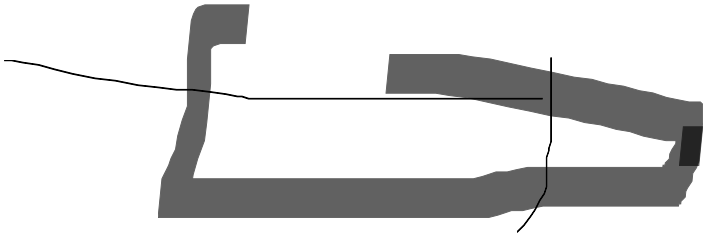
Hochschule Offenburg
offenburg.university

Bachelor-Thesis

Entwicklung einer IoT-Anwendung zur Spracherkennung

Erstellt von	Marian Siebert
Hochschule	Hochschule Offenburg
Fakultät	Elektrotechnik und Informationstechnik
Studiengang	Elektrotechnik/Informationstechnik
Bearbeitungszeitraum	15.10.2018 bis 12.02.2019
Betreuender Professor	Prof. Dr.-Ing. Daniel Fischer
Zweiter Betreuer	Frank Erdrich, M.Sc.



Eidesstattliche Erklärung

Hiermit versichere ich eidesstattlich, dass die vorliegende Thesis von mir selbstständig und ohne unerlaubte fremde Hilfe angefertigt worden ist, insbesondere, dass ich alle Stellen, die wörtlich oder annähernd wörtlich oder dem Gedanken nach aus Veröffentlichungen, unveröffentlichten Unterlagen und Gesprächen entnommen worden sind, als solche an den entsprechenden Stellen innerhalb der Arbeit durch Zitate kenntlich gemacht habe, wobei in den Zitaten jeweils der Umfang der entnommenen Originalzitate kenntlich gemacht wurde. Die Arbeit lag in gleicher oder ähnlicher Fassung noch keiner Prüfungsbehörde vor und wurde bisher nicht veröffentlicht. Ich bin mir bewusst, dass eine falsche Versicherung rechtliche Folgen haben wird.

Datum Unterschrift

Vorwort

Die automatische Spracherkennung ist ein umfangreiches und sehr interessantes Gebiet, vor allem wenn eine entsprechende Anwendung auf Embedded Systemen mit begrenzten Ressourcen umgesetzt werden soll. So gibt es einerseits bereits viele fertige Spracherkennungssysteme, die aber andererseits nur selten zuverlässig und in Echtzeit auf einem Mikrocontroller laufen.

Ich möchte mich an dieser Stelle bei allen Personen bedanken, die mich während dieser Bachelor-Thesis unterstützt haben. Ein besonderer Dank geht dabei an Prof. Dr.-Ing. Daniel Fischer für die Betreuung während der Arbeit und die Möglichkeit, dieses Thema überhaupt bearbeiten zu können. Des Weiteren bedanke ich mich bei Frank Erdrich für einige hilfreiche Hinweise zur Spracherkennung und bei Fabian Veit für die praktische Unterstützung bei einigen aufgetretenen Problemen.

Zusammenfassung

In dieser Abschlussarbeit wird ein Keyword-Spotting-System zur Erkennung von einzelnen Wörtern oder Kommandos auf einem Single-Board-Computer implementiert. Ein Teil der Signalverarbeitung erfolgt dabei auf einem Cortex-M4-Prozessor mit angeschlossenem Mikrofon, von welchem aus Sprachinformationen in der Form von Mel Frequency Cepstral Coefficients (MFCC) an einen Cortex-A9-Pozessor geschickt werden. Die eigentliche Spracherkennung wird dann in einer von zwei Varianten ausgeführt: Entweder über einen Mustervergleich mit Dynamic Time Warp (DTW) für eine sprecherabhängige, oder mithilfe eines künstlichen neuronalen Netzes (KNN) für eine sprecherunabhängige Erkennung. Entsprechend den erfolgreich detektierten Worten werden dann Kommandos an einen MQTT-Broker geschickt, wodurch das Gerät auch an das Internet of Things (IoT) angebunden werden kann.

Die Arbeit gibt zunächst einen Einblick in die theoretischen Grundlagen der automatischen Spracherkennung sowie die dabei verwendeten Verfahren zur Merkmalsextraktion, Mustererkennung und zu neuronalen Netzen, bevor dann die Entwicklung des Keyword-Spotting-Systems auf einem UDOO Neo Full Board beschrieben wird.

Abstract

The subject of this work is the development of a keyword spotting system for the recognition of short words or commands on a single board computer. One part of the signal processing takes place on a Cortex-M4 processor with a microphone attached. It sends the speech information in the form of mel-frequency cepstral coefficients (MFCC) to a Cortex-A9 processor. There are two options in order to carry out the actual keyword detection: Either with a dynamic time warp (DTW) algorithm, where recognition is speaker-dependent, or with an artificial neural network (ANN) approach to achieve speaker-independence. The successfully detected words are then transmitted to an MQTT broker, so that the device can also be connected to the Internet of Things (IoT).

This document describes the theoretical background of automatic speech recognition (ASR) and the methods that are used to implement the keyword spotting system on a UDOO Neo Full board.

Inhaltsverzeichnis

Abbildungsverzeichnis VII

Tabellenverzeichnis IX

Abkürzungsverzeichnis IX

1. Einleitung 1

1.1. Aufgabenstellung 2

1.2. Verwendete Hardware 2

1.3. Verwendete Software 4

2. Theoretische Grundlagen 5

2.1. Sprache und Hörverstehen 5

2.2. Signalvorverarbeitung 7

2.2.1. Signalverstärkung 8

2.2.2. Tiefpassfilterung 8

2.2.3. Analog-Digital-Wandlung 10

2.2.4. Hochpassfilterung 10

2.3. MFCC-Merkmalsextraktion 12

2.3.1. Framing 14

2.3.2. Fensterfunktion 15

2.3.3. Leistungsspektrum 15

2.3.4. Mel-Filterbank 16

2.3.5. Logarithmus 18

2.3.6. Diskrete Kosinustransformation (DCT) 19

2.3.7. Zusätzliche Merkmale 21

2.4. Dynamic Time Warp (DTW) 22

2.4.1. Pfadsuche 24

2.4.2. Funktionale Anpassungen zur Spracherkennung 25

2.5. Künstliche Neuronale Netze (KNN) ... 27
2.5.1. Aufbau ... 27
2.5.2. Training ... 30
2.5.3. Netztopologien ... 32
2.6. Message Queuing Telemetry Transport (MQTT) ... 36
3. Implementierung eines Keyword-Spotting-Systems ... 38
3.1. CMSIS-DSP-Bibliothek ... 39
3.1.1. Optimierung der Kopierfunktion ... 40
3.1.2. Optimierung der Diskreten Fouriertransformation ... 42
3.2. Signalvorverarbeitung ... 44
3.3. MFCC-Merkmalsextraktion ... 47
3.4. Spracherkennungsmodul ... 48
3.4.1. Mustervergleich mit DTW ... 49
3.4.2. Neuronales Netz ... 51
4. Fazit ... 54
5. Ausblick ... 55
Literaturverzeichnis ... 56
Anhang ... A - I
A.1 Optimierte Kopierfunktion ... A - I
A.2 Radix-2 FFT-Algorithmus ... A -II
A.3 Bedienung der Anwendung ... A-III
A.4 Probleme während der Entwicklung ... A-IV
A.5 Inhalt der CD-ROM ... A -V

Abbildungsverzeichnis

Abb. 1.1 - UDOO Neo Board mit aufgestecktem Mikrofonmodul 3

Abb. 2.1 - Relevanz verschiedener Oktavbänder für die Sprachverständlichkeit [3] 7

Abb. 2.2 - Abtastung eines Sinussignals 8

Abb. 2.3 - Darstellung des Aliasing-Effekts im Spektrum 9

Abb. 2.4 - Passiver Tiefpass als einfaches RC-Glied 9

Abb. 2.5 - Pol-Nullstellen-Diagramm eines digitalen Filters zur Entfernung des Gleichsignalanteils 11

Abb. 2.6 - Übertragungsfunktion eines digitalen Filters zur Entfernung des Gleichsignalanteils 11

Abb. 2.7 - Implementierungsstruktur eines digitalen Filters zur Entfernung des Gleichsignalanteils [4] 11

Abb. 2.8 - Pol-Nullstellen-Diagramm (links) und Übertragungsfunktion (rechts) eines Präemphase-Filters 12

Abb. 2.9 - Audiosignal eines einzelnen Lautes (links) und das dazugehörige Spektrum (rechts) 13

Abb. 2.10 - Grober Ablauf der MFCC-Merkmalsberechnung 14

Abb. 2.11 - Darstellung einer Mel-Filterbank mit zehn Dreiecksfiltern 17

Abb. 2.12 - Schematische Darstellung der Datenkompression mit einer DCT 20

Abb. 2.13 - Beispiel für die Anwendung des DTW-Algorithmus auf zwei Zahlenfolgen (rechts das Ergebnis) 23

Abb. 2.14 - Beispiel für die Einzeldistanzmatrix (links) und die kumulierte Distanzmatrix (rechts) einer DTW-Pfadsuche 24

Abb. 2.15 - Beispiel für die kumulierte Distanzmatrix einer DTW-Pfadsuche mit variablem Anfangsfeld 25

Abb. 2.16 - Beispiele für globale Bereichsbegrenzungen beim DTW-Algorithmus links: Sakoe-Chiba-Band rechts: Itakura-Parallelogram 26

