable `error_timer_interrupt` definiert, in der gespeichert wird, ob während der Kommunikation mit dem *Robotino*® ein Time-out aufgetreten ist. Der Initialisierungszustand ist `NO`. Die Variable wird nur innerhalb der Interrupt Service Routine (ISR) des Time-out-Interrupts verändert.

7.4 Die einzelnen Quellcodedateien

Im Folgenden werden die Inhalte der einzelnen Quellcodedateien genauer erklärt. Die einzelnen Funktionen werden aufgezählt, ihre Aufgabe und die Vorgehensweise erläutert.

7.4.1 Ansteuerung der Motoren (*motor.c* und *motor.h*)

Da die Definitionen der verwendeten Strukturen und Unionen für die Motorsteuerung länger und komplexer sind, werden diese in einer separaten Header-Datei abgelegt (*motor.h*).
Diese enthält zwei Definitionen für die Festlegung der Drehrichtung eines Motors und eine weitere Definition für die maximal mögliche Geschwindigkeit:
- `FORWARD = 1`
- `BACKWARD = 2`
- `MAXIMUM_VELOCITY = 100`

Des Weiteren werden zwei spezielle Datentypen aus Unionen und Strukturen definiert, die dem genauen Aufbau der Kommunikationspakete entsprechen und eine einfache und eindeutige Interpretation ermöglichen. Für das Senden bzw. Empfangen der Pakete ist es wichtig, dass auf jedes Byte einzeln per Index – wie bei einem normalen Array – zugegriffen werden kann. Dadurch wird der Sende- bzw. Empfangsvorgang erheblich vereinfacht.
Der Aufbau entspricht dem seriellen Kommunikationsprotokoll des *Robotino*® (siehe 2.1.3, [4]).

Für den Aufbau des Datentyps des Sendepaketes werden folgende Hilfsdatentypen benötigt:
- `one_motor` bildet hierbei den Anteil des Sendepaketes für einen einzelnen Motor ab. Auf jede Eigenschaft kann einfach und direkt zugegriffen werden.
- `complete_packet` stellt das komplette zu sendende Paket zusammen. Es enthält die drei Startbytes, ein weiteres Kontrollbyte, vier Motorpakete (`one_motor`) und die drei Stoppbytes.

Der verwendete Datentyp, der sowohl byteweisen als auch elementweisen Zugriff ermöglicht, heißt `send_packet`. Dabei handelt es sich um eine Union.
Für den Aufbau des Datentyps des Antwortpaketes werden folgende Hilfsdatentypen benötigt:

- `response_from_one_motor` bildet hierbei den Anteil des Antwortpaketes für einen einzelnen Motor ab. Jede Eigenschaft kann einfach und direkt ausgelesen werden.
- `complete_response` stellt das komplette zu empfangende Paket zusammen. Es enthält die drei Startbytes, die Bytes der Master-AD-Umsetzer, die Mastertime, vier Motorpakete (`response_from_one_motor`) und die drei Stoppbytes.

Der verwendete Datentyp, der sowohl byteweisen als auch elementweisen Zugriff ermöglicht, heißt `response_packet`. Dabei handelt es sich ebenfalls um eine Union.

Sowohl das Sendepaket als auch das Antwortpaket werden als globale Variable definiert:
- `send_packet cp`
- `response_packet rp`

Beide Variablen werden nicht initialisiert.

Die Datei *motor.c* beinhaltet die drei Funktionen:

- `reset_cp_values`
  Das komplette Sendepaket wird initialisiert. Zuerst werden alle Bytes gelöscht. Im Anschluss werden die Start- und Stoppbytes sowie die Regelparameter der Motoren auf die vorgegebenen Default-Werte gesetzt. Alle Bremsen und LEDs der einzelnen Motortreiber werden abgeschaltet und die Notbremse gelöscht.
  Das Antwortpaket wird ebenfalls byteweise gelöscht, jedoch nicht neu beschrieben.
  Die Funktion benötigt keine Parameter. Ein Rückgabewert wird nicht benötigt.

- `set_motor_velocity_and_direction`
  Die Funktion erwartet als Übergabeparameter die Nummer des Motors (0 – 2) sowie dessen neue Geschwindigkeit und die dazugehörige Drehrichtung. Die Funktion speichert die Geschwindigkeit dem Motor zugeordnet im Sendepaket ab. Die Drehrichtung wird umgerechnet und ebenfalls dem Motor zugeordnet gespeichert. Ein Rückgabewert wird nicht benötigt.

- `calculate_motor`
  Die Funktion erwartet als Übergabeparameter die Nummer des Motors (0 – 2), die Fahrtrichtung des Roboters als Winkel (in Grad) und die Geschwindigkeit des Roboters. Da zwei verschiedene Rückgabewerte vorgesehen sind, werden die entsprechenden Variablen als Pointer übergeben.
  Die Funktion passt den Fahrtwinkel an den jeweiligen Motor des omnidirektionalen Antriebes an (siehe 2.1.1). Mit dem angepassten Winkel und der Geschwindigkeit des Roboters wird die Geschwindigkeit und die Drehrichtung des Motors berechnet. Sowohl Motorgeschwindigkeit als auch Drehrichtung werden – über die beiden übergebenen Pointer – zurückgegeben. Diese beiden Werte werden im Hauptprogramm durch einen Aufruf von `set_motor_velocity_and_direction` in das Sendepaket übernommen.

7.4.2 Kommunikation mit dem DSP (*usart_dsp.c*)

Die Kontrolle, ob ein neues Byte vom DSP empfangen wurde, findet direkt in der Hauptschleife der `main.c` statt, da hier durch das Warten auf ein Byte die kontinuierliche Kommunikation mit dem *Robotino*® nicht unterbrochen wird. Das Abholen des neu empfangenen Bytes geschieht ebenfalls in der Hauptschleife der `main.c`. Daher enthält diese Datei nur die Initialisierungsfunktion:

- `init_usart_to_dsp`
  Die Funktion setzt alle Parameter, die zum Betrieb der RS232-Schnittstelle zum Empfangen von Bytes vom DSP nötig sind. Die Kommunikation arbeitet mit acht Datenbits, einem Stoppbit und ohne Parität. Die Datenrate (Baudrate) beträgt 9600 Bits pro Sekunde. Es werden keine Interrupts der USART-Schnittstelle des *ATmega162* eingesetzt. Daher bleiben diese abgeschaltet.
  Die Funktion benötigt keine Parameter. Ein Rückgabewert wird ebenfalls nicht benötigt.

7.4.3 Kommunikation mit dem *Robotino*® (*usart_mc.c*)

Die Datei beinhaltet die drei Funktionen:

- `init_usart_to_motor_controller`
  Die Funktion setzt alle Parameter, die zum Betrieb der RS232-Schnittstelle zur Kommunikation mit dem *Robotino*® nötig sind. Die Kommunikation arbeitet mit acht Datenbits, einem Stoppbit und ohne Parität. Die Datenrate (Baudrate) beträgt 115200 Bits pro Sekunde. Es werden keine Interrupts der USART-Schnittstelle des *ATmega162* eingesetzt. Daher bleiben diese abgeschaltet.
  Die Funktion benötigt keine Parameter. Ein Rückgabewert wird ebenfalls nicht benötigt.

- `send_byte_to_motor_controller`
  Die Funktion wartet, bis das Senderegister der USART-Schnittstelle leer ist, und sendet das zu übergebende Byte an den *Robotino*®. Ein Rückgabewert wird nicht benötigt.

- `receive_byte_from_motor_controller`
  Die Funktion wartet, bis ein neues Byte vom *Robotino*® im Empfangsregister der USART-Schnittstelle bereit liegt, holt dieses ab und gibt es als Rückgabewert zurück. Ein Parameter wird nicht benötigt.

7.4.4 Display (*lcd.c*)

Für das Ansteuern des Displays müssen eine bestimmte Signalreihenfolge und die vorgegebenen minimalen Wartezeiten unbedingt eingehalten werden, da das Display ansonsten unzuverlässig arbeiten könnte.
Die Signalverläufe des Ansteuervorganges werden im Folgenden gezeigt und erläutert:

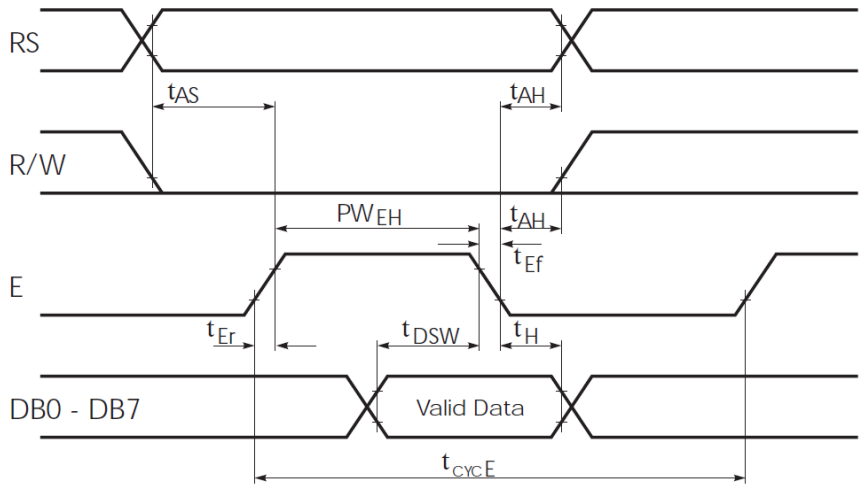

Abbildung 7-2: Timing des Displays beim Schreibzugriff [14]

Da jeder Zugriff auf das Display ein Schreibzugriff ist, wird der Lesezugriff hier nicht behandelt. Die entscheidenden Zeiten sind in der nachfolgenden Tabelle aufgeführt. Sie werden von den einzelnen Funktionen entsprechend umgesetzt und immer eingehalten. (Die Zeiten, die nicht aufgeführt sind, sind nicht relevant und müssen daher nicht weiter berücksichtigt werden.)

Bezeichnung	min. Wartezeit (nach [14])
Adress Setup Time t_{AS}	140ns
Adress Hold Time t_{AH}	10ns
Data Setup Time t_{DSW}	195ns
Data Hold Time t_{H}	20ns
Enable Puls Width, High Level PW_{EH}	450ns
Enable Cycle Time t_{CYCE}	1000ns

Tabelle 7-2: Vorgegebene Wartezeiten bei der LCD-Ansteuerung (nach [14])

Die Quellcodedatei *lcd.c* enthält neun Funktionen, von denen fünf direkte Treiberfunktionen sind. Drei weitere Funktionen greifen auf die Treiberfunktionen zu, um Daten an das Display zu senden. Keine der Funktionen gibt einen Rückgabewert zurück.

- `init_LCD`
  Die Funktion initialisiert die Portpins für die acht Datensignale und für die zwei Steuersignale am Mikroprozessor als Ausgänge. Die Datenbits werden gelöscht, E und RS werden auf low gesetzt und die Hintergrundbeleuchtung abgeschaltet.

- `LCD_background_light_on`
  Die Funktion schaltet die Hintergrundbeleuchtung des Displays ein. Bei Displays ohne Hintergrundbeleuchtung kann diese Funktion zwar benutzt werden, zeigt jedoch kein Resultat.

- `LCD_background_light_off`
  Die Funktion schaltet die Hintergrundbeleuchtung des Displays aus. Bei Displays ohne Hintergrundbeleuchtung kann diese Funktion zwar benutzt werden, zeigt jedoch kein Resultat.

- `LCD_reset`
  Die Funktion initialisiert das Display und stellt die Kommunikations- und Arbeitsparameter ein. Folgende Parameter werden der Reihe nach gesetzt (nach [14], [16]):
  - 8 Bit Datenlänge, 4 Zeilen, Schriftgröße 5x7 Pixel
  - Display einschalten, Cursor unsichtbar, kein Blinken des Cursors
  - Display löschen, Cursor in die erste Zeile und erste Spalte setzen
  - Cursor Auto-Inkrement

  Beide eingesetzten Displays verwenden für die erste und zweite Zeile denselben Adressraum (0x00 – 0x0F und 0x40 – 0x4F, [14]). Daher kann das Display immer als Display der Größe 4x16 initialisiert werden.

- `LCD_send_byte`
  Die Funktion sendet ein Datenbyte direkt an das Display. Eine Anpassung von Sonderzeichen findet nicht statt. Diese Funktion wird nur von `LCD_send_character` aufgerufen.

- `LCD_send_character`
  Die Funktion übersetzt die Sonderzeichen ä, ö, ü, Ä, Ö, Ü und ß in die Zeichentabelle des LCDs [14], [16]. Liegt kein Sonderzeichen vor, wird das Byte ohne diese Anpassung direkt mit der Funktion `LCD_send_byte` an das Display gesendet.

- `LCD_position`
  Die Funktion setzt den Cursor des LCDs an die zu übergebende Stelle. Übergeben werden müssen die Spalte (X-Koordinate) und die Zeile (Y-Koordinate). Das Zählen beginnt wie üblich bei 0.