Abb. 2.17 - Schematischer Aufbau eines KNN mit vier Schichten 27

Abb. 2.18 - Schematische Darstellung eines vereinfachten Neuronenmodells 28

Abb. 2.19 - Schematischer Aufbau eines künstlichen neuronalen Netzes mit einem Bias-Neuron ... 28
Abb. 2.20 - Identische Abbildungsfunktion ... 29
Abb. 2.21 - ReLU-Funktion ... 29
Abb. 2.22 - Sprungfunktion ... 29
Abb. 2.23 - Tangens Hyperbolicus ... 29
Abb. 2.24 - Schematischer Vergleich des gewöhnlichen Backpropagation-Algorithmus mit resilient Backpropagation ... 32
Abb. 2.25 - Spektrogramme (links) und MFCC-Sequenzen (rechts) für zwei Instanzen des Wortes "eight" desselben Sprechers ... 33
Abb. 2.26 - Beispiel einer zweidimensionalen Faltung mit zwei verschiedenen Filtern ... 35
Abb. 2.27 - Beispiel eines Max-Pooling-Filters ... 36
Abb. 2.28 - Publish/Subscribe-Architektur bei MQTT ... 37
Abb. 3.1 - Gesamtaufbau des Spracherkennungssystems ... 38
Abb. 3.2 - Performance-Vergleich verschiedener Kopierfunktionen ... 41
Abb. 3.3 - Performance-Vergleich von drei Varianten zur Bestimmung des Leistungsspektrums ... 43
Abb. 3.4 - Verdrahtung des UDOO Neo mit dem Mikrofonmodul ... 44
Abb. 3.5 - Pol-Nullstellen-Diagramm des entworfenen Filters ... 45
Abb. 3.6 - Übertragungsfunktion des verwendeten digitalen Hochpassfilters ... 46
Abb. 3.7 - Erste kanonische Filterstruktur des entworfenen digitalen Hochpassfilters ... 46
Abb. 3.8 - Hamming-Fenster ... 47
Abb. 3.9 - Grafische Oberfläche der implementierten Demoanwendung ... 48
Abb. 3.10 - Schematische Darstellung der verwendeten DTW-Bereichsbegrenzung ... 49
Abb. 3.11 - Schematischer Aufbau des verwendeten neuronalen Netzes ... 51
Abb. 5.1 - Aufbau eines Spracherkennungssystems unter Verwendung der CMSIS-NN-Bibliothek ... 55

Tabellenverzeichnis

Tabelle 2.1 - Typische Topologien von neuronalen Netzen 32

Tabelle 3.1 Übersicht der verwendeten Parameter zur MFCC-Merkmalsextraktion 47

Tabelle 3.2 Übersicht über die Parameter des verwendeten neuronalen Netzes 52

Abkürzungsverzeichnis

ADC	Analog-to-Digital Converter
AMP	Asymmetric Multiprocessing
ANN	Artificial Neural Network, dt. Künstliches Neuronales Netz (KNN)
ASR	Automatic Speech Recognition, dt. automatische Spracherkennung
CMSIS	Cortex Microcontroller Software Interface Standard
CNN	Convolutional Neural Network
CSV	Comma Seperated Values
DCT	Discrete Cosine -Transform, dt. Diskrete Kosinustransformation
DFT	Diskrete Fouriertransformation
DSP	Digital Signal Processing oder Digitaler Signalprozessor
DTW	Dyamic Time Warp
EPIT	Enhanced Periodic Interrupt Timer
FFT	Fast Fourier Transform, dt. schnelle Fouriertransformation
FPU	Floating Point Unit
HMM	Hidden Markov Modell
I2S	Inter IC-Sound
IDE	Integrated Development Environment, dt. Integrierte Entwicklungsumgebung
IoT	Internet of Things, dt. Internet der Dinge
KNN	Künstliches Neuronales Netz
LPC	Linear Predictive Coding
MCU	Microcontroller Unit
MFCC	Mel Frequency Cepstral Coefficients
MQTT	Message Queuing Telemetry Transport
PLP	Perceptual Linear Predictive
QoS	Quality of Service
ReLU	Rectified Linear Unit
RPMsg	Remote Processor Messaging
RPROP	Resilient (Back-)Propagation
RTOS	Real-Time Operating System, dt. Echtzeitbetriebssystem
SIMD	Single Instruction Multiple Data
TCP	Transmission Control Protocol

1. Einleitung

Ein wichtiger Aspekt bei der Entwicklung neuer technischer Produkte ist die Benutzerschnittstelle. Während Eingaben früher vornehmlich durch mechanische Taster erfolgten, stehen heute vor allem kapazitive Berührungssensoren/Touchscreens im Vordergrund. Um die Benutzerschnittstelle noch komfortabler zu gestalten, geht der Trend mittlerweile aber auch zu komplett berührungslosen Alternativen wie der Sprachsteuerung. So haben sprachgesteuerte Systeme im Alltag vieler Menschen eine breite Verwendung gefunden und die großen Internetunternehmen bieten auch jeweils eigene Lösungen in diesem Bereich an. Der Smartphone-Markt wird vor allem von Apple´s Siri und dem Google Assistant dominiert, während im Bereich der Heimautomatisierung (Stichwort Smart Home) zusätzlich noch Amazon mit seinen Echo-Geräten eine große Rolle spielt. Die Funktionsweise ist bei den meisten dieser Systeme ähnlich: Das Gerät hört laufend seine Umgebung mit einem integrierten Mikrofon ab, um auf ein Schlüsselwort („Hey Google“, „Siri“, „Alexa“, etc.) reagieren zu können. Die Schlüsselworterkennung wird dabei in der Regel auf dem Gerät selbst berechnet, während alle darauffolgenden Sprachinformationen in die Cloud hochgeladen werden. Die Verarbeitung auf externen Servern bietet vor allem zwei technische Vorteile:

- Die Unternehmensserver bieten eine deutlich größere Rechenleistung, wodurch die Spracherkennung mit höherer Genauigkeit und in Echtzeit ablaufen kann.
- Die übermittelten Daten können zur Verbesserung des bestehenden Systems genutzt werden.

Allerdings entsteht dadurch automatisch auch eine gewisse Problematik rund um das Thema Datenschutz, da auch potenziell sensible Daten automatisch weitergeleitet werden können.

Die beschriebenen Systeme sollen einerseits immer leistungsfähiger werden, was durch immer bessere Hardware möglich wird, andererseits geht der Trend aber auch zur Nutzung von Geräten mit begrenzten Ressourcen. Beispielsweise werden im Rahmen des Internet of Things (IoT) immer mehr kleinere Geräte an das Internet angebunden, um mit anderen Geräten und Benutzern kommunizieren und interagieren zu können. Auch auf diesen Systemen lässt sich grundsätzlich eine Spracherkennung implementieren, welche die Sprachdaten nicht nur in die Cloud schickt, sondern diese lokal verarbeitet.

Je nach Anwendungsfall und der zur Verfügung stehenden Ressourcen existieren verschiedene kommerzielle und nicht-kommerzielle Bibliotheken für die Spracherkennung (wie z.B. Pocketsphinx). Diese arbeiten oft mit sogenannten Hidden Markov Modellen (HMM), welche das Sprachsignal als statistische Abfolge von Lauten oder Wörtern auffassen. Im

Rahmen dieser Abschlussarbeit wird hingegen ein anderer Ansatz verfolgt: Es soll ein komplett eigenständiges System auf Basis der Anforderungen im folgenden Kapitel entwickelt werden.

1.1. Aufgabenstellung

Das Ziel dieser Arbeit ist die Entwicklung eines Keyword-Spotting-Systems zur Erkennung von kurzen Wörtern und Kommandos auf einem Single-Board-Computer. Dieser soll mit einem Mikrofon ausgestattet werden und möglichst zuverlässig und vor allem in Echtzeit vordefinierte Worte erkennen. Das Ergebnis dieser Erkennung soll dann über das MQTT-Protokoll an einen Server geschickt werden, um eine mögliche Verwendung im „Internet der Dinge“ zu demonstrieren. Diese Anbindung hat nichts mit der zuvor beschriebenen Sprachverarbeitung in der Cloud zu tun. Das Spracherkennungssystem soll vollständig auf dem UDOO Neo Board laufen.

Als optionale Anforderung soll die Spracherkennung zudem noch sprecherunabhängig sein, das heißt das System soll von verschiedenen Personen verwendet werden können.

1.2. Verwendete Hardware

Als Zielplattform für das Spracherkennungssystem dient ein UDOO Neo Full Board. Dieses ist fest vorgegeben, da es unter anderem auch im Labor Embedded Systems II der Hochschule Offenburg verwendet wird. Das Board besitzt einen NXP i.MX 6SoloX Prozessor, welcher leistungsmäßig vergleichbar mit einem Raspberry Pi ist. Der Prozessor beinhaltet einen leistungsfähigen Cortex-A9 Prozessor, welcher zusätzlich durch einen Cortex-M4-Koprozessor unterstützt wird. Der M4-Prozessor ist dabei für Echtzeit- und I/O-Funktionen gedacht, während der Cortex-A9 rechenintensivere Anwendungen ausführen soll. Dementsprechend läuft auf dem Cortex-A9 auch das Linuxbasierte Betriebssystem UDOObuntu, während auf dem Cortex-M4 FreeRTOS zum Einsatz kommt. Das Board bietet außerdem zahlreiche Sensoren und Schnittstellen. Besonders erwähnenswert sind im Rahmen dieser Arbeit:

- Arduino-kompatible Pins, wodurch ein Aufsteckmodul auch für andere Boards wiederverwendet werden kann
- Micro-USB für die Stromversorgung und wahlweise auch zur Kommunikation mit einem PC
- Ethernet für eine zuverlässigere Kommunikation mit einem PC (vor allem für die Programmierung des Cortex-M4 und eine Remote-Desktop-Verbindung)
- Wi-Fi für die Verbindung zu einem MQTT-Broker

Ein sehr wichtiger Aspekt in einem Spracherkennungssystem ist natürlich auch das verwendete Mikrofon. Hier soll ein low-cost Elektret-Mikrofon mit einem MAX9814-Mikrofonverstärker eingesetzt werden. Dieser liefert eine analoge Ausgangsspannung, welche dann mit einem gewöhnlichen ADC eingelesen werden kann. Des Weiteren hat das Modul eine automatische Verstärkungsregelung, um eine Übersteuerung bei zu hohen Lautstärken zu verhindern. Damit werden weiter entfernte oder leise Geräusche in ähnlicher Lautstärke eingelesen wie nahe oder laute Geräusche, was je nach Art der Geräusche von Vor- oder Nachteil sein kann. Mithilfe dieses Mikrofonmoduls können Frequenzen im Bereich von 20 Hz bis 20 kHz aufgenommen werden, was den Gesamten für den Menschen hörbaren Bereich des Spektrums umfasst. Über zwei zusätzliche Pins kann bei diesem Verstärkermodul zudem von außen festgelegt werden, welche Standardverstärkung (40 dB, 50 dB oder 60 dB) und welches Verhältnis der Attack- und Release-Zeiten verwendet werden soll. Für nähere Informationen dazu sei hier lediglich auf das entsprechende Datenblatt verwiesen [1], da diese Optionen in dieser Arbeit keine entscheidende Verwendung finden. In Abb. 1.1 ist das UDOO Neo Board mit aufgestecktem Mikrofonmodul zu sehen.

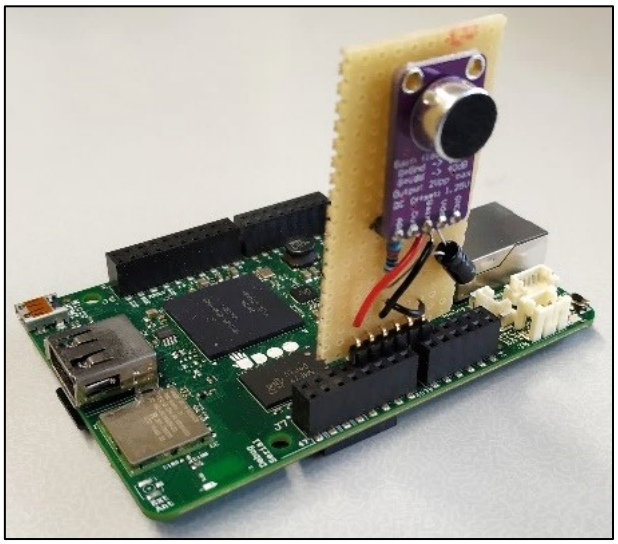


Abb. 1.1 - UDOO Neo Board mit aufgestecktem Mikrofonmodul

1.3. Verwendete Software

Im Rahmen dieser Arbeit werden einige Softwaretools zur Entwicklung des Spracherkennungssystems genutzt:

- Die Programmierung des M4-Cores auf dem UDOO Neo Board ist standardmäßig mit der Arduino IDE vorgesehen. Um eine größere Funktionalität nutzen zu können wird im Rahmen dieser Abschlussarbeit aber mit der **Eclipse IDE** sowie der **GNU Toolchain** gearbeitet. Damit die Spracherkennung parallel zur Interprozessorkommunikation stattfinden kann, wird **FreeRTOS** als Echtzeitbetriebssystem verwendet. Außerdem wird die **CMSIS-DSP-Bibliothek** verwendet, um Zugriff auf zahlreiche optimierte Funktionen rund um die digitale Signalverarbeitung zu erhalten. Nähere Betrachtungen zur DSP-Bibliothek befinden sich in Kapitel 3.1.
- Um die verarbeiteten Sprachinformationen vom Cortex-M4 zum Cortex-A9 zu übertragen wird die Schnittstelle **OpenAMP** bzw. das Linux-Framework **Remote Processor Messaging (RPMsg)** verwendet
- Für den A9-Core soll eine grafische Oberfläche mit **Qt-Creator** erstellt werden. Die Anwendung läuft dann auf dem Betriebssystem **UDOObuntu**.
- Als MQTT-Broker sowie für den zugehörigen Client auf der Zielhardware wird die Open-Source Software **Mosquitto** verwendet
- Für die Erstellung eines neuronalen Netzes (siehe Kapitel 3.4.2) wird die Software **MemBrain** verwendet. Diese ist für nicht-kommerzielle Zwecke kostenlos und ermöglicht den Aufbau des Netzes unter voller Kontrolle jedes einzelnen Neurons mithilfe einer intuitiv bedienbaren grafischen Oberfläche. Zudem können Trainings- und Testdaten über eine einfache csv-Schnittstelle (Comma Separated Values) eingelesen und für das Training zahlreiche Trainingsalgorithmen verwendet werden. Ein weiterer großer Pluspunkt des Programms ist ein integrierter Codegenerator, welcher universell einsetzbaren C-Code generiert, um das neuronale Netz auf dem UDOO Neo Board einzusetzen.
- In Kapitel 5 wird zusätzlich noch eine andere Variante zur Erstellung eines neuronalen Netzes vorgestellt. Dabei wird das **Tensorflow-Framework** von Google für das Design und Training des Netzes verwendet. Aufbauend auf Tensorflow stellt das Unternehmen ARM einige **Python**-Skripte bereit, mithilfe derer man das Modell dann auf Cortex-M-Prozessoren portieren kann.

Mit Ausnahme der beiden letzten Punkte bezüglich der neuronalen Netze sind die genannten Tools vorgegeben und zu Beginn der Arbeit auch schon größtenteils eingerichtet. Die restlichen Softwaremodule werden selbst implementiert (siehe Kapitel 3.2 bis 3.4.1).

2. Theoretische Grundlagen

In diesem Kapitel sollen zunächst die Grundlagen zur automatischen Spracherkennung betrachtet werden. Dabei wird erst auf die Entstehung und das Verständnis der menschlichen Sprache und danach auf die in dieser Arbeit verwendeten technischen Verfahren eingegangen.

2.1. Sprache und Hörverstehen

Die menschliche Sprache lässt sich auf vielen verschiedenen linguistischen Ebenen beschreiben. Für diese Arbeit ist besonders die phonemische Ebene von Bedeutung. Hier werden die vom Menschen produzierbaren Laute in sogenannte Phoneme eingeteilt, um verschiedene Bedeutungen zum Ausdruck zu bringen. In den meisten Sprachen (wie Deutsch, Englisch, etc.) sind die Phoneme dabei unabhängig von der Tonhöhe eines Lautes. Es gibt aber auch Sprachen (wie z.B. Chinesisch) in denen der Tonhöhenverlauf zur Bedeutung von Wörtern beiträgt und sich folglich eine noch genauere Einteilung der Phoneme ergibt. Wenn zwei verschiedene Laute zum selben Phonem gehören, weil sie dieselbe Bedeutung haben, so werden sie auch als Allophone bezeichnet [2]. Andere linguistische Ebenen wie beispielsweise die Syntax werden in dieser Arbeit nicht betrachtet, da lediglich ein Keyword-Spotting-System entwickelt wird. Es ist also irrelevant, was vor oder nach dem Schlüsselwort gesprochen wird, lediglich die zeitliche Abfolge der Phoneme ist interessant.

Da sehr viele Organe und der spezifische Körperaufbau für die Klangformung beim Menschen verantwortlich sind, hört sich jeder Mensch beim Sprechen auch unterschiedlich an. Die Entstehung der Sprache lässt sich in zwei getrennte Bereiche untergliedern [2]:

- Schallproduktion:
  Der eigentliche Schall wird bekanntlich durch die Stimmbänder erzeugt. Diese schwingen mit einer Grundfrequenz, die bei jedem Menschen etwas variieren kann. Ein deutlicher Unterschied ergibt sich bei den Grundfrequenzen der Geschlechter: Während die Stimmbänder bei Männern im Durchschnitt mit etwa 120 Hz schwingen, sind dies bei Frauen etwa 220 Hz. Bei Kindern liegt der Wert vor dem Stimmbruch in der Regel noch höher. Aufgrund von Anordnung und Schwingverhalten der Stimmbänder ergeben sich zusätzlich zur Grundfrequenz auch noch zahlreiche deutlich ausgeprägte Oberwellen, welche mit Vielfachen der Grundfrequenz schwingen.

- Klangformung

  Die Grundfrequenz mit ihren Oberschwingungen enthält zunächst keine Sprach- und nur wenige Sprecherinformationen. Diese kommen erst durch eine spezifische Klangformung hinzu. Dabei wird der in Schwingung versetzte Luftstrom durch den Vokaltrakt (Rachen, Mund und Nase) und die Artikulatoren (Zunge, Lippen, Kiefer, Zähne, Gaumen) so verändert, dass das resultierende Signal als Sprache aufgefasst werden kann. Diese Veränderung lässt sich näherungsweise als lineares Filter mit einer spezifischen Übertragungsfunktion interpretieren, welche auf das Anregungssignal angewendet wird. Dabei werden bestimmte Bereiche des Spektrums gedämpft, andere treten in Form von Resonanzfrequenzen hervor.

Um die gesprochene Sprache wieder interpretieren zu können, muss sie von einem geeigneten akustischen Empfänger aufgenommen werden. Das menschliche Ohr kann akustische Wellen im Bereich von etwa 20 Hz bis 20 kHz wahrnehmen. Die obere Grenze nimmt vor allem altersbedingt bei den meisten Menschen ab, weswegen ältere Leute sehr hohe Töne nicht mehr wahrnehmen können. Die Tonhöhe wird zudem nicht in einem linearen Maßstab, sondern logarithmisch wahrgenommen. Somit bewirkt eine Verdoppelung der Frequenz eine Tonerhöhung um eine Oktave. Auch die subjektiv empfundene Lautstärke hängt näherungsweise in logarithmischen Maßstab von der wirklichen Schallintensität ab, weswegen der Schalldruckpegel oft auch in Dezibel angegeben wird. Diese Besonderheiten müssen auch in einem Spracherkennungssystem berücksichtigt werden, da das Sprachverständnis des Menschen möglichst gut imitiert werden soll.

Für viele Audio-Anwendungen (vor allem in der Musikbranche) ist der gesamte hörbare Frequenzbereich wichtig, um einen möglichst authentischen Höreindruck zu erhalten. Deshalb wird auch bei Speichermedien wie der CD-ROM oft eine Abtastfrequenz von über 40 kHz verwendet. Für die Sprachverarbeitung ist dies aber nicht nötig und kann die Spracherkennung sogar negativ beeinflussen. Der Grund dafür ist, dass sich die Informationen im Sprachsignal auf einen viel kleineren Bereich des Frequenzspektrums konzentrieren. Abb. 2.1 zeigt die Wichtigkeit der einzelnen Oktavbänder für die Sprachverständlichkeit. Dargestellt sind dabei lediglich die Mittenfrequenzen, welche sich anhand der folgenden Formel berechnen lassen:

$$f_M = \sqrt{f_u \cdot f_o} \tag{2.1}$$

f_u: untere Grenze des Oktavbandes
f_o: obere Grenze des Oktavbandes

Die obere Bandgrenze hat in der Formel die doppelte Frequenz der unteren Bandgrenze.

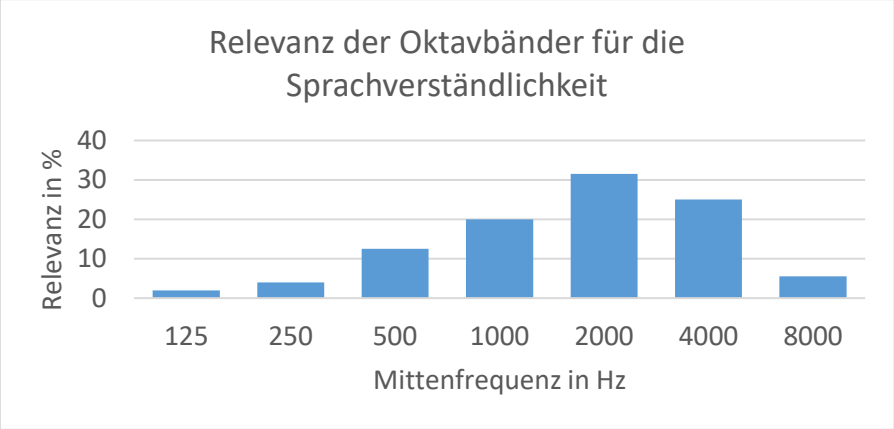


Abb. 2.1 - Relevanz verschiedener Oktavbänder für die Sprachverständlichkeit [3]

Die Abbildung zeigt, dass allein die Bänder mit den Mittenfrequenzen von 500 Hz bis 4 kHz ausreichen, um den Großteil der menschlichen Sprache verstehen zu können. Entsprechend haben Frequenzen oberhalb von 8 kHz einen sehr geringen Einfluss auf das Sprachverständnis, weswegen dies auch als Grenze für das zu entwickelte Spracherkennungssystem gewählt wird (siehe Kapitel 3).