- `LCD_send_string`
  Die Funktion gibt einen übergebenen String zeichenweise über die Funktion `LCD_send_character` auf dem Display aus. Die Ausgabe wird entweder durch das Erreichen des Stringendezeichens `\0` oder durch Erreichen der in der *const.h* festgelegten Stringlänge `STRING_LENGTH` beendet.
  Die Startposition der Ausgabe muss ebenfalls übergeben werden. Bevor die Ausgabe des Strings beginnt, wird der Cursor mit Hilfe der Funktion `LCD_position` positioniert. Ab dieser Position beginnt die Ausgabe. Dabei muss beachtet werden, dass der String über den Rand des Displays hinausgeschrieben werden kann, wenn der String nicht durch `\0` abgeschlossen ist und die X-Koordinate nicht dem ersten Zeichen der Displayzeile entspricht.

- `LCD_send_number`
  Die Funktion zerlegt die übergebene `unsigned short`-Zahl in Hunderter, Zehner und Einer. Dabei werden die einzelnen Ziffern umgehend in ein ASCII-Zeichen übersetzt und mit `LCD_send_character` zum Display übertragen. Die Zahl wird immer rechtsbündig ausgegeben.
  Beim Aufruf der Funktion muss die Startposition der Ausgabe übergeben werden. Diese wird direkt an `LCD_position` weitergeleitet. Die Ausgabe führender Nullen muss aktiviert oder deaktiviert werden.

7.4.5 LEDs (*led.c*)

Die Ansteuerung der beiden LEDs ist sehr einfach. Für eine vereinfachtere Programmierung werden trotzdem Funktionen geschrieben, da hierdurch die Lesbarkeit des Quellcodes erleichtert wird. Alle Funktionen benötigen keine Parameter und geben keinen Rückgabewert zurück.

Die Datei beinhaltet die fünf Funktionen:
- `init_leds`
  Die beiden Pins, an die die LEDs angeschlossen sind, werden als Ausgänge definiert. Beide LEDs werden abgeschaltet.

- `error_LED_on`
  Die Funktion schaltet die Error LED (rot) ein.

- `error_LED_off`
  Die Funktion schaltet die Error LED (rot) aus.

- `response_LED_on`
  Die Funktion schaltet die Antwort LED (gelb) ein.

- `response_LED_off`
  Die Funktion schaltet die Antwort LED (gelb) aus.

7.4.6 Taster am Roboter (*buttons.c*)

Die vier in den Deckel des *Robotino*® eingebauten Taster können von dem Mikroprozessor ausgewertet werden. In der aktuellen Realisierung werden die Tasten zwar initialisiert aber nicht weiter verwendet, da die Tasten durch die Unterbringung der Kommunikationsplatine außerhalb des *Robotino*® nicht mehr angeschlossen werden können. Für mögliche spätere Erweiterungen sind folgende Funktionen bereits enthalten:

- `init_buttons`
  Die vier Pins, an die die Taster angeschlossen sind, werden als Eingänge definiert. Die Initialisierungsfunktion benötigt keine Parameter und gibt keinen Rückgabewert zurück.

- `buttons`
  Die Funktion prüft und bewertet den Gesamtzustand der vier Pins. Der Rückgabewert wird mit den in der *const.h* definierten Konstanten codiert. Sind mehrere Tasten gedrückt, werden nicht alle Tasten ausgewertet, sondern der Wert `MORE_BUTTONS` zurückgegeben.

7.4.7 Fehler-Timer (*error_timer.c*)

Wird die Kommunikation mit dem *Robotino*® unterbrochen, so bleibt dieser nach einer Sekunde stehen und sendet keine weiteren Bytes über die serielle Schnittstelle. Damit die Hauptschleife nicht beim Warten auf Bytes vom *Robotino*® stecken bleibt und weiterhin die empfangenen Kommandos vom DSP empfangen, interpretieren und die entsprechende Mitteilung auf dem LCD ausgeben kann, wird die Möglichkeit eines Time-out benötigt. Der Time-out-Zähler wird vor dem Senden des Datenpaketes an den *Robotino*® gestartet und im Anschluss an das letzte vom *Robotino*® empfangene Byte gestoppt.
Sollte ein Time-out auftreten, so wird der Timer innerhalb der Interrupt Service Routine gestoppt. Durch den Einsatz der globalen Variable `error_timer_interrupt`, die in der *const.h* definiert ist und während der Hauptschleife laufend abgefragt wird, kann die Kommunikation mit dem *Robotino*® abgebrochen werden. Das Hauptprogramm arbeitet normal weiter, jedoch wird die Kommunikation komplett ausgelassen. Der Roboter und die Kommunikationsplatine müssen abgeschaltet und nach einer Wartezeit von mindestens zehn Sekunden wieder eingeschaltet werden.
Der hierfür verwendete Timer 1 ist ein 16-Bit-Timer, d.h. nach Erreichen des Wertes 65535 läuft der Timer über und der Overflow Interrupt wird ausgelöst [11].
Die Zeitdauer des Time-out kann mit nachfolgender Gleichung berechnet werden. Dabei sind N der verwendete Taktteiler (256) und f_{CLK} die Frequenz des verwendeten Quarzes.

$$T_{Time-out} = \frac{1}{f_{CLK}} * N * 2^{16} = \frac{1}{14{,}7456MHz} * 256 * 65536 = 1{,}137s$$

Gleichung 7-1: Zeitdauer des Time-out (*error_timer.c*)

Somit ist die tatsächliche Zeitdauer etwas länger als eine Sekunde, was an dieser Stelle aber nicht relevant ist.

Die Datei beinhaltet die drei Funktionen:

- `init_error_timer`
  Der Timer 1 wird initialisiert. Er kann an zwei Pins PWM-Signale (engl.: Pulse Width Modulation) ausgeben. Diese Funktion wird nicht genutzt und daher abgeschaltet. Der Timer wird für einen normalen Zählbetrieb eingestellt, jedoch noch nicht gestartet. Der Zählerstand und alle Output Compare Register werden gelöscht. Nur der Timer Overflow Interrupt wird zugelassen, das zugehörige Flag gelöscht.

- `start_error_timer`
  Die Funktion löscht den aktuellen Zählerstand des Timers und startet diesen mit dem Taktteiler 256.

- `stop_error_timer`
  Die Funktion stoppt den Timer 1 und löscht den Zählerstand.

Zusätzlich ist die Interrupt Service Routine enthalten. Diese wird ausgeführt, sobald das Overflow Flag des Timers 1 gesetzt wird.

- `ISR (TIMER1_OVF_vect)`
  Innerhalb der Interrupt Service Routine wird der Timer gestoppt. Der Zählerstand wird gelöscht, das Interrupt Flag zurückgesetzt. Auf dem LCD wird, über zwei Displayzeilen verteilt, die Fehlermeldung ausgegeben: `communication lost -> repower`. Die globale Variable `error_timer_interrupt` wird auf `YES` gesetzt.

7.4.8 Wartefunktion (*wait.c*)

Für die Ansteuerung des Displays sind Wartezeiten erforderlich. Die längste Wartezeit beträgt eine Millisekunde. Um den Programmieraufwand übersichtlich zu halten, wird nur diese eine Wartezeit implementiert.
Der hierfür verwendete Timer 0 ist ein 8-Bit-Timer [11]. Der Wert, der in das Output Compare Register geschrieben werden muss, wird mit nachfolgender Gleichung berechnet. Dabei sind N der verwendete Taktteiler (64), f_{CLK} die Frequenz des verwendeten Quarzes und t die vorgegebene Wartezeit in Sekunden.

$$OCR0 = t * \frac{f_{CLK}}{N} = 0{,}001s * \frac{14{,}7456MHz}{64} = 230{,}4 \approx 230$$

Gleichung 7-2: Wert für das Output Compare Register der Funktion `wait_one_ms`

Da für den Funktionsaufruf, das Starten des Timers und die Schleife zum Abfragen des Output Compare Flags weitere Prozessortakte benötigt werden, wird der genau berechnete Wert für das Output Compare Register abgerundet. Somit wird die Wartezeit von einer Millisekunde eingehalten.

Die Datei beinhaltet die zwei Funktionen:

- `init_wait_timer`
  Der Timer 0 wird initialisiert. Er kann an einem Pin ein PWM-Signal ausgeben. Diese Funktion wird nicht genutzt und daher abgeschaltet. Der Timer wird für einen normalen Zählbetrieb eingestellt, jedoch noch nicht gestartet. Der Zählerstand wird gelöscht und das Output Compare Register auf den zuvor berechneten Wert 230 gesetzt. Ein Interrupt wird nicht verwendet.

- `wait_one_msec`
  Die Funktion löscht den Zählerstand, setzt das Output Compare Register auf den zuvor berechneten Wert 230 und startet den Timer mit dem gewählten Taktteiler 64. Während des Zählvorganges wird das Output Compare Flag kontinuierlich abgefragt. Ist das Flag gesetzt, werden der Timer gestoppt, der Zählerstand gelöscht und die Funktion beendet.

7.4.9 Kommunikation mit dem *ROBO Interface* (*robo_interface.c*)

Für die Kommunikation mit dem *ROBO Interface* werden vier Portpins benötigt und das entsprechende Kommando parallel angelegt (siehe 2.2.1).
Da der Roboter wie ein Auto aufgebaut ist, würde er beim Kommando „links“ oder „rechts“ solange im Kreis fahren, bis das nächste Kommando empfangen wird. Daher wird ein Timer eingesetzt, der – nachdem das Kommando „links“ oder „rechts“ zum *ROBO Interface* übertragen wurde – nach seinem Ablaufen automatisch das Kommando „halt“ überträgt.
Der hierfür verwendete Timer 3 ist ein 16-Bit-Timer [11]. Die Zeitdauer wird beim Starten des Timers in Millisekunden eingestellt. Somit kann der Winkel, um den sich der Roboter nach rechts oder nach links dreht, eingestellt werden. Der Wert, mit dem die Zeit von einer Millisekunde multipliziert und in das Output Compare Register geschrieben werden muss, wird mit nachfolgender Gleichung berechnet. Dabei sind N der verwendete Taktteiler (1024) und f_{CLK} die Frequenz des verwendeten Quarzes.

$$OCR3A = t * \frac{f_{CLK}}{N} = 0{,}001s * \frac{14{,}7456MHz}{1024} = 14{,}4 \approx 14$$

Gleichung 7-3: Grundwert für das Output Compare Register (*robo_interface.c*)

Da für den Funktionsaufruf, das Starten des Timers und die Schleife zum Abfragen des Output Compare Flags weitere Prozessortakte benötigt werden, wird der genau berechnete Wert für das Output Compare Register abgerundet. Somit wird die Wartezeit von einer Millisekunde eingehalten. Um mehrere Millisekunden warten zu können, wird der berechnete Wert für das Output Compare Register mit der Anzahl der Millisekunden multipliziert.

Die Datei beinhaltet die drei Funktionen:
- `init_robo_interface`
  Die vier verwendeten Pins werden als Ausgänge definiert und das Kommando „halt“ angelegt (0x00).
  Der Timer 3 wird initialisiert. Er kann an zwei Pins PWM-Signale ausgeben. Diese Funktion wird nicht genutzt und daher abgeschaltet. Der Timer wird für einen normalen Zählbetrieb eingestellt, jedoch noch nicht gestartet. Der Zählerstand und alle Output Compare Register werden gelöscht. Nur der Output Compare A Interrupt wird zugelassen, das zugehörige Flag gelöscht.

- `robo_interface_send_command`
  Die Funktion setzt den Zustand des Kommandos an den vier für die Kommunikation mit dem *ROBO Interface* vorgesehenen Pins. Da diese nicht fortlaufend an einem Port untergebracht sind, muss das zu übergebende Kommando auf die beiden verwendeten Ports verteilt werden. Als Erstes werden alle vier Pins gelöscht und im Anschluss der neue Zustand gesetzt. Um sicherzustellen, dass die anderen Pins der Ports nicht mit beschrieben werden, wird das Kommando auf zwei Hilfsvariablen aufgeteilt und durch Bitoperationen entsprechend angepasst in die Portregister geschrieben.

- `start_left_rigth_timer`
  Die Funktion löscht den aktuellen Zählerstand des Timers. Die übergebene Wartezeit (in Millisekunden) wird mit dem mit Gleichung 7-3 berechneten Grundwert 14 multipliziert. Das Ergebnis wird in das Output Compare A Register geschrieben und der Timer mit dem Taktteiler 1024 gestartet. Zum Anzeigen des aktiven Timers erscheint auf dem Display in der zweiten Zeile rechts außen ein Punkt. Dieser wird beim Auslösen des Output Compare A Interrupts wieder gelöscht.

Zusätzlich ist die Interrupt Service Routine enthalten. Diese wird ausgeführt, sobald das Output Compare Flag A des Timers 3 gesetzt wird.