2.2. Signalvorverarbeitung

Die Signalvorverarbeitung hat die Aufgabe, das Signal eines Mikrofoneingangs für die darauffolgende Merkmalsextraktion so aufzubereiten, dass störende Signalanteile entfernt oder zumindest reduziert werden. Dazu gehört unter anderem:

- Veränderung des Signalpegels
- Reduzierung des Rauschens
- Verkleinerung des Frequenzbereichs auf den für die Sprachverarbeitung relevanten Bereich
- Verhinderung von Aliasing

Durch Beachtung der in den Unterkapiteln beschriebenen Schritte kann die Spracherkennung signifikant verbessert werden.

2.2.1. Signalverstärkung

Je nach Typ des verwendeten Mikrofons hat das elektrische Signal nur einen Aussteuerungsbereich im Millivoltbereich. Würde man ein solches Signal direkt auf einen gewöhnlichen Analog-Digital-Wandler geben, so wäre das Quantisierungsrauschen im Vergleich zum Informationssignal sehr groß. Deshalb muss das Audiosignal vorher noch mithilfe eines Operationsverstärkers so verstärkt werden, dass möglichst der gesamte Aussteuerbereich des AD-Wandlers verwendet wird. Da viele Mikrofonmodule bereits einen brauchbaren Verstärker beinhalten, wird hier nicht näher auf dessen Auslegung eingegangen.

2.2.2. Tiefpassfilterung

Das menschliche Gehör ist in der Lage Frequenzanteile bis etwa 20 kHz zu hören. Deshalb sind auch viele gebräuchliche Mikrofone bis zu diesem Wert ausgelegt. Für die Verständlichkeit der menschlichen Sprache sind aber im Wesentlichen nur die Signalanteile bis ca. 5 kHz verantwortlich [2]. Höhere Frequenzen sollten vor der Merkmalsextraktion aus dem Audiosignal herausgefiltert werden. Diese Tiefpassfilterung kann grundsätzlich analog oder digital erfolgen, jedoch sollte ersteres hier bevorzugt werden, da es sonst bei zu geringer Abtastfrequenz zum sogenannten Aliasing kommen kann, was im Folgenden näher erläutert wird.

Ein abgetastetes Signal $x[n]$ ist zeitdiskret und enthält die Werte des analogen Signals zu den Abtastzeitpunkten (siehe Abb. 2.2).

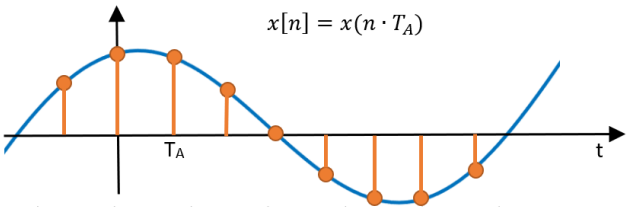


Abb. 2.2 - Abtastung eines Sinussignals

Mathematisch kann die Abtastung auch als Multiplikation einer Dirac-Impulsfolge mit dem zeitkontinuierlichen Signal $x(t)$ interpretiert werden:

$$x_A(t) = T_A \cdot x(t) \cdot \sum_{n=-\infty}^{\infty} \delta(t - n \cdot T_A) \tag{2.2}$$

$\delta(t)$:	Dirac-Impuls
$x_A(t)$:	abgetastetes Signal
T_A:	Abtastzeit

Durch eine Fouriertransformation ergibt sich daraus im Frequenzbereich folgendes Spektrum des abgetasteten Signals:

$$X_A(\omega) = \sum_{n=-\infty}^{\infty} X(\omega - n \cdot \omega_A) \tag{2.3}$$

$X(\omega)$: Spektrum von $x(t)$
ω_A: Abtastkreisfrequenz

Das Spektrum des abgetasteten Signals entspricht also einer periodischen Wiederholung des Originalspektrums bei Vielfachen der Abtastfrequenz, wie es grafisch in Abb. 2.3 dargestellt ist.

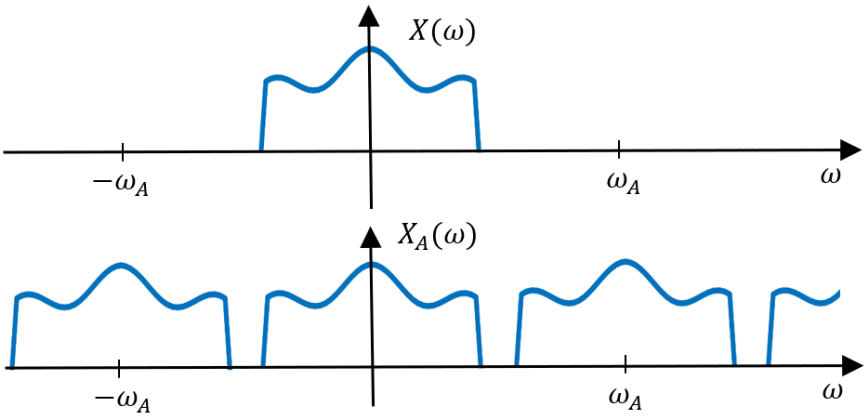


Abb. 2.3 - Darstellung des Aliasing-Effekts im Spektrum

Das Nyquist-Shannon-Abtasttheorem besagt, dass im Signal vor der Abtastung nur Frequenzen bis maximal zur halben Abtastfrequenz vorhanden sein dürfen, um das Signal später wieder exakt rekonstruieren zu können. Oder anders ausgedrückt: Wenn das Signal Frequenzen oberhalb der halben Abtastfrequenz enthält, überlappen sich die durch die Abtastung entstehenden Spiegelfrequenzen und es kommt zu Aliasing. Dabei wird das informationstragende Spektrum verfälscht, was danach nicht mehr umkehrbar ist. Um dies zu verhindern sollte das Audiosignal schon vor der Abtastung mit einem geeigneten analogen Tiefpass (Antialiasing-Filter) gefiltert werden.

Als analoger Tiefpass kann beispielsweise ein simples RC-Glied verwendet werden (siehe Abb. 2.4), welches folgende Spannungs-Übertragungsfunktion besitzt:

$$V_{out} = V_{in} \cdot \frac{1}{j\omega RC + 1} \tag{2.4}$$

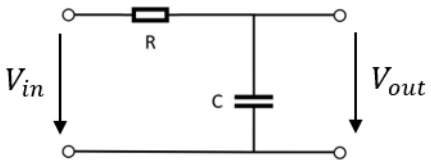


Abb. 2.4 - Passiver Tiefpass als einfaches RC-Glied

Ein solcher passiver Tiefpass ist durch die Grenzfrequenz f_g gekennzeichnet, bei welcher die Signalamplitude im unbelasteten Fall um den Faktor $\frac{1}{\sqrt{2}}$ gedämpft wird. Frequenzen darüber werden zunehmend stärker gedämpft (20 dB pro Dekade).

$$f_g = \frac{\omega_g}{2\pi} = \frac{1}{2\pi \cdot R \cdot C} \tag{2.5}$$

2.2.3. Analog-Digital-Wandlung

Wenn das Audiosignal nicht über eine digitale Schnittstelle (z.B. I2S) sondern über einen AD-Wandler eingelesen werden soll, so benötigt dieses einen Gleichspannungsanteil, der so hoch ist, dass keine negativen Spannungen am Eingang des AD-Wandlers auftreten, der Bereich auch nicht übersteuert wird. Falls das verwendete Mikrofonmodul dies nicht schon standardmäßig implementiert, kann das Signal über einen zusätzlichen Spannungsteiler eingekoppelt werden. Die Auflösung spielt im Bereich der üblichen AD-Wandler keine große Rolle für die Genauigkeit der späteren Spracherkennung. Wichtig ist aber wie in Kapitel 2.2.2 erläutert, dass das Audiosignal mit einer Frequenz abgetastet wird, die ausreichend hoch ist, sodass kein Aliasing auftritt.

2.2.4. Hochpassfilterung

Um den Gleichsignalanteil nach der Abtastung wieder zu eliminieren, wird das Audiosignal üblicherweise mit einem digitalen Hochpass gefiltert. Wenn dabei möglichst keine anderen Frequenzen beeinflusst werden sollen, wird hierfür ein Filter mit möglichst steil ansteigender Übertragungsfunktion benötigt, was sich im Pol-Nullstellen-Diagramm der z-Übertragungsfunktion dadurch ausdrückt, dass mindestens ein Pol und eine Nullstelle dicht beieinander sind (siehe Abb. 2.5). Ein solches Filter besitzt beispielsweise die Übertragungsfunktion:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1 - z^{-1}}{1 - \beta \cdot z^{-1}} \tag{2.6}$$

$X(z)$: z-Transformierte der Eingangsdaten
$Y(z)$: z-Transformierte der Ausgangsdaten

Der Koeffizient β ist hierbei ein Maß für die Steilheit der Übertragungsfunktion. Je näher dieser Wert bei eins liegt, desto steiler ist die Flanke in Abb. 2.6 . Durch Umformen und anschließende Rücktransformation erhält man die Implementierungsstruktur gemäß der Formel:

$$y[n] = x[n] - x[n-1] + \beta \cdot y[n-1] \qquad n \in \mathbb{Z} \tag{2.7}$$

Eine alternative Implementierung, die mit nur einem Speicherbaustein auskommt, ist in Abb. 2.7 dargestellt.