- `ISR (TIMER3_COMPA_vect)`
  Innerhalb der Interrupt Service Routine wird der Timer gestoppt. Der Punkt auf dem Display wird wieder gelöscht. Um den Roboter anzuhalten, wird mit der Funktion `robo_interface_send_command` das Kommando „halt“ an das *ROBO Interface* übertragen. Der Zählerstand wird gelöscht, das Interrupt Flag zurückgesetzt. Eine globale Variable wird nicht verändert. Auf den *Robotino*® hat dieser Vorgang keinen Einfluss.

8 Implementierung des Simulationsmodells auf dem DSP

Die Implementierung der Spracherkennung auf dem DSP erfolgt nach einem ganz anderen Prinzip als die Implementierung in *Matlab*®. Prinzipiell werden auf dem DSP dieselben Verarbeitungsschritte wie im *Matlab*®-Modell ausgeführt, jedoch sind grundlegende Besonderheiten zu beachten.

8.1 Verwendete Hardware

Eingesetzt wird ein *D.Module.C6713* des Herstellers *D.SignT GmbH & Co. KG*. Dieses Board enthält einen digitalen Signalprozessor *TMS320C6713* des Herstellers *Texas Instruments* [7]. Der DSP arbeitet mit einer Taktfrequenz von 300 MHz. Bei dem Prozessor handelt es sich um einen 32 Bit Floating-Point-Prozessor. Mit auf dem Modul untergebracht sind 16 MB externer SDRAM, 512 KB externer SBSRAM und 2 MB externer Flash. Aus Gründen der Geschwindigkeit sollten Variablen, die wegen ihrer Größe nicht in den internen RAM des DSPs (IRAM) passen, im SBSRAM abgelegt werden, da dieser einen schnelleren Zugriff als der SDRAM erlaubt. Zusätzlich ist auf dem Modul eine RS232-Schnittstelle integriert. Programmiert wird der Prozessor über eine JTAG-Schnittstelle.
Zum Konfigurieren und Starten der Hardware werden einige Funktionen mitgeliefert. Um diese verwenden zu können, muss daher die Datei *bios.h* mit in die DSP-Software eingebunden werden. Bevor die Hardware nicht konfiguriert ist, kann z. B. nicht auf den SBSRAM zugegriffen werden.
Das Board ist so aufgebaut, dass es mit weiteren Modulen zusammengesteckt werden kann. Hierzu werden alle wichtigen Signale, Busse und Schnittstellen über mehrere Stiftleisten nach außen geführt. Wichtig dabei ist, dass sich das *D.Module.C6713* beim Stapeln mehrerer Platinen oben befindet, da ansonsten der Anschluss der JTAG-Schnittstelle nicht mehr erreicht werden könnte.
Beide Module werden zusammen auf die Kommunikationsplatine aufgesteckt und mit vier Schrauben fixiert. Dieser „Platinenstapel“ ist im Folgenden abgebildet.

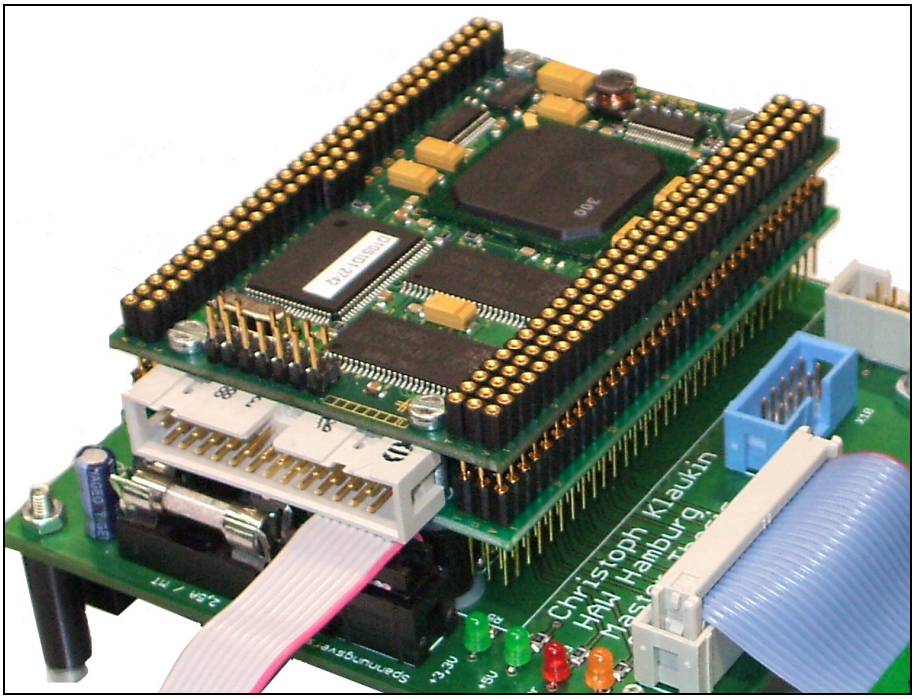

Abbildung 8-1: *D.Module.C6713*, *D.Module.PCM3003* und Kommunikationsplatine

Die unterste Platine ist hier die selbst entwickelte, gefertigte Kommunikationsplatine. Die mittlere Platine ist das *D.Module.PCM3003* und die obere Platine das *D.Module.C6713*.
Da das *D.Module.C6713* über keine eigenen Audioeingänge und -ausgänge verfügt, wird hier ein *D.Module.PCM3003*, ebenfalls vom Hersteller *D.SignT GmbH & Co. KG*, eingesetzt. Das Board enthält vier Audio-Codecs *PCM3003*, die insgesamt acht Audioeingänge und acht Audioausgänge mit einer Auflösung von je 16 Bit zur Verfügung stellen [6]. Die Samplefrequenz wird über drei Lötbrücken eingestellt. Dabei stehen sechs verschiedene Frequenzen, die für alle Ein- und Ausgänge zusammen gelten, zur Auswahl.
Das *D.Module.C6713* und das *D.Module.PCM3003* kommunizieren über den McBSP0 und den McBSP1 des DSPs miteinander.

8.2 Die grundlegende Struktur der DSP-Software

Der größte Teil des Ablaufes der DSP-Software ist direkt in der Hauptdatei *word_recognition.c* enthalten. Um eine höhere Verarbeitungsgeschwindigkeit zu erreichen, werden viele in *Matlab*® als separate Funktionen programmierte Programmteile direkt in das Hauptprogramm `main` eingebaut. Nur die Berechnung der LPC-Koeffizienten und das DTW werden in zwei separate Funktionen und Dateien ausgelagert.
Um auch bei größeren Variablen keine Speicherprobleme zu bekommen, werden diese in den externen SBSRAM ausgelagert. Dabei handelt es sich ausschließlich um globale Variablen. Die Größe des Stack und des Heap werden in der *word_recognition.cmd* auf jeweils 0x1500 vergrößert, damit auch für lokale Variablen genügend Speicherplatz zur Verfügung steht.
Der prinzipielle Ablauf des Hauptprogramms wird in dem folgenden Diagramm dargestellt. Eine nähere Erläuterung relevanter Abschnitte erfolgt im weiteren Verlauf des Kapitels.

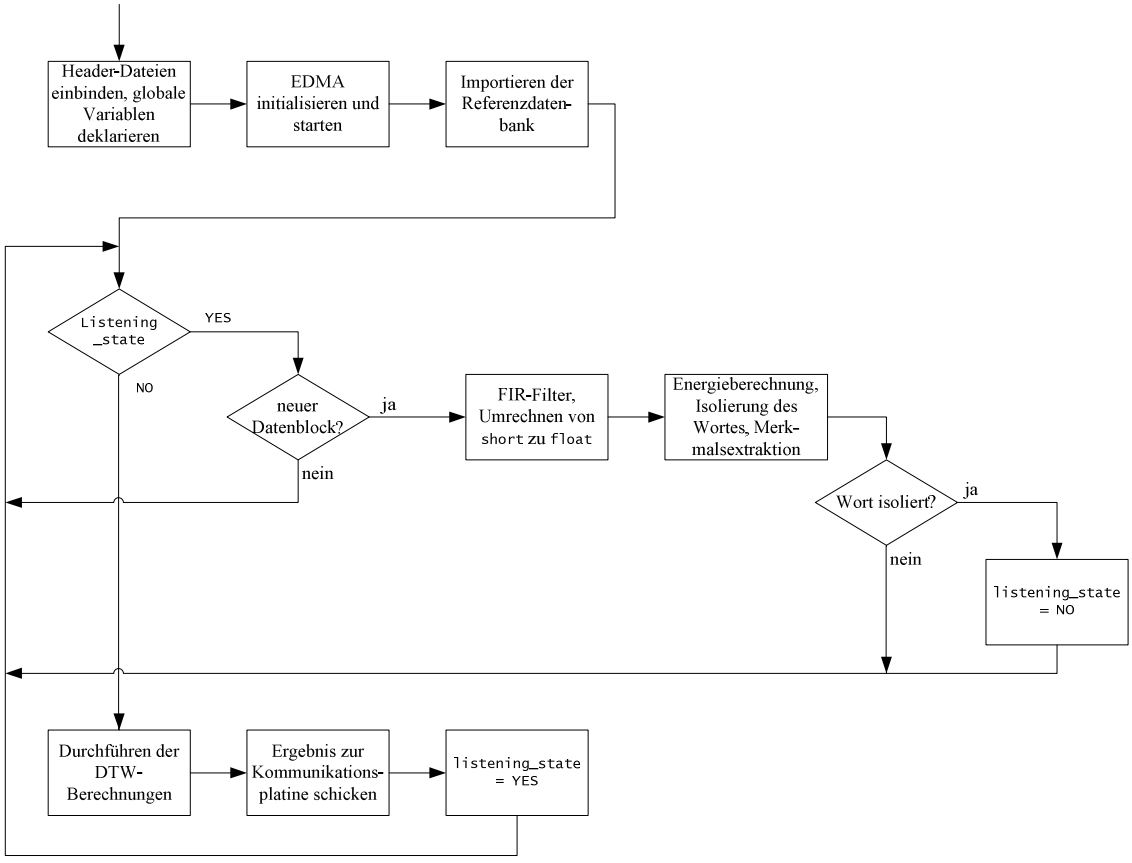

Abbildung 8-2: Ablauf des Hauptprogrammes auf dem DSP

Das Ergebnis der Spracherkennung soll möglichst schnell ermittelt werden. Außerdem soll nicht wie in *Matlab*® eine mehrere Sekunden lange Aufnahme durchgeführt werden, die im Nachhinein untersucht wird. Stattdessen soll kontinuierlich „gelauscht“ und unmittelbar analysiert werden.
Auf dem DSP wird nur die Spracherkennung ohne die Möglichkeit zur Aufnahme einer Referenzdatenbank implementiert. Dabei werden alle Berechnungen mit Ausnahme des FIR-Bandpassfilters im Datenformat `float` implementiert.
Um nicht die komplette Initialisierung der Hardware und des EDMA neu programmieren zu müssen, wird eine bei den Modulen mitgelieferte Demosoftware, die die acht Eingangskanäle unter Einsatz des EDMA direkt an die acht Ausgänge kopiert, verwendet und entsprechend erweitert.

8.3 Referenzdatenbank

Die Referenzdatenbank wird mit dem DSP nicht neu aufgenommen, sondern aus *Matlab*® exportiert. Hierzu wird die Funktion *mat_to_txt.m* verwendet. Diese befindet sich im Verzeichnis des *Matlab*®-Simulationsmodells. Mit diesem *Matlab*®-Programm wird ein auswählbarer Sprecher aus der Referenzdatenbank (*mat*-Datei, siehe 4.5) in die Textdatei *speaker_reference_database.txt* exportiert. Dabei wird die Anzahl der Wörter in der ersten Zeile der Textdatei abgespeichert. Die Länge der einzelnen Wörter wird als Blockanzahl – durch Semikolons voneinander getrennt – für jedes einzelne Wort in die zweite Zeile der Textdatei gespeichert. Die einzelnen LPC-Koeffizienten werden der Reihe nach als Fließkommazahlen mit zehn Nachkommastellen abgelegt, wobei in jeder Zeile ein Wert gespeichert wird. Ein Semikolon kommt hier nicht zum Einsatz.
Die so exportierte Referenzdatenbank *speaker_reference_database.txt* muss in das Verzeichnis des DSP-Projektes kopiert werden. Sie wird nach jedem Programmstart von der DSP-Software importiert und in die globale Variable `reference_data`, die im externen SBSRAM liegt, geschrieben. Danach steht diese für die Spracherkennung mit dem DSP zur Verfügung.