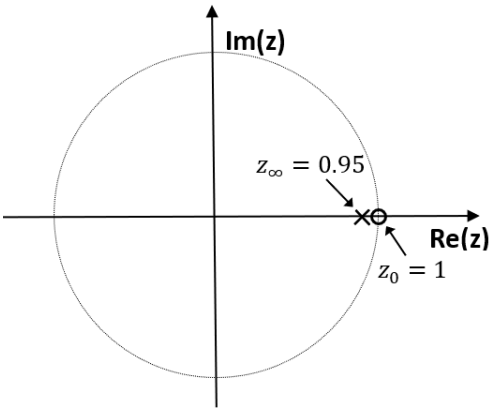


Abb. 2.5 - Pol-Nullstellen-Diagramm eines digitalen Filters zur Entfernung des Gleichsignalanteils

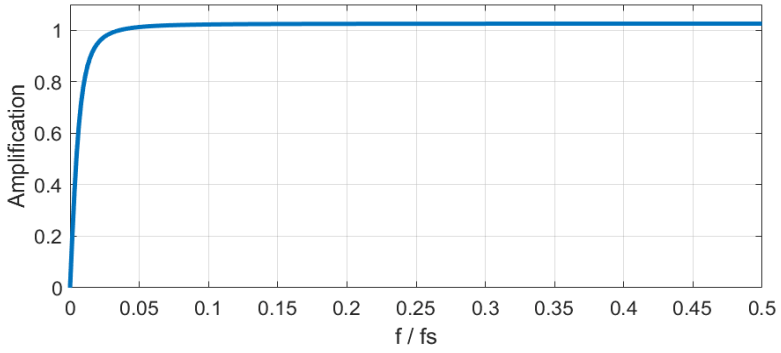


Abb. 2.6 - Übertragungsfunktion eines digitalen Filters zur Entfernung des Gleichsignalanteils

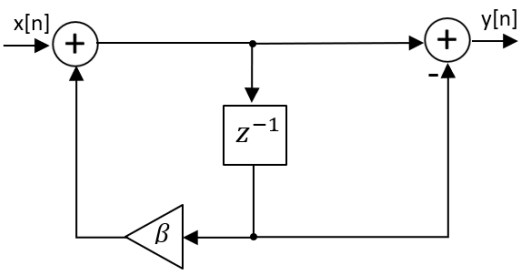


Abb. 2.7 - Implementierungsstruktur eines digitalen Filters zur Entfernung des Gleichsignalanteils [4]

Die höherfrequenten Signalkomponenten sind in vielen Spracherkennungssystemen durch die Einflüsse des Übertragungskanals sowie des Antialiasing-Filters gedämpft. Für die Effizienz der später angewendeten diskreten Kosinus Transformation (DCT) ist es aber wünschenswert, wenn die Einhüllende des Spektrums möglichst flach ist [5]. Deshalb sollten die höheren Frequenzen mithilfe eines sogenannte **Präemphase-Filters** (engl. Preemphasis filter) verstärkt werden. Die Übertragungsfunktion eines solchen Filters ist gegeben durch:

$$H(z) = \frac{Y(z)}{X(z)} = 1 - \alpha \cdot z^{-1} \tag{2.8}$$

Typische Werte für α liegen auch hier nahe unterhalb von eins [5]. Abb. 2.8 zeigt das Pol-Nullstellen-Diagramm (links) sowie den Amplitudengang (rechts) eines Präemphase-Filters für $\alpha = 1$, wobei der Gleichanteil hierbei komplett unterdrückt wird und damit das zuvor beschriebene Filter zur Eliminierung des Gleichanteils überflüssig wird. Die z-Rücktransformation ergibt für die Implementierung eine einfache Differenzbildung:

$$y[n] = x[n] - \alpha \cdot x[n-1] \qquad n \in \mathbb{Z} \tag{2.9}$$

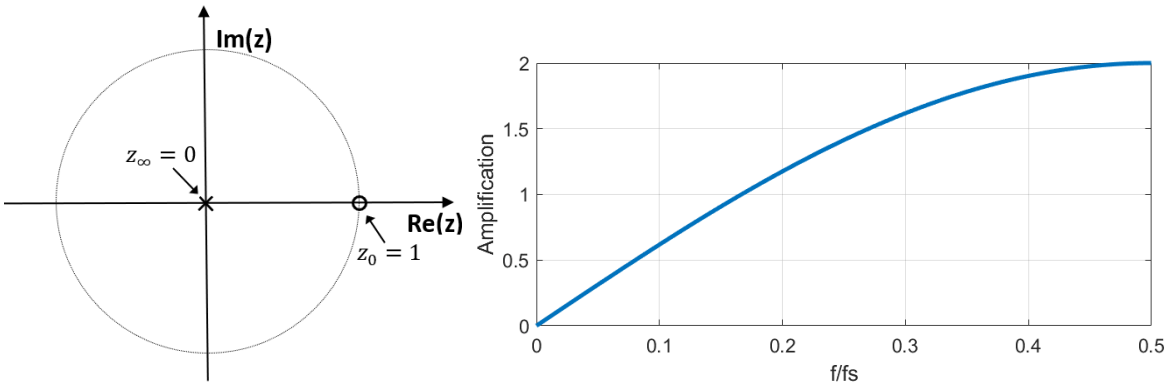


Abb. 2.8 - Pol-Nullstellen-Diagramm (links) und Übertragungsfunktion (rechts) eines Präemphase-Filters

2.3. MFCC-Merkmalsextraktion

Audiosignale stellen den Schalldruckpegel im zeitlichen Verlauf dar. Ein für den Menschen interpretierbares Audiosignal weist aber die Besonderheit auf, dass die enthaltenen Informationen nicht direkt aus dem Zeitsignal gewonnen werden können, da das menschliche Ohr nicht den Schalldruckpegel selbst, sondern die Überlagerung verschiedener Sinussignale unterschiedlicher Frequenz wahrnehmen kann. Abb. 2.9 (links) zeigt das Audiosignal eines gesprochenen Phonems. Aus diesem lässt sich ohne weiteres nur der zeitliche Verlauf des

Schalldruckpegels bzw. der damit verbundenen Lautstärke erfassen. Um weitere Informationen über den Laut zu erhalten, muss das Signal deshalb in den Frequenzbereich transformiert und dort weiter untersucht werden. Auf der rechten Seite der Abbildung ist die Fouriertransformation des Audiosignals dargestellt, in welcher man erkennen kann, dass bestimmte Frequenzen besonders stark im Signal vertreten sind.

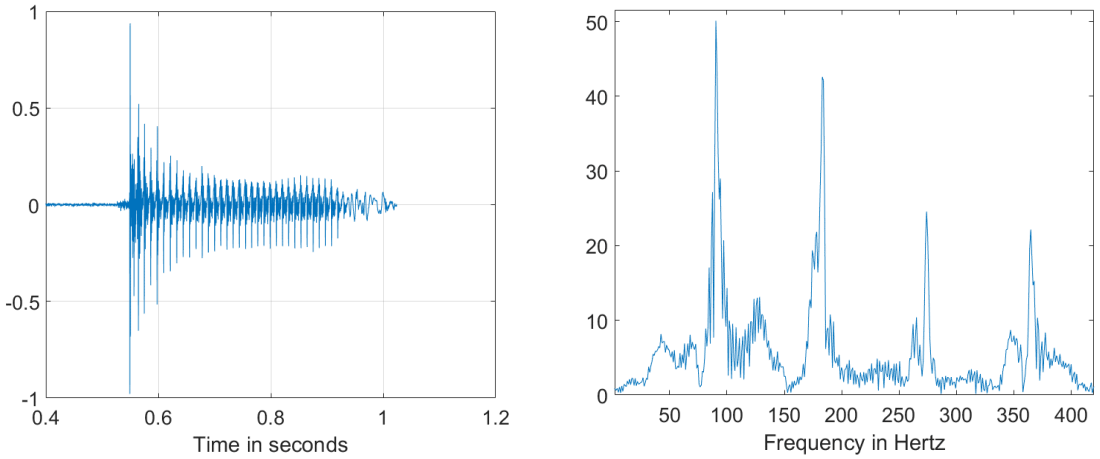


Abb. 2.9 - Audiosignal eines einzelnen Lautes (links) und das dazugehörige Spektrum (rechts)

Für die Klassifizierung eines Wortes oder eines Lautes ist aber nicht jede Information im Spektrum interessant. Zum Beispiel könnte man sich nur für einen bestimmten Frequenzbereich interessieren, oder die Frequenzauflösung ist höher als eigentlich benötigt. Um in der weiteren Verarbeitung nicht Rechenzeit und Speicher zu verschwenden, ist es daher sinnvoll, die Frequenzinformationen einer geeigneten Datenkompression zu unterziehen. Man versucht also, dem Sprachsignal möglichst signifikante Merkmale zu entnehmen, um die weitere Verarbeitung zu verbessern und effizienter zu machen. Dieser gesamte Vorgang wird deshalb auch als Merkmalsextraktion (engl. „feature extraction“) und das Resultat als Merkmalsvektor bezeichnet.

Als Resultat der Merkmalsextraktion steht somit ein Vektor, welcher die Informationen des Sprachsignals in möglichst kompakter Form darstellen soll. Für die heutige Sprachverarbeitung kommen im Wesentlichen drei verschiedene Verfahren zur Merkmalsextraktion zum Einsatz:

- Linear Predictive Coding (LPC)
- Mel Frequency Cepstral Coefficients (MFCC)
- Perceptual Linear Predictive (PLP, und die Erweiterung RASTA-PLP)

Im Rahmen dieser Arbeit wird lediglich die **MFCC-Merkmalsextraktion** verwendet, da sie auch in einer Vielzahl anderer Anwendungen erfolgreich eingesetzt wird.

Die Grundidee bei der Berechnung der MFCC-Merkmalsvektoren ist es, das menschliche Hörverhalten nachzubilden. Damit ein Computer Wörter wie ein Mensch erkennen kann, sollte er auch Informationen über die Funktionsweise des Hörens besitzen. Abb. 2.10 zeigt den groben Ablauf, um die Mel-Frequenz-Cepstrum-Koeffizienten zu berechnen. Je nach Anwendung und gewünschter Informationsmenge können dabei im Ergebnis unterschiedliche Merkmale enthalten sein. Im Folgenden wird das grundsätzliche Vorgehen beschrieben.

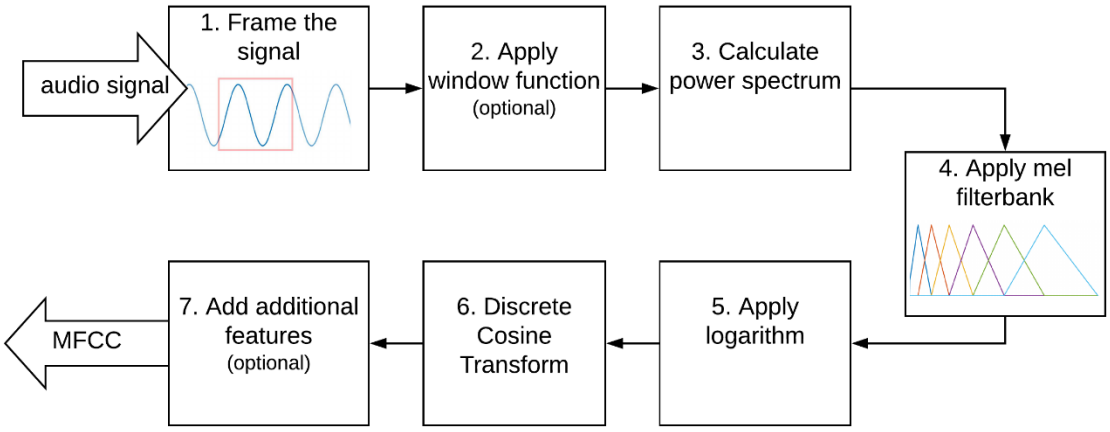


Abb. 2.10 - Grober Ablauf der MFCC-Merkmalsberechnung

2.3.1. Framing

Die Datenverarbeitung erfolgt üblicherweise nicht auf einem kontinuierlichen Fluss von Daten, sondern blockweise. Die Audiosamples werden in Blöcke von etwa 20 ms bis 40 ms eingeteilt [6]. Eine zu kurze Blockdauer führt dabei zu einer unzuverlässigen Schätzung des Spektrums, während eine zu große Blockdauer eine Mittelung über verschiedene Laute bewirkt. Die verschiedenen Blöcke sollten sich auch überlappen (z.B. um 50%), damit bestimmte Signalanteile nicht so stark durch die später angewendete Fensterfunktion unterdrückt werden.

Mathematisch kann der Abtastwert im i-ten Block an der n-ten Stelle wie folgt aus den Eingangsdaten $x_0[n]$ berechnet werden:

$$x_i[n] = x_0\left[\, i \cdot (\, N_b - \, N_{overlap}) + n\right] \qquad 0 \leq i \quad , \quad 0 \leq n < N_b \qquad (2.10)$$

N_b: Blocklänge

$N_{overlap}$: Anzahl überlappender Samples

2.3.2. Fensterfunktion

Um das Spektrum eines (als stationär angenommenen) Signals exakt zu bestimmen, müsste man die Fouriertransformierte des Signals über einen langen Zeitraum berechnen, was aus Aufwandsgründen nicht möglich ist. Deshalb wird das Signal wie schon erwähnt in Blöcke eingeteilt, was allerdings rechnerisch der Multiplikation mit einer Rechteckfunktion entspricht. Diese Multiplikation spiegelt sich im Frequenzbereich in einer Faltung mit einer Sinus-Cardinalis-Funktion wieder, wodurch man nicht mehr das exakte Spektrum, sondern nur noch eine Schätzung dafür erhält (Ausnahme: Die Fensterbreite ist ein ganzzahliges Vielfaches der Periodendauer). Diese Schätzung wird auch Periodogramm genannt. Der dadurch entstehende Fehler wird als sogenannter Leck-Effekt bezeichnet, da die Signalenergie, die eigentlich einem bestimmten Frequenzband zugeordnet ist, in nebenstehende Frequenzbänder „abfließt".

Das Ausmaß des Leck-Effekts kann durch die Wahl einer geeigneten Fensterfunktion erheblich reduziert werden. Die Fensterfunktion f_w hat die gleiche Länge wie die Datenblöcke und wird mit den Abtastwerten des Audiosignals multipliziert.

$$y_i[n] = x_i[n] \cdot f_w[n] \qquad 0 \leq i \quad , \quad 0 \leq n < N_b \tag{2.11}$$

2.3.3. Leistungsspektrum

Da die eigentliche Nutzinformation dem Spektrum eines Audiosignals zu entnehmen ist, werden die Datenblöcke einzeln fouriertransformiert. Das Ergebnis dieser Transformation ist im Allgemeinen komplexwertig und besitzt einen Betrag und eine Phase. Die Phaseninformation kann aber verworfen werden, weil man nur an der spektralen Zusammensetzung eines Datenblocks interessiert ist und nicht an der zeitlichen Abfolge der spektralen Komponenten. Somit erhält man durch die Bildung des Betrags ein reellwertiges Betragsspektrum. Grundsätzlich ist es egal, ob man dieses danach noch quadriert, um das Leistungsspektrum zu erhalten, da alle Werte später noch logarithmiert werden. Deshalb unterscheidet sich das Ergebnis bei der Verwendung des Betragsspektrums anstelle des Leistungsspektrums nur um den Faktor Zwei. Eine Normierung des Leistungsspektrums ist für die Berechnung der MFCC-Merkmale ebenfalls nicht notwendig.

Zuerst muss die Fouriertransformierte berechnet werden:

$$Y_i[k] = \sum_{n=0}^{N_b-1} y_i[n] \cdot e^{-j \cdot \frac{2\pi nk}{N_b}} \qquad 0 \leq i \quad , \quad 0 \leq k < N_b \tag{2.12}$$

Danach erfolgt die Berechnung des Leistungsspektrums:

$$P_i[k] = |Y_i[k]|^2 \qquad 0 \leq i \quad , \quad 0 \leq k < N_b \tag{2.13}$$

Aufgrund dessen, dass die Eingangsdaten reellwertig sind, ist die Fouriertransformierte konjugiert symmetrisch, bzw. auch symmetrisch zur Ordinatenachse sobald man den Betrag bildet. Da diese Symmetrie zu redundanten Informationen im Spektrum führt, kann die Hälfte des Spektrums verworfen werden. Diese Datenreduktion führt später auch zu einer erheblichen Einsparung an Rechenzeit.

Hinweis: $P_i[k]$ schätzt das Leistungsspektrum eines Leistungssignals. Wenn im Folgenden von Energien die Rede ist, bezieht sich dies auf die Signalenergie des Spektrums, da dieses auch als neues Signal aufgefasst werden kann.

2.3.4. Mel-Filterbank

Die Fouriertransformation liefert am Ende des vorherigen Schrittes einen reellwertigen Vektor, dessen Datenpunkte die Energien in verschiedenen Frequenzbändern repräsentieren. Diese sind in äquidistanten Frequenzabständen Δf (auch Frequenzauflösung genannt) verteilt, welche sich wie folgt berechnen lassen:

$$\Delta f = \frac{F_s}{N_b} \tag{2.14}$$

F_s: Abtastfrequenz in Hz
N_b: Blocklänge

In einem Diagramm wäre die Frequenzachse also entsprechend linear skaliert. Das menschliche Hörempfinden verläuft aber keineswegs linear entlang der Frequenzachse. Vielmehr kann ein Mensch zwei Töne in einem niederfrequenten Bereich viel besser unterscheiden, als in einem höherfrequenten Bereich (solange der absolute Frequenzabstand gleich ist). Dieses Verhalten wird näherungsweise durch eine logarithmische Skalierung der Frequenzachse modelliert. Um die subjektiv empfundene Tonhöhe wiederum linear darzustellen, wird die sogenannte Mel-Skala eingeführt.

Die Umrechnung von Hz in Mel erfolgt durch:

$$m(f) = 2595mel \cdot log_{10}\left(1 + \frac{f}{700Hz}\right) \tag{2.15}$$

Um von Mel wieder zu Hertz zu kommen, lässt sich folgende Formel anwenden:

$$f(m) = 700Hz \cdot 10^{\left(\frac{m}{2595mel} - 1\right)} \tag{2.16}$$

Ein weiterer Aspekt des menschlichen Hörempfindens ist, dass eng benachbarte Frequenzen generell schlecht unterschieden werden können und zu Frequenzgruppen zusammengefasst werden [2]. Deshalb werden die Spektren der einzelnen Datenblöcke mithilfe von Dreiecksfiltern ebenfalls zu Gruppen zusammengefasst. Die Breite dieser Dreiecke steigt logarithmisch mit der Frequenz in Hertz (siehe Abb. 2.11), was auf der Mel-Skala einer konstanten Breite entspricht. Da die Dreiecke rechnerisch mit den Spektren multipliziert werden, entspricht dies einer Bandpassfilterung des ursprünglichen Audiosignals mit verschiedenen Bandpässen. Die Dreiecke sollten sich überlappen, um an den Rändern keine Informationen zu verlieren. Trägt man alle Dreiecksfilter zusammen in einem Diagramm auf, spricht man von einer Mel-Filterbank, welche in Abb. 2.11 exemplarisch für zehn Filter dargestellt ist.

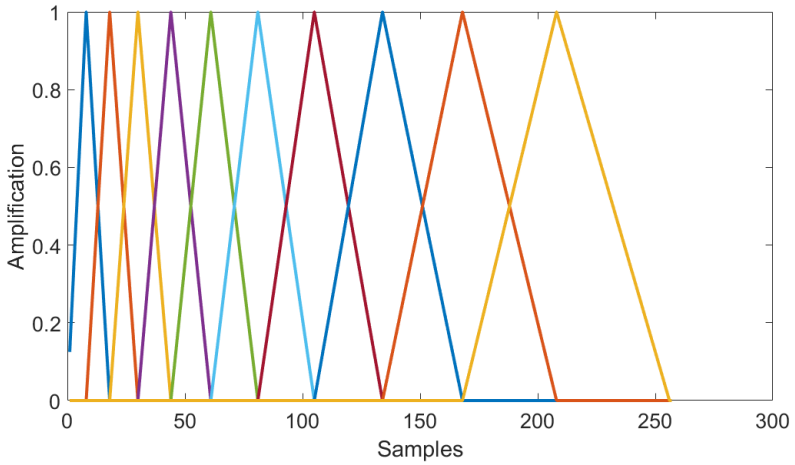


Abb. 2.11 - Darstellung einer Mel-Filterbank mit zehn Dreiecksfiltern

Für jedes Filter (und pro Block) erhält man am Ende einen einzigen Wert, da die spektralen Komponenten nach der Filterung entlang der Frequenzachse aufsummiert werden. So hätte man nach Anwendung einer Mel-Filterbank mit 30 Filtern für jeden Block auch 30 reelle Werte, welche die spektralen Energien in den entsprechenden Frequenzbändern repräsentieren und sich wie folgt berechnen lassen:

$$E_i(k) = \sum_{n=0}^{0{,}5 \cdot N_b} P_i[n] \cdot H_k[n] \qquad 0 \leq i \quad , \quad 0 \leq k < M \tag{2.17}$$

$P_i[n]$:	Leistungsspektrum des i-ten Blocks
$H_k[n]$:	k-te Dreiecksfunktion
N_b:	Blocklänge
M:	Anzahl Filter

Die einzelnen Dreiecksfilter ergeben sich anhand folgender Formel [6]:

$$H_k[n] = \begin{cases} \dfrac{n - f[k-1]}{f[k] - f[k-1]} & falls\ f[k-1] \leq n \leq f[k] \\ \dfrac{f[k+1] - n}{f[k+1] - f[k]} & falls\ f[k] \leq n \leq f[k+1] \\ 0 & sonst \end{cases} \tag{2.18}$$

Die Funktion $f[k]$ stellt dabei eine Frequenz-Liste (in Hertz) der Größe $M + 2$ dar, in welcher die Frequenzen äquidistante Abstände in Mel haben. Oder anders ausgedrückt enthält $f[k]$ alle Frequenzen, durch welche die Dreiecke in Abb. 2.11 begrenzt sind. Dadurch sind automatisch auch die Mittenfrequenzen der Dreiecke enthalten.