8.4 Blockbasierte Aufnahme der Samples per EDMA

Die insgesamt vier auf dem *D.Module.PCM3003* enthaltenen Audio-Codecs sind über zwei McBSP-Schnittstellen mit dem DSP verbunden. Auf diese Weise werden pro McBSP-Interrupt vier Audiokanäle in jede Richtung übertragen, entweder die Kanäle 0 und 2 über McBSP0 und die Kanäle 4 und 6 über McBSP1 oder die Kanäle 1 und 3 über McBSP0 und die Kanäle 5 und 7 über McBSP1. Würde die Software direkt mit diesen Interrupts arbeiten, würde das Programm nur solange richtig arbeiten, bis eine ungerade Anzahl Interrupts nicht verarbeitet werden würde. In diesem Falle würde sich die Kanalzuordnung verschieben. Kanal 0 würde zu Kanal 1 und umgekehrt, Kanal 2 würde zu Kanal 3 und umgekehrt usw.
Daher bietet sich hier der Einsatz des EDMA an. Die McBSP-Interrupts werden direkt an den EDMA weitergeleitet und im Hintergrund bearbeitet. Hierzu werden zwei Ping-Pong-Puffer, die aus jeweils zwei gleichwertigen Datenblöcken bestehen, eingesetzt. Der Puffer `adcbuffer` ist für die Samples der Audioeingänge vorgesehen, der Puffer `dacbuffer` für die Samples der Audioausgänge.
Während jeweils eine Hälfte der beiden Puffer vom EDMA bearbeitet wird, steht die andere Hälfte dem normalen Programmablauf zur Verfügung.

Der EDMA übernimmt folgende Funktionen:

- Der EDMA kopiert alle acht Eingangskanäle in einer festgelegten Reihenfolge in das Feld `adcbuffer`. Das `short`-Feld verfügt über drei Dimensionen. Somit ist es möglich, direkt auf einen Kanal und ein bestimmtes Sample zuzugreifen.
- Der EDMA gibt alle acht Ausgangskanäle in einer festgelegten Reihenfolge aus dem Feld `dacbuffer` aus. Das `short`-Feld verfügt über drei Dimensionen. Somit ist es möglich, direkt auf einen Kanal und ein bestimmtes Sample zuzugreifen.
- Beide Felder (`adcbuffer` und `dacbuffer`) sind als Ping-Pong-Puffer ausgeführt. Der EDMA kopiert solange Eingangs- bzw. Ausgangssamples, bis der jeweilige Puffer komplett bearbeitet ist. Erst dann wird ein EDMA-Interrupt ausgelöst. Die Größe der Puffer wird durch die Konstante `BLOCKSIZE` auf 240 eingestellt. Das bedeutet, dass für jeden Kanal 240 Samples aufgenommen werden und erst dann ein Interrupt ausgelöst wird.
- In der Interrupt Service Routine des EDMA-Interrupts wird die Nummer des soeben vom EDMA fertig bearbeiteten Datenblockes in die globale Variable `block` geschrieben. Diese ist entweder 0 oder 1. Der soeben vom EDMA fertig bearbeitete Datenblock steht bis zum nächsten EDMA-Interrupt dem normalen Programmablauf zur Verfügung. Der andere Datenblock, der bisher für den normalen Programmablauf zur Verfügung stand, wird jetzt vom EDMA bearbeitet. Bei jedem EDMA-Interrupt tauschen die Datenblöcke ihre Funktion.

Die `BLOCKSIZE` ist deshalb auf den Wert 240 eingestellt, da dies genau der von der Spracherkennung verwendeten Blockgröße entspricht. Somit vereinfacht sich die weitere Vorgehensweise.
Obwohl nur Kanal 0 verwendet wird, müssen alle acht Kanäle eingelesen bzw. ausgegeben werden und die Puffer eine entsprechende Größe aufweisen.
Durch eine `BLOCKSIZE` von 240 Samples und einer Samplefrequenz von 8 KHz ergeben sich ca. 33,3 EDMA-Interrupts pro Sekunde.

8.5 Bandpassfilter und Konvertieren zu Fließkommazahlen

Auf dem Signalprozessor wird dasselbe FIR-Bandpassfilter wie bei dem Simulationsmodell mit *Matlab*® eingesetzt. Dieses Filter wird in Kapitel 4.3.1 spezifiziert. Während des dort beschriebenen Filterdesigns mit der Datei *bandgap.m* werden die berechneten Filterkoeffizienten in einer *mat*-Datei für das Simulationsmodell gespeichert und im Anschluss in der Datei *bandgap.h* abgelegt. Diese Datei wird in das Projektverzeichnis des DSP-Programmes kopiert und in das Programm eingebunden.
Die Funktion zum Filtern des Audiosignals wird während dieser Arbeit nicht selbst programmiert. Es handelt sich dabei um die in Assembler programmierte Funktion `FIR_filter`, die in der Datei *FIR_filter_asm.asm* abgespeichert ist. Diese Funktion wurde von Herrn Prof. Dr.-Ing. Ulrich Sauvagerd programmiert und für die Praktika im Rahmen der Vorlesungen zur digitalen Signalverarbeitung zur Verfügung gestellt.

Die Filterung geschieht in einer Schleife. Dabei wird jedes Sample des Kanals 0 aus dem aktuellen Ping-Pong-Puffer `adcbuffer` gelesen und gefiltert. Wichtig ist, dass die Inhalte der Verzögerungsglieder gespeichert und die entsprechenden Werte beim nächsten Sample weiter verwendet werden.
Direkt nachdem ein Sample gefiltert wurde, wird das Ergebnis in eine `float`-Zahl umgerechnet. Nach Abschluss der Filterung eines Datenblockes stehen die gefilterten Daten im `float`-Format in dem Feld `filtered_float_data` für eine weitere Verarbeitung bereit.

Der Amplitudengang des Filters wird mit einem *R&S®UPV Audio Analyzer* untersucht. Für die Messung wird die Software des DSPs angepasst, da bei der Spracherkennung keine Audioausgänge vorgesehen sind. Als Audioeingang wird der Eingangskanal 0 des *D.Module.PCM3003* verwendet. Das unbearbeitete Eingangssignal wird direkt an den Ausgangskanal 1 kopiert und das mit dem Bandpassfilter bearbeitete Eingangssignal über den Ausgangskanal 0 des *D.Module.PCM3003* ausgegeben.
Der gemessene Amplitudengang ist in der nachfolgenden Abbildung dargestellt. Dabei gibt die blaue Messkurve den Amplitudengang des Bandpassfilters wieder, während die rote Kurve das unverarbeitete, durchgeschliffene Eingangssignal wiedergibt. Um Übersteuerungen zu vermeiden, wird mit einer effektiven Generatorspannung von 500mV gemessen. Dieser Wert wird auch als Referenzwert für den Analyzer verwendet, so dass sich im Durchlassbereich ca. 0 dB einstellen.

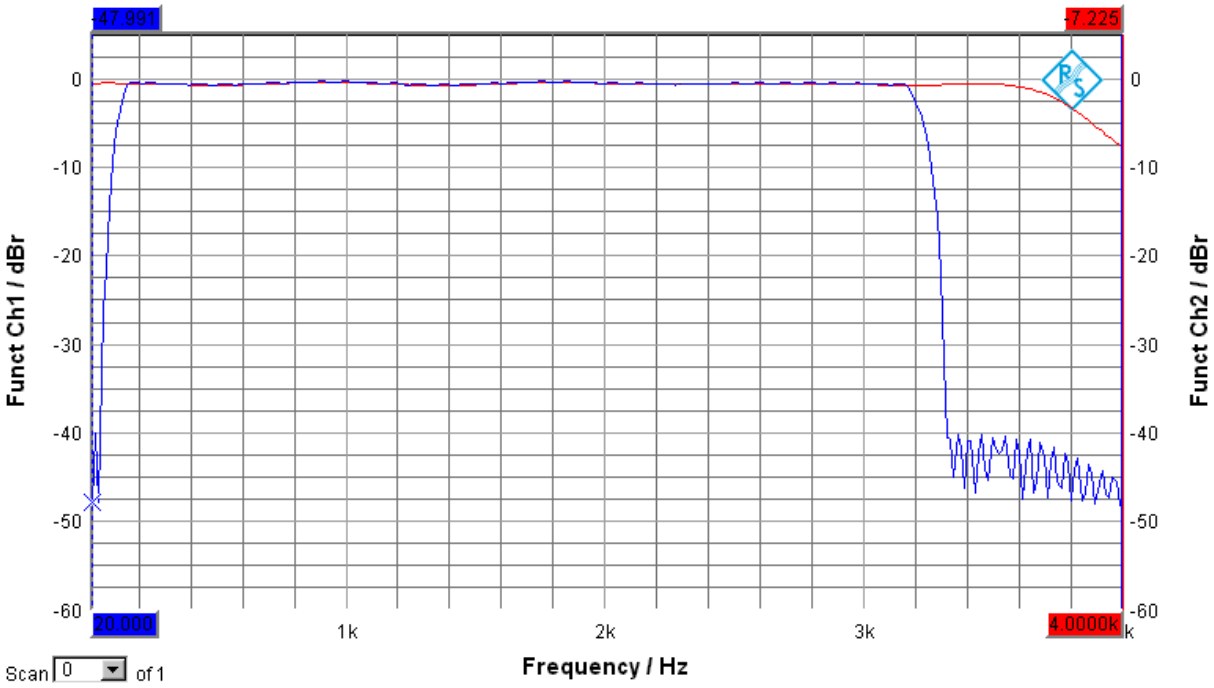

Abbildung 8-3: Amplitudengang des Bandpassfilters bei der DSP-Implementierung

Das Bandpassfilter erfüllt die in Kapitel 4.3.1 aufgestellten Spezifikationen.
Bei der roten Messkurve ist der Einfluss des Tiefpassfilters der Audio-Codecs *PCM3003* ab ca. 3,6 KHz zu erkennen. Dieses Tiefpassverhalten ist auch bei dem Amplitudengang erkennbar.
Im Anschluss der Messung werden die beiden für die Ausgabe benötigten Programmzeilen aus der DSP-Software entfernt.

8.6 Ping-Pong-Puffer zur Blockbildung, Hanning-Fenster, Energieberechnung

Die Programmierung der Energieberechnung sowie der Isolierung des Wortes und der Merkmalsextraktion durch die Berechnung der LPC-Koeffizienten ist aufwendig, da auf dem DSP kein über einen längeren Zeitraum aufgenommener, zusammenhängender Datensatz existiert.
Während eines EDMA-Interrupts, der einen neuen Datenblock mit den neuen, zuletzt aufgenommenen Samples liefert, müssen insgesamt drei Datenblöcke untersucht werden.

Hierzu ist es erforderlich, dass der Datenblock des vorherigen EDMA-Interrupts ebenfalls zur Verfügung steht. Nur dann können die Datenblöcke – wie benötigt – überlappen.
Dieses wird durch einen weiteren Ping-Pong-Puffer realisiert. Dazu werden drei Variablen benötigt, das zweidimensionale Feld `double_buffer[2][BLOCKSIZE]` und die beiden Zeiger `filtered_float_data` und `old_data_block`.
Zu Beginn werden dem Zeiger `filtered_float_data` die Adresse `&double_buffer[0][0]` und dem Zeiger `old_data_block` die Adresse `&double_buffer[1][0]` zugewiesen.
Im Anschluss an jeden Durchlauf der dreistufigen Blockbildung werden den beiden Zeigern die jeweils andere Adresse des Feldes `double_buffer` zugewiesen. Auf diese Weise entfällt ein Kopiervorgang. Diese gewählte Vorgehensweise ist einfacher und schneller als das Kopieren.
Bei der im vorherigen Kapitel beschriebenen Umrechnung der gefilterten Samples in das `float`-Format werden die umgerechneten Samples direkt in das Feld, das der Zeiger `filtered_float_data` adressiert, geschrieben.

Die bei der Blockbildung verwendete Zuordnung der Samples aus `old_data_block` und `filtered_float_data` ist in der folgenden Abbildung dargestellt.

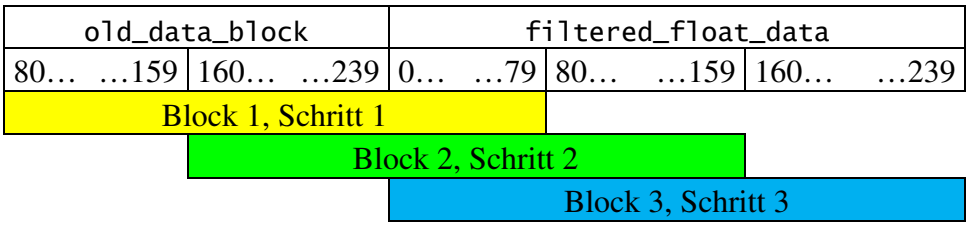

Abbildung 8-4: Blockbildung bei der DSP-Implementierung

In der Abbildung wird die Zählung des Feldindizes bei Null begonnen, wie es bei der Programmierung in C üblich ist.