2.3.5. Logarithmus

Auch die Lautstärke eines Audiosignals wird vom Menschen nicht linear zum Schalldruckpegel, sondern näherungsweise logarithmisch wahrgenommen. Deshalb werden die Signalenergien in diesem Schritt logarithmiert:

$$E_{log,i}(k) = log\big(E_i(k)\big) \qquad 0 \leq i \quad , \quad 0 \leq k < M \tag{2.19}$$

Beim Berechnen des Logarithmus für jede Signalenergie eines Dreiecksfilters muss darauf geachtet werden, dass der entsprechende Wert nicht null ist, da der Logarithmus nur für positive Werte definiert ist. Falls ein nicht-positiver Wert auftritt, sollte dieser vor dem Logarithmieren auf einen sehr kleinen positiven Wert nahe null gesetzt werden.

2.3.6. Diskrete Kosinustransformation (DCT)

Die DCT transformiert ein Signal aus dem Originalbereich (oft Zeitbereich) in den Frequenzbereich. Sie hat also grundsätzlich eine ähnliche Aufgabe wie die DFT.

Bei der Anwendung der DFT geht man immer davon aus, dass das Signal in Zeit- und Frequenzbereich periodisch fortgesetzt wird. Die periodische Fortsetzung des Signals $F_{log,i}$ eines jeden Blocks ergibt aber keinen Sinn, weil man (insbesondere durch das Weglassen des halben Spektrums aus dem dritten Schritt) die Funktionswerte an den beiden Rändern unabhängig voneinander gemacht hat. Aus diesem Grund benutzt man hier stattdessen die DCT-Typ 2.

Die DCT-II lässt sich wie folgt berechnen [7]:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot cos\left(\frac{\pi}{N} \cdot \left(n + \frac{1}{2}\right) \cdot k\right) \qquad 0 \leq k < N \qquad (2.20)$$

N: Anzahl Werte in $x[n]$

Hinweis: Man unterscheidet vier Typen der DCT, die sich in der Art ihrer impliziten periodischen Fortsetzung unterscheiden. Der hier verwendete DCT-Typ 2 (oder DCT-II) ist die übliche Variante, bei welcher das Signal an dessen Ende implizit in gespiegelter Form wiederholt wird. Somit gibt es an den Rändern keine Sprungstellen.

Alternativ kann man die DCT auch aus dem Ergebnis einer DFT berechnen. Dies kann Sinn machen, wenn auf einem Zielsystem optimierte Verfahren für die DFT verfügbar sind.

Die Anwendung der DCT auf die logarithmierten Signalenergien aus Schritt fünf bewirkt eine verlustbehaftete Datenkompression, sofern man beim Resultat die höherwertigen Komponenten vernachlässigt (bzw. gleich null setzt). Abb. 2.12 zeigt dieses Prinzip anhand eines Beispiels: Das Signal mit den vielen Spitzenwerten wird zunächst DCT-transformiert. Beim Ergebnis werden dann die höherwertigen Koeffizienten auf null gesetzt und dass Signal danach wieder zurücktransformiert. Als Resultat ergibt sich das dargestellte Signal ohne die Peaks, welches man auch als eine Art Mittelung verstehen kann.

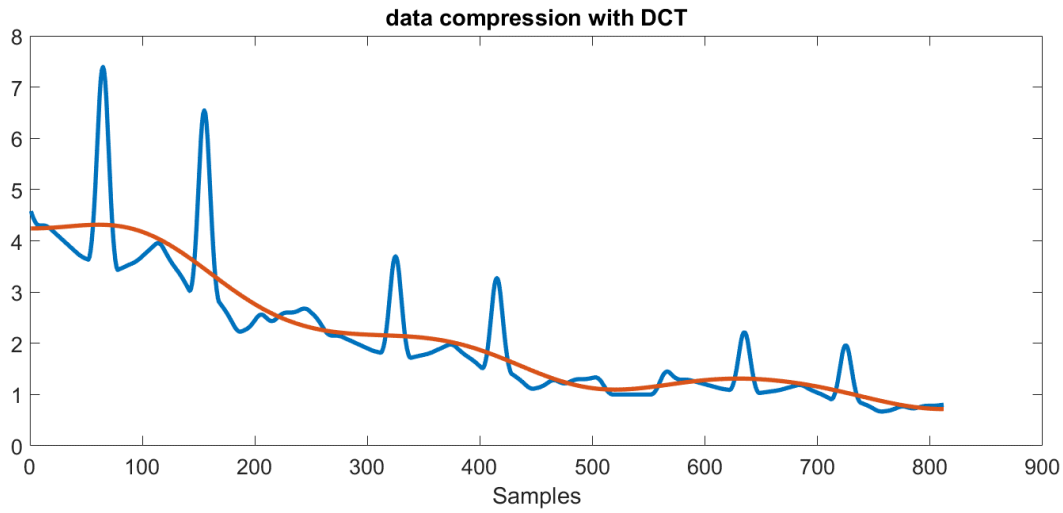


Abb. 2.12 - Schematische Darstellung der Datenkompression mit einer DCT

Die behaltenen niederwertigen Datenpunkte der DCT repräsentieren langsame Änderungen des Originalsignals. Die hochfrequenten Anteile, die durch die Kompression auf null gesetzt werden, sind für die Spracherkennung sowieso eher störend, da man nur am groben Verlauf bzw. der Hüllkurve der Signalenergien interessiert ist. Das Weglassen dieser Anteile kann die Spracherkennung deshalb sogar noch verbessern. Üblicherweise werden nur die ersten 12 bis 13 DCT-Werte berechnet [6], welche man dann auch als Mel-Frequency-Cepstral-Coefficients (MFCC) bezeichnet.

Unabhängig von der Anwendung spricht man von einem Cepstrum, wenn ein Originalsignal in den Frequenzbereich transformiert, dort logarithmiert und anschließend wieder zurück transformiert wird. Wenn ein Zeitsignal durch ein lineares, zeitinvariantes System läuft, entspricht dies einer Faltung des Signals mit der Impulsantwort des Systems, bzw. im Frequenzbereich einer Multiplikation der Fouriertransformierten mit der Übertragungsfunktion. Durch die Logarithmierung im Frequenzbereich geht diese Multiplikation aber in eine Addition über. Das Besondere am Cepstrum ist nun, dass diese Addition auch bei der Rücktransformation erhalten bleibt, solange man lineare Transformationen anwendet. Dadurch kann im Cepstrum das ursprüngliche Anregungssignal vom durchlaufenen Filter durch eine einfache Subtraktion getrennt werden, wenn eines von beiden bekannt ist.

Dieses Prinzip wird auch bei der Berechnung der MFCC angewendet: Das Ergebnis der DCT (also das Cepstrum) enthält Informationen über das Anregungssignal, wie auch über das durchlaufene Filter. Das Anregungssignal wird durch die Stimmbänder generiert und besteht (bei stimmhaften Lauten) aus der Überlagerung verschiedener harmonischer Schwingungen, oder anders ausgedrückt aus einer Grundfrequenz und deren Vielfachen. Dieses Signal durchläuft dann den menschlichen Vokaltrakt (Rachen, Mund, Nase, etc.), bevor es von einer

anderen Person interpretiert werden kann. Dieser Vorgang kann näherungsweise als Filterung mit einem linearen Filter interpretiert werden. Das Anregungssignal ist bei jedem Menschen unterschiedlich, da sich zum einen die Grundfrequenz und zum anderen die Stärke der Oberschwingungen unterscheiden. Der Einfluss des Vokaltraktes hingegen kann als ähnlich betrachtet werden, weshalb man für die automatische Spracherkennung diese beiden Teile gerne trennen würde. Bei der MFCC-Berechnung wird dies in gewissem Maße durch das Weglassen der höherwertigen Koeffizienten realisiert, weil in diesen unter anderem die Information über das Anregungssignal steckt.

2.3.7. Zusätzliche Merkmale

Nachdem die MFCC-Werte berechnet wurden, können je nach gewünschter Funktionalität des Spracherkennungssystems Merkmale hinzugefügt oder entfernt werden. Manchmal ist es sinnvoll, die Signalenergie eines Blockes des Audiosignals dem Merkmalsvektor hinzuzufügen. Oft ist aber auch genau dies störend, da die Signalenergie ein Indikator für die Lautstärke ist, diese aber bei demselben Laut in einem großen Bereich variieren kann. Der erste MFCC-Koeffizient ist ebenfalls ein Repräsentant für die Lautstärke eines Blocks, weswegen dieser oft vor der Klassifizierung verworfen wird. Des Weiteren können bei Bedarf zum Beispiel die Lage der Formanten oder die Grundfrequenz des Sprechers als zusätzliche Merkmale hinzugefügt werden.

Eine übliche Erweiterung sind zudem die sogenannten Delta-Koeffizienten. Ein MFCC-Vektor allein sagt nur etwas über den zugrundeliegenden Audioblock aus, jedoch kann auch dessen zeitliche Änderung zur Spracherkennung beitragen. Deshalb können zusätzlich noch Delta-Koeffizienten als zeitliche Ableitung der MFCC-Werte eingesetzt werden.

Eine gebräuchliche Formel zur Berechnung eines Delta-Vektors ist [6]:

$$d_i[k] = \frac{\sum_{n=1}^{L} n(\, c_{i+n}[k] - \, c_{i-n}[k]\,)}{2 \cdot \sum_{n=1}^{L} n^2} \tag{2.21}$$

$d_i[k]$: k-ter Delta-Koeffizient aus i-tem Block
$c_i[k]$: k-ter MFCC-Koeffizient aus i-tem Block

Der Wert L stellt ein zeitliches Fenster dar, über welchem der entsprechende Delta-Koeffizient berechnet wird. Typische Werte für L liegen zwischen eins und drei [2]. Des Weiteren kann nach der Berechnung der Delta-Werte auch deren Ableitung nach der gleichen Formel gebildet werden. Diese zweite Ableitung der MFCC-Vektoren bezeichnet man dann entsprechend als Delta-Delta-Koeffizienten.