1. Schritt:
   Die Samples 80 bis 239 aus `old_data_block` werden in den temporären Datenblock `data_for_lpc_calculation` an die Samplepositionen 0 bis 159 kopiert. Dabei wird keine direkte Kopie vorgenommen, sondern das in `data_for_lpc_calculation` zu speichernde Sample wird mit dem zugehörigen Wert des Hanning-Fensters multipliziert. Außerdem wird bei jedem einzelnen Sample unter Verwendung der Gleichung 4-1 die quadratische Energie des Blockes sampleweise berechnet. Bei der Energieberechnung werden die unbearbeiteten Samples aus `old_data_block` verwendet.
   Im Anschluss werden die Samples 0 bis 79 aus `filtered_float_data` in den temporären Datenblock `data_for_lpc_calculation` an die Samplepositionen 160 bis 239 kopiert. Auch hier erfolgt eine direkte Multiplikation mit dem zugehörigen Wert des Hanning-Fensters. Die Berechnung der quadratischen Energie des Blockes wird fortgesetzt. Bei der Energieberechnung werden auch hier die unbearbeiteten Samples aus `filtered_float_data` verwendet.
   Nach Abschluss der beiden Kopiervorgänge wird das Ergebnis der quadratischen Energie des Blockes logarithmiert.
   Somit liegen die logarithmierte quadratische Energie des Blockes und der bereits mit dem Hanning-Fenster bearbeitete Datenblock, aus dem direkt die LPC-Koeffizienten berechnet werden könnten, vor.
   Direkt im Anschluss folgt ein einzelner Durchgang des modifizierten Zustandsautomaten zur Isolierung des Wortes. Dieser wird im nächsten Kapitel beschrieben.

2. Schritt:
   Dieser Schritt ist aufgebaut wie der erste, jedoch werden hier die Samples 160 bis 239 aus `old_data_block` in den temporären Datenblock `data_for_lpc_calculation` an die Samplepositionen 0 bis 79 und die Samples 0 bis 159 aus `filtered_float_data` an die Samplepositionen 80 bis 239 kopiert.
   Direkt im Anschluss folgt wieder ein einzelner Durchgang des modifizierten Zustandsautomaten zur Isolierung des Wortes.

3. Schritt:
   Dieser Schritt ist ebenfalls wie der erste Schritt aufgebaut, jedoch werden hier keine Samples aus `old_data_block` mehr verwendet. Alle Samples werden aus `filtered_float_data` in den temporären Datenblock `data_for_lpc_calculation` kopiert. Dabei werden sowohl die Multiplikation mit dem Hanning-Fenster als auch die Berechnung der Energie des Blockes durchgeführt.
   Direkt im Anschluss folgt wieder ein einzelner Durchgang des modifizierten Zustandsautomaten zur Isolierung des Wortes.

8.7 Isolierung des Wortes und Merkmalsextraktion

Das prinzipielle Vorgehen der Isolierung des Wortes entspricht dem aus Kapitel 4.3.4. Jedoch gibt es bei der DSP-Implementierung zwei Besonderheiten:
1. Es existiert keine durchgehende Aufnahme des Wortes mit Vor- und Nachlauf. Stattdessen werden laufend neue, kleine Datenblöcke aufgenommen.
2. Es werden nicht nacheinander die Wortgrenzen gesucht, die Länge geprüft und zum Abschluss die LPC-Koeffizienten berechnet. Hier laufen diese drei Schritte ineinander verschachtelt ab.

Für die Beschreibung der angepassten Vorgehensweise werden die von der DSP-Software verwendeten Variablen- und Konstantennamen verwendet.
- `status`: aktueller Status der Wortgrenzenerkennung
- `record_counter`: Laufindex für die Blockindizierung im Feld der LPC-Koeffizienten `lpc_data`
- `logarithmical_energy`: logarithmierte quadratische Energie des aktuellen Blockes
- `start_of_word`: Nummer des ersten Blockes des gefundenen Wortes im Feld der LPC-Koeffizienten
- `end_of_word`: Nummer des letzten Blockes des gefundenen Wortes im Feld der LPC-Koeffizienten
- `gap_in_1_start`: Nummer des ersten Blockes der Energielücke (Wortpause) in Zone 1 im Feld der LPC-Koeffizienten
- `gap_in_2_start`: Nummer des ersten Blockes der Energielücke (Wortpause) in Zone 2 im Feld der LPC-Koeffizienten
- `UPPER_THRESHOLD`: oberer Schwellwert
- `LOWER_THRESHOLD`: unterer Schwellwert
- `CORRECT_THRESHOLD`: Korrekturwert der beiden Schwellwerte für den Einsatz eines anderen Mikrofons
- `MAXIMUM_SLOPE_LENGTH`: maximale Blockanzahl für die Anstiegszeit der Energie von Zone 1 bis Zone 3
- `MAXIMUM_BREAK_LENGTH`: maximale Blockanzahl der Energielücke (Wortpause)
- `MINIMUM_WORD_LENGTH`: minimale Blockanzahl eines Wortes
- `MAXIMUM_WORD_LENGTH`: maximale Blockanzahl eines Wortes

Die Variablen haben dieselbe Funktion wie bei dem Zustandsautomaten im Simulationsmodell. Nur die beiden Konstanten `CORRECT_THRESHOLD` und `MAXIMUM_WORD_LENGTH` sind neu hinzugekommen.

Der Zustandsautomat wird aus Kapitel 4.3.4 übernommen und an die beiden oben aufgeführten Besonderheiten angepasst. Dieser angepasste Zustandsautomat ist in der nachfolgenden Grafik dargestellt. Gezeigt werden nur die Bedingungen, bei denen ein Zustand in einen anderen übergeht. Werden nicht alle Bedingungen für den Wechsel vom aktuellen Zustand in einen anderen Zustand erfüllt, bleibt der aktuelle Zustand bestehen.
Aus Übersichtsgründen werden die genauen Inhalte der einzelnen Zustände nicht mit abgebildet. Es werden die von der DSP-Software verwendeten Variablen-, Konstanten- und Zustandsnamen übernommen.

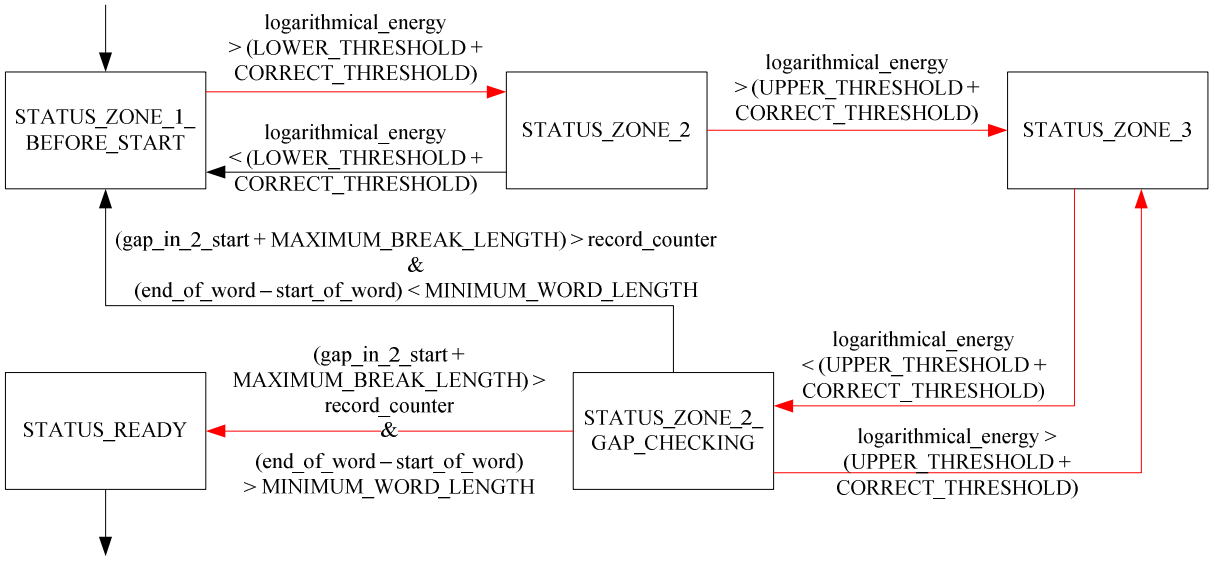

Abbildung 8-5: Zustandsautomat der Wortgrenzenerkennung beim DSP

Bei jedem Schritt der Blockbildung (siehe Kapitel 8.6) wird genau ein Block untersucht. Es wird nur ein Zustandswechsel innerhalb des Zustandsautomaten ausgeführt, wobei auch möglich ist, dass ein Zustand erhalten bleibt. Der Laufindex `record_counter` wird in jedem Schritt der Blockbildung einmal inkrementiert, sofern der Zustand nicht `STATUS_ZONE_1_BEFORE_START` ist oder bleibt.
Der Beginn eines Wortes `start_of_word` ist in der Regel 0. Dieser Wert kann aber auch einen anderen Wert annehmen, sofern die in Kapitel 4.3.4 beschriebene Ausnahme (sehr langsamer Energieanstieg von Zone 1 bis Zone 3 am Anfang eines Wortes) an der markanten Stelle 2 in Abbildung 4-8 eintrifft.
Der Zustand `status` und die für die Wortisolierung nötigen Werte `record_counter`, `start_of_word`, `end_of_word`, `gap_in_1_start` und `gap_in_2_start` werden gespeichert. Der Zustandsautomat wird mit diesen Werten beim nächsten Schritt der Blockbildung fortgesetzt, bis ein komplettes, gültiges Wort isoliert wurde.
Durch diese Vorgehensweise kann unmittelbar nach Sprechen des Wortes eine Identifikation per DTW erfolgen.

Im Folgenden werden nur die Anpassungen des Zustandsautomaten erläutert.

- Bei jedem Zustandsübergang, der im obigen Zustandsautomaten durch einen roten Pfeil dargestellt ist, wird ein Satz LPC-Koeffizienten berechnet und über den Laufindex

record_counter direkt in das Feld der LPC-Koeffizienten lpc_data gespeichert. Im Anschluss wird record_counter inkrementiert und überprüft. Hat er die maximale Wortlänge MAXIMUM_WORD_LENGTH erreicht, geht der Zustandsautomat in den STATUS_READY über; die Variable listening_state wird auf NO gesetzt. Somit wird beim nächsten Auftreten eines EDMA-Interrupts ein Mustervergleich per DTW eingeleitet.

- Wird bei der Untersuchung des Energiewertes des aktuellen Blockes in STATUS_ZONE_2, STATUS_ZONE_3 oder STATUS_ZONE_2_GAP_CHECKING festgestellt, dass der Zustand bestehen bleibt, wird auch hier ein Satz LPC-Koeffizienten berechnet und über den Laufindex record_counter direkt in das Feld der LPC-Koeffizienten lpc_data gespeichert. Im Anschluss wird record_counter inkrementiert und überprüft. Hat er die maximale Wortlänge MAXIMUM_WORD_LENGTH erreicht, geht der Zustandsautomat in den STATUS_READY über; die Variable listening_state wird auf NO gesetzt. Somit wird beim nächsten Auftreten eines EDMA-Interrupts ein Mustervergleich per DTW eingeleitet. Ansonsten wird der Zustandsautomat verlassen und mit dem nächsten Schritt der Blockbildung fortgefahren. Der Zustand status und alle für die Wortisolierung nötigen Werte werden gespeichert.

- Im Zustand STATUS_ZONE_2_GAP_CHECKING wird, nachdem das Ende eines Wortes festgestellt wurde, geprüft, ob das Wort die MINIMUM_WORD_LENGTH überschreitet und es sich somit um ein gültiges Wort handelt. Ist dieses der Fall, geht der Zustandsautomat in den Zustand STATUS_READY über; die Variable listening_state wird auf NO gesetzt. Somit wird beim nächsten Auftreten eines EDMA-Interrupts ein Mustervergleich per DTW eingeleitet.
  Ist das isolierte Wort zu kurz, werden alle Hilfsvariablen auf Null gesetzt und der Zustandsautomat geht in den Zustand STATUS_ZONE_1_BEFORE_START über. Der Wert von listening_state bleibt YES.

8.8 Mustervergleich mit DTW

Der DTW-Algorithmus der DSP-Implementierung ist eine direkte Umsetzung der *dtw.m* des Simulationsmodells (siehe Kapitel 4.6 und 4.8). Die Funktion ist in der Datei *dtw.c* gespeichert. Beide Funktionen sind identisch aufgebaut.
Sind die Koeffizienten eines unbekannten, isolierten Wortes bestimmt, werden dessen LPC-Koeffizienten per DTW mit den Koeffizienten eines jeden vorhandenen Wortes der Referenzdatenbank verglichen. Eine Kontrolle der Längendifferenz wie beim Simulationsmodell erfolgt nicht. Ein Referenzwort wird nur dann von dem Vergleich ausgeschlossen, wenn von ihm keine LPC-Koeffizienten in der Referenzdatenbank existieren.
Jedes einzelne DTW-Ergebnis wird in einer Tabelle gespeichert. Nachdem der Datensatz des unbekannten Wortes mit jedem Datensatz aus der Referenzdatenbank verglichen wurde, kann das Wort identifiziert werden. Hierzu wird das kleinste der DTW-Ergebnisse ausgewählt. Das unbekannte Wort gilt als identifiziert. Es handelt sich somit um das Wort, dessen Referenzdatensatz an dem Vergleich mit dem kleinsten gültigen DTW-Ergebnis beteiligt ist. Direkt im Anschluss wird das empfangene Wort als Steuerkommando per RS232 an die Kommunikationsplatine gesendet. Diese zeigt das Wort über ein Display an, so dass der Sprecher zusätzlich zu der Bewegung des Roboters über das Wort informiert wird.
Weitere Bewertungskriterien sind nicht implementiert.
Alle für die Isolierung des Wortes benötigten Variablen werden auf Null und die Variable listening_state auf YES gesetzt. Der DSP beginnt erneut mit dem „Lauschen“ und wartet auf ein neues, gesprochenes Wort.

8.9 Globale Systemparameter und globale Variablen

Die DSP-Software der Spracherkennung verfügt über mehrere wichtige globale Parameter und globale Variablen. Diese müssen mit den Einstellungen des Simulationsmodells übereinstimmen, da die Referenzdatenbank aus dem Simulationsmodell übernommen wird. Im Gegensatz zu dem Simulationsmodell sind alle Parameter während der Laufzeit nicht veränderbar.
Bis auf einen globalen Systemparameter werden alle in der *const.h* definiert:

- `NO = 0`: Konstante, die für eine bessere Übersicht bei Abfragen verwendet wird
- `YES = 1`: Konstante, die für eine bessere Übersicht bei Abfragen verwendet wird
- `BLOCKSIZE = 240`: Anzahl der Samples pro Block, entspricht 30 ms
- `P = 10`: Anzahl der verwendeten LPC-Koeffizienten
- `SHIFT = 80`: Anzahl der Samples, um die der folgende Block verschoben wird, entspricht 10 ms
- `MAXIMUM_WORD_LENGTH = 80`: maximale Länge eines Wortes in Blöcken, entspricht 820 ms
- `NUMBER_OF_SPEECH_COMMANDS = 8`: Anzahl implementierter Wörter
- `UPPER_THRESHOLD = -2.00`: oberer Schwellwert der Wortgrenzenerkennung in dB/20
- `LOWER_THRESHOLD = -2.50`: unterer Schwellwert der Wortgrenzenerkennung in dB/20
- `CORRECT_THRESHOLD = 1.7`: Korrekturwert für die beiden Schwellwerte der Wortgrenzenerkennung in dB/20
- `MINIMUM_WORD_LENGTH = 19`: minimale Wortlänge in Blöcken, entspricht 210 ms
- `MAXIMUM_BREAK_LENGTH = 11`: maximale Länge der Energielücke (Pause) in Blöcken, entspricht 130 ms
- `MAXIMUM_SLOPE_LENGTH = 9`: maximale Länge der Anstiegszeit am Wortanfang bzw. der Abfallzeit am Wortende in Blöcken, entspricht 110 ms

Des Weiteren sind Konstanten für den Zustandsautomaten der Wortgrenzenerkennung enthalten:

- `STATUS_ZONE_1_BEFORE_START = 0`
- `STATUS_ZONE_2 = 2`
- `STATUS_ZONE_3 = 3`
- `STATUS_ZONE_2_GAP_CHECKING = 4`
- `STATUS_READY = 6`

Die einzige Konstante, die in *word_recognition.c* definiert wird, ist `PRINTF`. Ist diese Definition vorhanden, wird nach der DTW-Berechnung eine Tabelle mit allen Ergebnissen ausgegeben. Ist diese Definition nicht vorhanden, wird das Ergebnis nur über RS232 an die Kommunikationsplatine übertragen. Es erfolgt keine Ausgabe der Ergebnistabelle.

Die globalen Variablen werden vor dem Beginn von `main` in *word_recognition.c* definiert.
Folgende Variablen werden von der Originalversion des Demo-Programmes übernommen:

- `adcbuffer`: Ping-Pong-Puffer für die Audioeingänge
- `dacbuffer`: Ping-Pong-Puffer für die Audioausgänge
- `block`: Nummer der aktuellen Ping-Pong-Puffer, die für den normalen Programmablauf zur Verfügung stehen

Die folgenden drei globalen Variablen werden im SBSRAM angelegt:

- `reference_data`: eindimensionales Feld für die Referenzdatenbank
- `lpc_data`: zweidimensionales Feld für die berechneten LPC-Koeffizienten eines Wortes
- `D`: eindimensionales Feld für den DTW-Algorithmus

9 Testergebnisse der DSP-Implementierung

Während der Umsetzung des Simulationsmodells der Spracherkennung auf dem DSP wurde jeder einzelne programmierte Funktionsteil der DSP-Software ausführlich getestet, um diesen als Fehlerquelle so weit wie möglich ausschließen zu können. Dabei wurden z. B. feste Zahlenfolgen als Eingangsdaten verwendet, um einen direkten Vergleich mit den einzelnen Funktionen des Simulationsmodells zu erhalten und die Ergebnisse somit zu verifizieren.
Ein zusammenhängender Funktionstest wurde jedoch erst möglich, nachdem sowohl eine Referenzdatenbank importiert als auch eine Spracherkennung durchgeführt werden konnten. Als Referenzdatenbank wurde die Datei *reference_database_7.mat* vom 19.11.2009, 11:55h in die Datei *reference_database.txt* exportiert. Somit verwendet der DSP dieselbe Referenzdatenbank wie das Simulationsmodell.
Die Erkennungsrate ist stark vom verwendeten Mikrofon abhängig. Das verwendete Mikrofon (siehe Abbildung 3-2) ist deutlich empfindlicher als das beim Simulationsmodell verwendete PC-Mikrofon. Aus diesem Grunde müssen die beiden Schwellwerte der Wortgrenzenerkennung durch den Korrekturwert `CORRECT_THRESHOLD` angehoben werden. Ansonsten würde das Umgebungsrauschen (z. B. Klimaanlage) bereits deutlich oberhalb des oberen Schwellwertes liegen. Eine Wortisolierung wäre somit nicht möglich.

Während des Testablaufes sprach ein Sprecher der Reihe nach einzeln die Worte „links“, „rechts“, „vorwärts“, „rückwärts“, „schneller“, „langsamer“, „halt“ und „drehe“. Die Rate der richtig zugeordneten Erkennungen bei Verwendung des Mikrofons von Abbildung 3-2 lag bei einem Wert von ca. 60%. Jedoch erfolgte bei den restlichen 40% eine falsche Zuordnung. Eine Abweisung fand nicht statt. Von der falschen Zuordnung waren besonders die Worte „langsamer“, „vorwärts“, „rückwärts“ und „halt“ betroffen.

Bei einem zweiten Test wurde der PC über den Ausgang der Soundkarte mit dem Audioeingang des DSP-Moduls verbunden. Hierzu wurde der rechte Kanal der Soundkarte an die Pins G1 (ADC 0 IN, Signalleitung) und H1 (AGND, Abschirmung) des *D.Module.PCM3003* angeschlossen [6].
Der Korrekturwert `CORRECT_THRESHOLD` musste hier auf 0 gesetzt werden, da diese Aufnahmen mit dem PC-Mikrofon aufgenommen worden sind. Voraussetzung war allerdings die richtige Einstellung des Ausgangspegels der Soundkarte. Dieser musste experimentell eingestellt werden. Auf diesem Wege konnten die in dem Unterverzeichnis *reference_files* des Simulationsmodells gespeicherten *wav*-Dateien der Referenzdatenbank abgespielt und das Ergebnis der Spracherkennung mit dem DSP untersucht werden. Alle in der Referenzdatenbank verwendeten Sprachproben wurden erkannt. Auch andere Aufnahmen gesprochener Wörter führten zu einer Erkennungsrate von ca. 75%. Die restlichen 25% wurden falsch zugeordnet, da keine Entscheidungskriterien für das Abweisen falscher Worte implementiert sind.

Der DSP liefert das Ergebnis der Spracherkennung sehr schnell. Der Sprecher hat das Wort gerade fertig gesprochen, da wird das Ergebnis bereits ausgegeben. Das Vorgehen, alle Schritte von der Energieberechnung bis hin zur Merkmalsextraktion nicht erst nach dem Aufnehmen des ganzen Wortes, sondern stetig während der Aufnahme der Datenblöcke durchzuführen, hat sich somit bewährt. Dabei erfolgt die Isolierung des Wortes durch die Berechnung der Energiegrenzen – ebenso wie die Berechnung der Blockenergie – ständig. Die Wortisolierung entscheidet laufend, ob ein Block untersucht und somit die Aufnahme von LPC-Koeffizienten gestartet bzw. beendet oder ob der Block verworfen wird.
Mit diesem zufriedenstellenden Ergebnis ist die Umsetzung des Simulationsmodells zur Spracherkennung auf dem DSP abgeschlossen.

10 Ausblick und Fazit

Es hat sich gezeigt, dass das Simulationsmodell durch das Optimieren der Referenzdatenbank und der erweiterten Bewertungskriterien vor und nach der DTW-Berechnung zu einer sehr guten Erkennungsrate von 95% und einer Abweisungsrate unbekannter Wörter von 75% geführt hat. Zudem wurde mit der Referenzdatenbank, die am 19.11.2009 aufgenommen wurde und die nur einen Sprecher enthält, die Grenze der sprecherabhängigen zur sprecherunabhängigen Einzelworterkennung überschritten. Das konnte durch Tests mit der Referenzdatenbank und sechs verschiedenen, nicht angelernten Testsprecherinnen und Testsprechern, die ebenfalls zu sehr guten Erkennungsraten führten, bewiesen werden. Diese Referenzdatenbank ist mittlerweile über zwei Monate alt und führt nach wie vor zu denselben Resultaten.

Die Erkennungsrate bleibt auch bei einem Vergrößern des Abstandes zwischen Mikrofon und Sprecher auf über einen Meter stabil. Bei Mikrofonen, die bei stärkeren Schallpegeln zum Übersteuern oder Begrenzen („Scheppern“) neigen, ist unbedingt ein gewisser Minimalabstand einzuhalten.

Die Qualität der Spracherkennung ist stark von der Qualität des Eingangssignals abhängig. Für eine weitere Verbesserung der Spracherkennung müsste das Eingangssignal von Störungen und Nebengeräuschen befreit werden. Hierzu sind zusätzliche Maßnahmen vor dem verwendeten FIR-Bandpassfilter erforderlich. Eine Möglichkeit wäre ein lineares oder zirkulares Mikrofonarray mit Beamforming. Die Spracherkennung unterscheidet momentan nicht immer zwischen „links“ und „linker“, „rechts“ und „rechter“ oder „schnell“ und „schneller“. Diese feinen Wortunterschiede könnten durch eine solche Störunterdrückung unterscheidbarer werden.

Die realisierte Spracherkennung verwendet als Merkmale LPC-Koeffizienten. Der Einsatz anderer oder zusätzlicher Merkmale wie z.B. der Cepstrum-Koeffizienten oder dynamischer Merkmale wurde erprobt. Jedoch wurden diese Versuche verworfen, da sie im Rahmen dieser Masterarbeit zu keinen weiteren Verbesserungen der Erkennungs- und Abweisungsrate führten.

Da die entwickelte DSP-Software nicht erst eine mehrere Sekunden lange Aufnahme vornimmt und diese anschließend analysiert, sondern laufend sehr kurze, zusammenhängende Zeitabschnitte mit einer Länge von jeweils 30 ms aufnimmt und diese unmittelbar analysiert, stehen, nachdem der Sprecher sein Wort beendet hat, bereits alle LPC-Koeffizienten dieses neuen, noch zu identifizierenden Wortes bereit. Der Mustervergleich per DTW kann sofort gestartet werden. Da dieser mit der entwickelten DSP-Software auf dem schnellen DSP eine sehr kurze Bearbeitungszeit in Anspruch nimmt, steht die Identifikation des gesprochenen Wortes unmittelbar nach dessen Aussprache bereit; der Roboter kann somit quasi in Echtzeit per Sprachkommando gesteuert werden.

Auf dem DSP sind die Bewertungskriterien der Längenkontrolle und des kleinsten DTW-Ergebnisses implementiert, jedoch nicht der Einsatz der Referenz-DTW-Ergebnisse. Daher weist die Spracherkennung mit dem DSP nur wenige, unbekannte Worte ab. Um die Abweisungsrate hier zu erhöhen, könnten die Referenz-DTW-Ergebnisse ebenfalls importiert und während des Mustervergleiches kontrolliert werden.

Da die Spracherkennung ein sehr kompliziertes und weitläufiges Themengebiet ist, war die Einarbeitung und Literaturrecherche entsprechend langwierig.
Eine besondere Herausforderung war, dass erst nachdem alle für die Vorverarbeitung, die Merkmalsextraktion und den Mustervergleich nötigen Programmteile programmiert und einzeln getestet worden waren, eine erste Simulation des zusammenhängenden Systems möglich war. Diese Simulation verlief jedoch äußerst zufriedenstellend, so dass nicht erst kompliziert nach Fehlerquellen gesucht werden musste. Statt dessen konnte direkt mit dem Variieren einzelner Systemparameter unter Beobachtung der Auswirkungen auf das Gesamtergebnis begonnen werden.

11 Quellenverzeichnis

[1] Wikipedia: Spracherkennung. Erschienen als Online-Dokumentation unter http://de.wikipedia.org/wiki/Spracherkennung, Stand vom 18.01.2010

[2] Ziad Ghadieh: Automatische Spracherkennung zur Robotersteuerung auf dem TI DSP D.MODULE.C6713. Erschienen als Masterarbeit an der HAW Hamburg / FH Westküste am 24.04.2008

[3] Wikipedia-User Jahobr: Robot omnidirectional movement speedvectors.PNG. Erschienen als Grafik unter http://de.wikipedia.org/wiki/Omnidirektionaler_Antrieb, Stand vom 13.07.2007

[4] Dr. Christian Verbeek: Serial line communication to IO-Board. Erschienen als Online-Dokumentation unter http://forum.openrobotino.org/showthread.php?t=17&highlight=serial, Stand vom 29.05.2008

[5] Kontron Embedded Modules AG: MOPSlcdSE and MOPS/SE User's Guide. Erschienen als Datenblatt auf der beim *Robotino*® mitgelieferten CD-ROM *Robotino® View Release 1.7c* unter \DOC\DE\Kontron_M_MOPSlcdSE_MOPSSE_PSTEM111.pdf, Stand vom 11.10.2004

[6] D.SignT GmbH & Co. KG: D.Module.PCM3003 User's Guide. Erschienen als Datenblatt unter http://www.dsignt.de/save/dmodule/ugdpcm3003.pdf, Stand von 02.2005

[7] D.SignT GmbH & Co. KG: D.Module.C6713 User's Guide. Erschienen als Datenblatt unter http://www.dsignt.de/save/dmodule/ugd6713.pdf, Stand von 09.2004

[8] Stephen Euler: Grundkurs Spracherkennung. Erschienen als Buch beim Friedr. Vieweg & Sohn Verlag, 1. Auflage vom April 2006, ISBN 3-8348-0003-1

[9] Lawrence Rabiner, Biing-Hwang Juang, B. Yegnanarayana: Fundamentals of speech recognition. Erschienen als Buch bei Dorling Kindersley (India) Pvt. Ltd, 1. Auflage von 2009, ISBN 978-81-7758-560-5

[10] Martin Werner: Digitale Signalverarbeitung mit Matlab – Praktikum. Erschienen als Buch beim Vieweg+Teubner Verlag, 1. Auflage 2008, ISBN 978-3-8348-0393-1

[11] Atmel Corporation: ATmega162. Erschienen als Datenblatt unter http://www.atmel.com/dyn/resources/prod_documents/doc2513.pdf, Stand vom 28.08.2007

[12] RECOM Electronic GmbH: INNOLINE DC/DC-Converter 2, 3, 4, 5 AMP SIP12 Vertical & Horizontal. Erschienen als Datenblatt unter http://www.recom-international.com/pdf/Innoline/R-5xxxPA_DA.pdf, Stand vom 19.05.2009

[13] Maxim Integrated Products: ±15kV ESD-Protected, +5V RS-232 Transceivers, MAX202E–MAX213E, MAX232E/MAX241E. Erschienen als Datenblatt unter http://datasheets.maxim-ic.com/en/ds/MAX202E-MAX241E.pdf, Stand von 03/2005

[14] ELECTRONIC ASSEMBLY GmbH: BLUE LINE DOTMATRIX LCD-SERIE 1x16..4x40. Erschienen als Datenblatt unter http://www.lcd-module.de/pdf/doma/blueline-w.pdf, Stand von 08.2008

[15] fischertechnik®: ROBO Interface Bedienungsanleitung. Erschienen als Dokumentation, Beilage des *ROBO Mobile Set* von *fischertechnik*®

[16] ELECTRONIC ASSEMBLY GmbH: LCD-MODUL 2x16 - 6,68mm INKL. KONTROLLER HD44780. Erschienen als Datenblatt unter http://www.lcd-module.de/pdf/doma/dip162-d.pdf, Stand von 03.2008

12 Anhang

12.1 Verzeichnis fachspezifischer Abkürzungen

ASCII	American Standard Code for Information Interchange
CLK	Clock
DSP	Digitaler Signalprozessor
DTW	Dynamic Time Warping
EDMA	Enhanced Direct Memory Access
FIR-Filter	Finite Impulse Response Filter
HMM	Hidden-Markov-Modell
I^2S	Inter-IC Sound Interface
ISP	In System Programmer
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
LPC	Linear Predictive Coding
McBSP	Multichannel Buffered Serial Port
MISO	Master In Slave Out
MOSI	Master Out Slave In
MSB	Most Significant Bit
PDIP	Plastic Dual Inline Package
PWM	Pulse Width Modulation
RISC	Reduced Instruction Set Computing
R/W	Read / Write
RxD	Receive Data
SBSRAM	Synchronous Burst Static Random Access Memory
SCK	Serial Clock
SDRAM	Synchronous Dynamic Random Access Memory
TQFP	Thin Quad Flat Package
TTL	Transistor-Transistor-Logik
TxD	Transmit Data
USART	Universal Synchronous/Asynchronous Receiver Transmitter
Wide SO	Wide Small Outline

12.2 Schaltpläne und Layout der Kommunikationsplatine

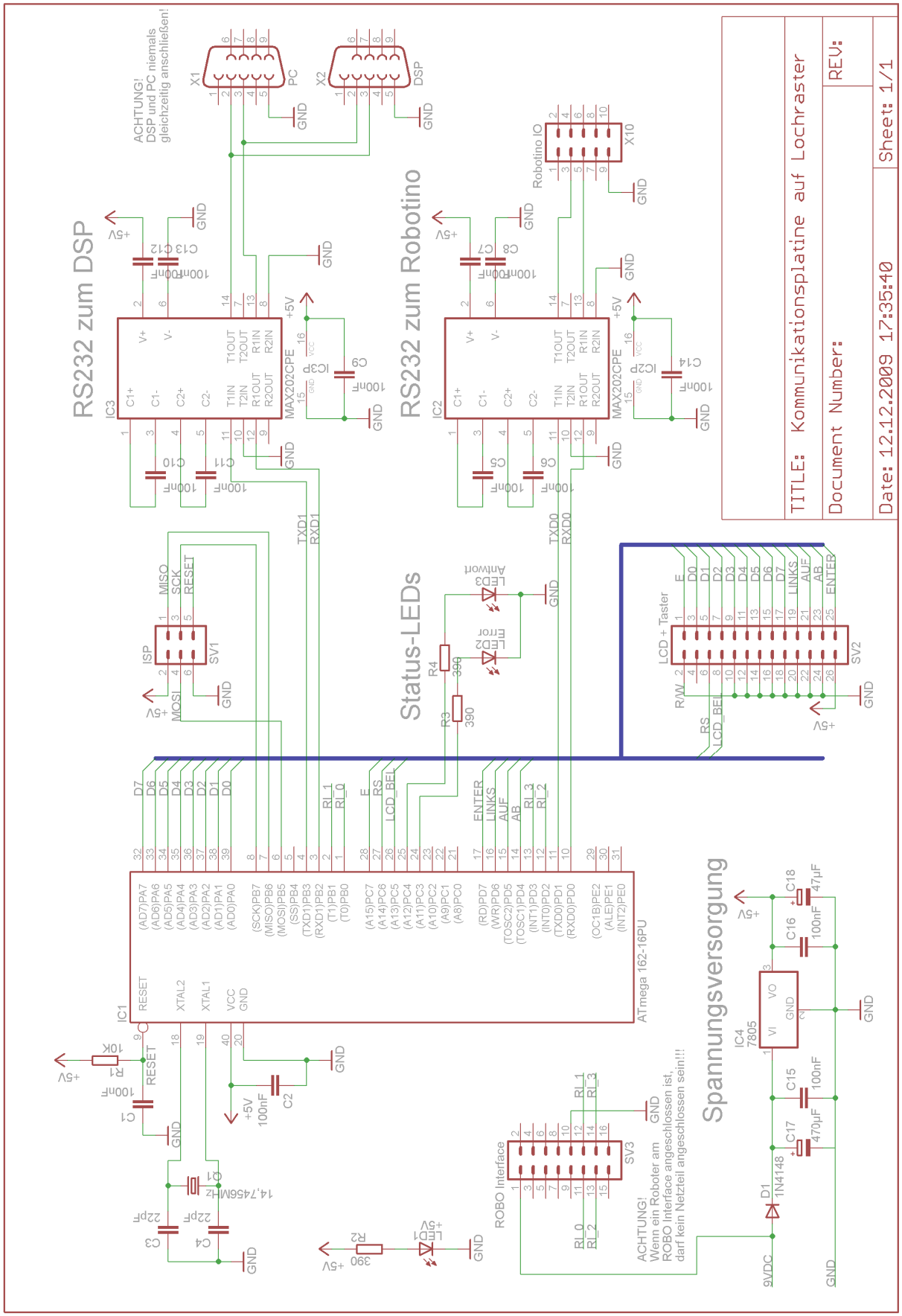

Abbildung 12-1: Schaltplan für den Aufbau auf einer Lochrasterplatine

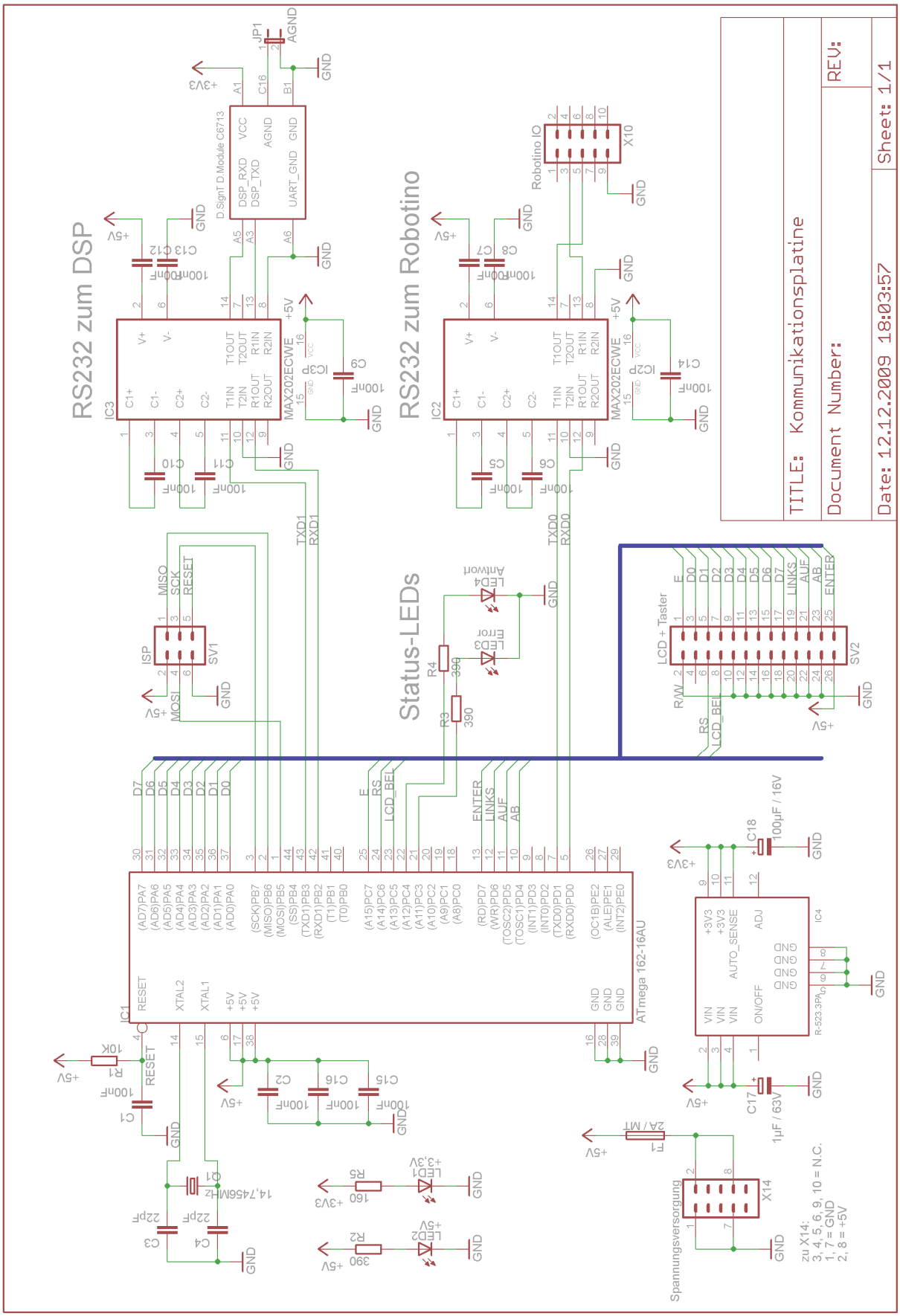

Abbildung 12-2: Schaltplan der gefertigten Kommunikationsplatine

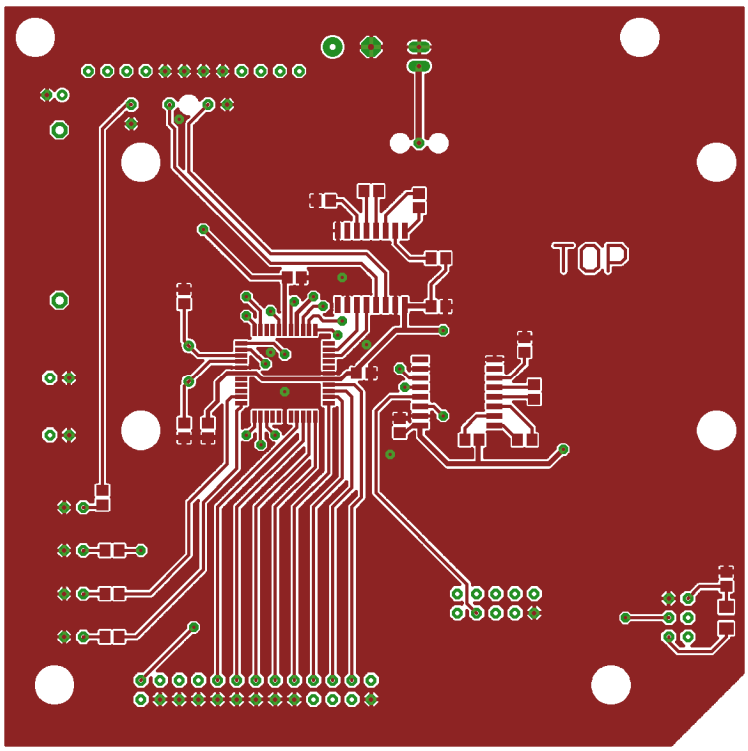

Abbildung 12-3: Obere Seite der Kommunikationsplatine in Originalgröße (TOP)

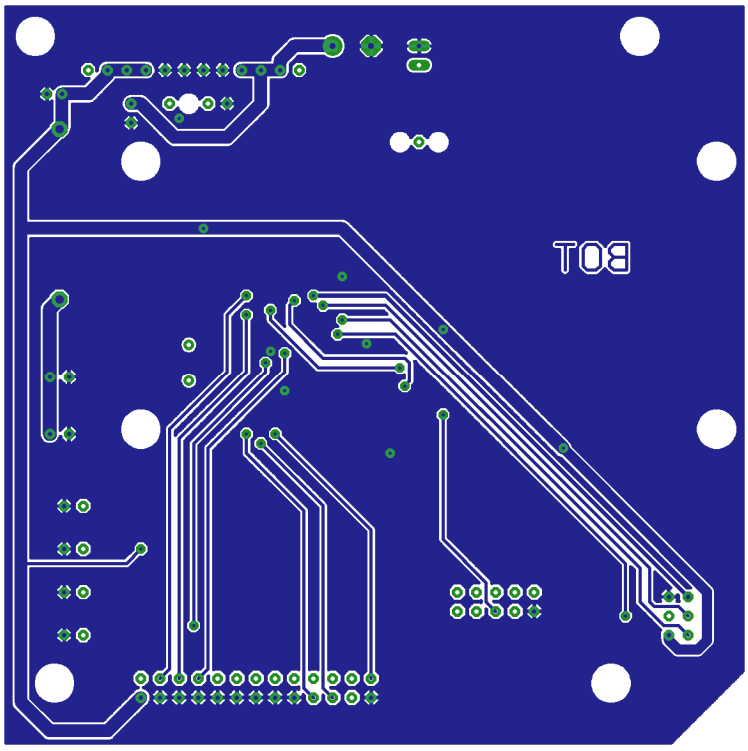

Abbildung 12-4: Untere Seite der Kommunikationsplatine in Originalgröße (BOTTOM)

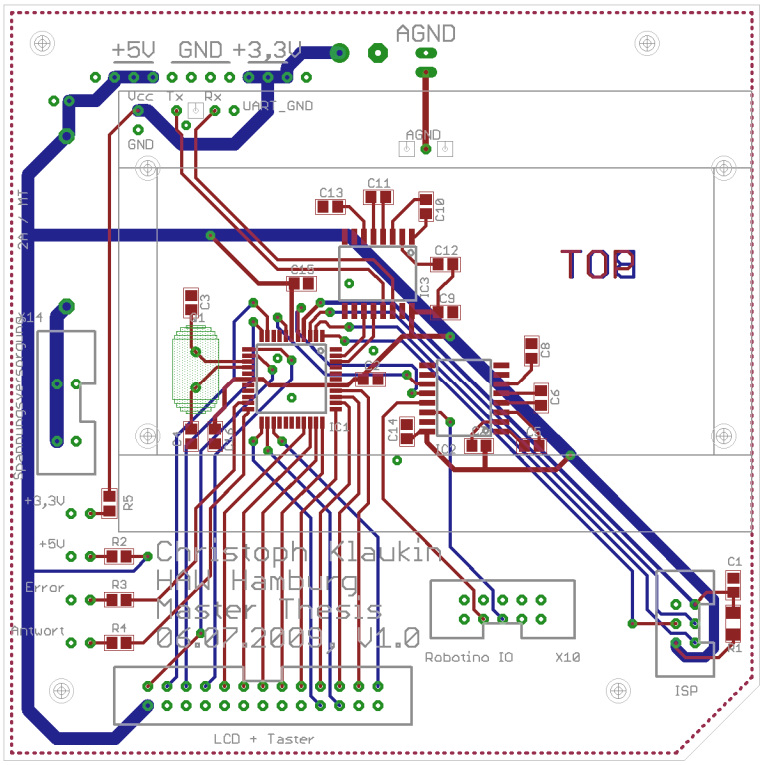

Abbildung 12-5: Layer der Kommunikationsplatine übereinander (Originalgröße)

12.3 Stücklisten

In diesem Kapitel werden sämtliche Stücklisten für die selbst gebauten und bestückten Platinen aufgeführt.

Für beide Platinenversionen werden zusätzlich noch folgende Materialien benötigt:
- 1x LCD *EA DIP162-DNLED*, Hersteller *ELECTRONIC ASSEMBLY GmbH*
- 1x 20cm Flachbandkabel, 26-polig, Rastermaß 2,54mm
- 1x Pfostenstecker, 26-polig, für Flachbandkabel, zweireihig, Rastermaß 2,54mm

Für die Kommunikation mit dem *Robotino*® werden zusätzlich noch folgende Materialien benötigt:
- 1x Pfostenstecker, 10-polig, für Flachbandkabel, zweireihig, Rastermaß 2,54mm
- 1x 25cm Flachbandkabel, 10-polig, Rastermaß 2,54mm
- 1x D-SUB-Stecker, 9-polig, für Flachbandkabel, Rastermaß 2,54mm

12.3.1 Stückliste für den Aufbau auf einer Lochrasterplatine

Bezeichnung im Schaltplan	Bauteil
R1	Kohleschichtwiderstand, 10KΩ, 5%, 0,25W
R2, R3, R4	Kohleschichtwiderstand, 390Ω, 5%, 0,25W
C1, C2, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15, C16	Folienkondensator, 100nF, 20%, Rastermaß 2,5mm
C3, C4	Keramikkondensator, 22pF, Rastermaß 2,5mm
C17	Elektrolytkondensator, 470µF, 16V, 20%, Rastermaß 5mm
C18	Elektrolytkondensator, 47µF, 16V, 20%, Rastermaß 2,5mm
Q1	Standardquarz, 14,7456MHz ±30ppm, Gehäuse HC49U-S
D1	*1N4148*
LED1	LED, low current, 5mm, diffus, grün
LED2	LED, low current, 5mm, diffus, rot
LED3	LED, low current, 5mm, diffus, gelb
IC1	*ATmega162-16PU*, Gehäuse DIP-40
IC2, IC3	*MAX202CPE*, Gehäuse PDIP-16
IC4	*7805*, Gehäuse TO220
SV1	Wannenstecker, 6-polig, gerade, zweireihig, schwarz, Rastermaß 2,54mm
SV2	Wannenstecker, 26-polig, gerade, zweireihig, grau, Rastermaß 2,54mm
SV3	Wannenstecker, 16-polig, gerade, zweireihig, schwarz, Rastermaß 2,54mm
X1	D-SUB-Buchse, 9-polig, gewinkelt, Printmontage
X2	D-SUB-Stecker, 9-polig, gewinkelt, Printmontage
X10	Wannenstecker, 10-polig, gerade, zweireihig, schwarz, Rastermaß 2,54mm

Tabelle 12-1: Stückliste für den Aufbau auf einer Lochrasterplatine

Für die Kommunikation mit dem *ROBO Interface* werden folgende Materialien benötigt:
- 1x Pfostenstecker, 16-polig, für Flachbandkabel, zweireihig, Rastermaß 2,54mm
- 1x 25cm Flachbandkabel, 16-polig, Rastermaß 2,54mm
- 1x Pfostenstecker, 26-polig, für Flachbandkabel, zweireihig, Rastermaß 2,54mm

12.3.2 Stückliste der gefertigten Kommunikationsplatine

Bezeichnung im Schaltplan	Bauteil
R1	SMD-Widerstand, 10KΩ, 5%, 0,25W, Bauform 0805
R2, R3, R4	SMD-Widerstand, 390Ω, 5%, 0,25W, Bauform 0805
R5	SMD-Widerstand, 160Ω, 5%, 0,25W, Bauform 0805
C1, C2, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15, C16	SMD-Vielschichtkondensator, 100nF, 10%, Bauform 0805
C3, C4	SMD-Keramikkondensator, 22pF, Bauform 0805
C17	Elektrolytkondensator, 1µF, 63V, 20%, Rastermaß 2mm
C18	Tantal-Kondensator, 100µF, 16V, Rastermaß 5mm
F1	*PL OGN-22,5*, Sicherungshalter, Printanschluss, 5x20mm, mit Sicherung 2A, MT
Q1	Standardquarz, 14,7456MHz ±30ppm, Gehäuse HC49U-S
LED1, LED2	LED, low current, 3mm, diffus, grün
LED3	LED, low current, 3mm, diffus, rot
LED4	LED, low current, 3mm, diffus, gelb
IC1	*ATmega162-16AU*, Gehäuse TQFP-44
IC2, IC3	*MAX202ECWE*, Gehäuse Wide SO-16
IC4	*R-523.3PA*, Gehäuse SIP12
SV1	Wannenstecker, 6-polig, gerade, zweireihig, grau, Rastermaß 2,54mm
SV2	Wannenstecker, 26-polig, gerade, zweireihig, grau, Rastermaß 2,54mm
X10	Wannenstecker, 10-polig, gerade, zweireihig, blau, Rastermaß 2,54mm
X14	Wannenstecker, 10-polig, gerade, zweireihig, schwarz, Rastermaß 2,54mm, ohne die Pins 3, 4, 5, 6, 9, 10

Tabelle 12-2: Stückliste der gefertigten Kommunikationsplatine

Um das *D.Module.C6713* und das *D.Module.PCM3003* auf die Kommunikationsplatine aufzustecken, werden folgende Materialien benötigt:
- 1x Kontaktbuchsenleiste, 9-polig, einreihig, trennbar, Rastermaß 2,54mm
- 4x Gewindeschraube, Zylinderkopf, Schlitz, M3, 30mm, mit passender Mutter
- 4x Distanzhülse, Kunststoff, schwarz, 12mm

Für die Versorgung der Kommunikationsplatine mit Spannung vom *Robotino*® werden folgende Materialien benötigt:
- 1x Pfostenstecker, 10-polig, für Flachbandkabel, zweireihig, Rastermaß 2,54mm
- 1x 25cm Flachbandkabel, 10-polig, Rastermaß 2,54mm
- 1x Laborstecker, 2mm, Lötanschluss, rot
- 1x Laborstecker, 2mm, Lötanschluss, schwarz

12.4 Inhalt der CD-ROM

Der Quellcode umfasst insgesamt 5713 Programmzeilen. Dabei entfallen 2730 Programmzeilen auf das Simulationsmodell mit *Matlab*®, 1660 auf die DSP-Software und die restlichen 1323 auf die Software des Prozessors der Kommunikationsplatine. Würden diese Programme hier abgedruckt, wäre die Masterarbeit um über 100 Seiten länger. Aus diesem Grunde befindet sich der komplette Quellcode auf der dieser Masterarbeit beigelegten CD-ROM.

Inhalt der CD-ROM:

- AVR_SW
- Spracherkennung_DSP
- Spracherkennung_Simulationsmodell
- Masterarbeit.pdf
- SW_ROBO_Interface.rpp

Die CD-ROM befindet sich auf der letzten Seite dieser Masterarbeit (Seite 96).

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, den 27.01.2010

(Christoph Klaukin)