Hinweis: Aus Formel (2.21) geht hervor, dass zur Berechnung eines Delta-Vektors auch zukünftige Audioblöcke ausgewertet werden müssen. Somit müssen die dafür benötigten Koeffizienten für eine Zeit von L Blöcken gespeichert werden, bevor der gesamte Merkmalsvektor berechnet werden kann. In einer Echtzeitanwendung erhält man dadurch eine entsprechende Verzögerung, die sich beispielsweise im Bereich von 10 ms bis 30 ms bewegt.

2.4. Dynamic Time Warp (DTW)

Die MFCC-Merkmalsextraktion liefert für jeden Block Eingangsdaten einen Merkmalsvektor mit einer festen Anzahl von Werten. Die zeitliche Abfolge dieser Merkmalsvektoren liefert eine Vektorfolge, welche für ein eingesprochenes Referenzwort abgespeichert werden kann. Soll nun im Rahmen der Spracherkennung dieses Wort erkannt werden, so lässt sich die gespeicherte mit einer neu aufgenommenen Vektorfolge abgleichen. Im einfachsten Fall wird dabei jeweils die Distanz zweier Merkmalsvektoren mit gleichem Index berechnet (Punkt-zu-Punkt-Abstand) und entlang der zeitlichen Achse aufsummiert. Ist das Ergebnis dann geringer als ein festgelegter Schwellwert, so kann die neue Vektorfolge als das wiedererkannte Referenzwort angenommen werden. Ein übliches Distanzmaß zwischen zwei Merkmalsvektoren x_1 und x_2 ist dabei die euklidische Distanz, wobei die Wurzel für ein Spracherkennungssystem auch weggelassen werden kann:

$$d = \sqrt{\sum_{n=0}^{M-1} (x_1[n] - x_2[n])^2} \tag{2.22}$$

M: Anzahl Merkmale in Merkmalsvektor

Hinweis: Da sich die einzelnen Werte in einem MFCC-Vektor bezüglich ihrer Wichtigkeit und des möglichen Wertebereichs stark unterscheiden, kann die Spracherkennung deutlich verbessert werden, wenn Merkmale mit verschiedenen Indizes auch unterschiedlich gewichtet werden. Die Mahalanobis-Distanz bezieht beispielsweise Streuung und Mittelwert der Merkmale in die Gewichtung mit ein, aber auch eine experimentelle Bestimmung der optimalen Gewichte ist möglich.

Um zwei Vektorfolgen Punkt für Punkt vergleichen zu können, müssen diese gleich lang sein. Ist dies nicht der Fall, muss eine der beiden Folgen zeitlich gestreckt oder gestaucht werden,

bis beide die gleiche Länge haben. Bei der Verwendung einer linearen Transformation wäre die Worterkennung aber sehr schlecht, da verschiedene Teile desselben Wortes unterschiedlich lange andauern können. So hängt zum Beispiel die Dauer von Vokalen stark von der Sprechgeschwindigkeit ab, die Dauer der Konsonanten aber nur unwesentlich. Deshalb muss hier eine nichtlineare Verzerrung erfolgen, um die Merkmale zweier Vektorfolgen zeitlich aufeinander abzustimmen. Dieses Problem kann durch den Algorithmus „Dynamic Time Warp“ (DTW) gelöst werden.

DTW ist eine dynamische Programmiertechnik, die dazu dient, die einzelnen Vektoren zweier Vektorfolgen einander so zuzuordnen, dass die Distanz zwischen den beiden Folgen minimal wird. Dabei werden die Folgen zeitlich nichtlinear verzerrt und die Distanzen der einzelnen Merkmalsvektoren entlang eines optimalen Pfads aufsummiert. Das genaue Vorgehen soll im Folgenden anhand eines Beispiels erläutert werden. Dabei werden zur einfacheren Darstellung zwei Zahlenfolgen anstatt zweier Vektorfolgen und nur ganzzahlige Distanzen verwendet. Abb. 2.13 zeigt links die beiden Zahlenfolgen $x[n]$ und $y[n]$, deren Ähnlichkeit mittels DTW bestimmt werden soll. Man erkennt schon mit bloßem Auge, dass die beiden Folgen einen ähnlichen Verlauf haben, die einzelnen Datenpunkte aber nicht zueinander passen. Zum direkten Vergleich ist rechts schon das Ergebnis nach der Zeitverzerrung dargestellt, welches mit dem DTW-Algorithmus ermittelt werden soll.

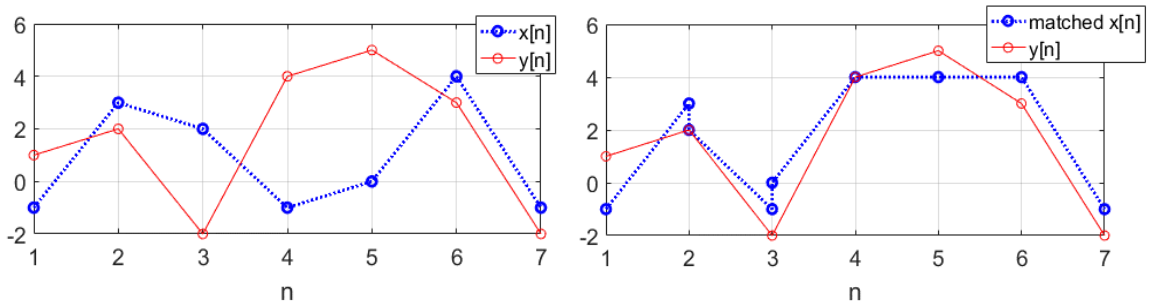


Abb. 2.13 - Beispiel für die Anwendung des DTW-Algorithmus auf zwei Zahlenfolgen (rechts das Ergebnis)

2.4.1. Pfadsuche

Beim klassischen DTW-Algorithmus wird zunächst eine Matrix erstellt, in welche die quadrierten Differenzen zwischen jedem Wert von $x[n]$ und jedem Wert von $y[n]$ geschrieben werden (siehe Abb. 2.14, links).

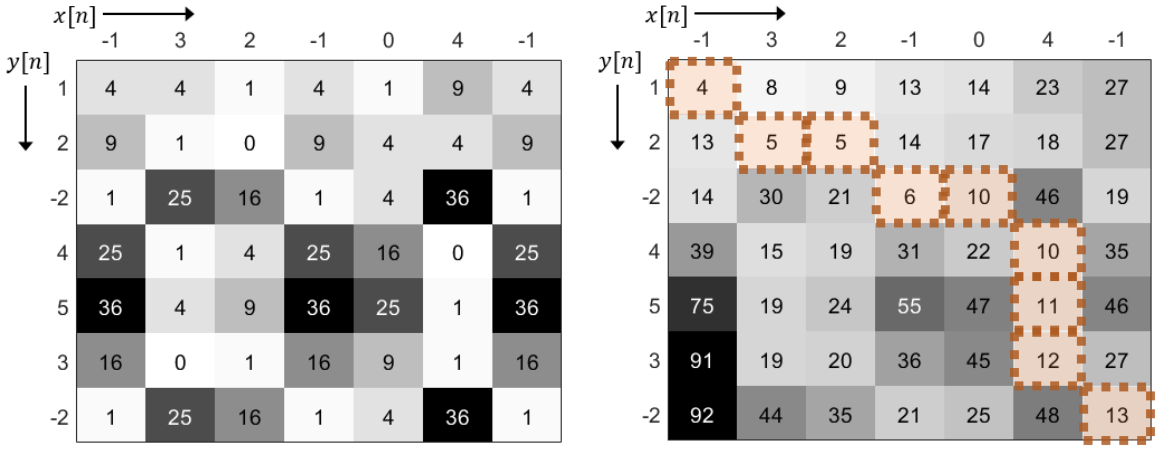


Abb. 2.14 - Beispiel für die Einzeldistanzmatrix (links) und die kumulierte Distanzmatrix (rechts) einer DTW-Pfadsuche

Hieraus wird nun ein Pfad vom Anfang (links oben) zum Ende (rechts unten) gesucht, entlang dessen die aufsummierten Distanzen minimal werden. Die dunkel dargestellten Felder mit hohen Distanzen müssen dabei entsprechend vermieden werden. Anhand der folgenden Formel ergibt sich eine neue Matrix, welche die kumulierten Distanzen beinhaltet:

$$s_{i,k} = d_{i,k} + min\left(s_{i-1,k}\ ,\ s_{i,k-1}\ ,\ s_{i-1,k-1}\right) \tag{2.23}$$

$d_{i,k}$: Distanz zwischen $x[i]$ und $y[k]$

$s_{i,k}$: Kumulierte Distanz entlang des DTW-Pfads

Falls für die Berechnung des Minimums ein benötigtes Feld nicht existiert (linke Spalte oder obere Zeile) wird das Minimum der restlichen Felder verwendet. Das erste Feld der beiden Matrizen ist identisch. Auf diese Weise entsteht der in Abb. 2.14 (rechts) dargestellte Pfad, welcher die beiden Zahlenfolgen optimal für das verwendete Distanzmaß aufeinander anpasst. Der letzte Wert der Matrix (rechts unten) repräsentiert dabei den Abstand der beiden Folgen. Durch das zusätzliche Abspeichern der Rückwärtszeiger bzw. des Minimums-Index aus Formel (2.23) für jedes Feld lässt sich nach der Berechnung der optimale Pfad zurückverfolgen. Das Ergebnis der DTW-Zeitverzerrung für $x[n]$ ist grafisch in Abb. 2.13 (rechts) dargestellt.

2.4.2. Funktionale Anpassungen zur Spracherkennung

Der DTW-Algorithmus lässt sich grundsätzlich für eine Vielzahl von Anwendungen verwenden, bei welchen die Ähnlichkeit zweier nichtlinear verzerrter Sequenzen bestimmt werden soll. Für den Einsatz in einem Spracherkennungssystem bieten sich aber zusätzlich noch einige funktionale Anpassungen an:

- Anstatt zweier Zahlenfolgen werden wie schon beschrieben Vektorfolgen zum Mustervergleich verwendet. Die Funktionsweise des Dynamic Time Warp bleibt identisch, lediglich die Bestimmung der Distanz zwischen zwei Merkmalsvektoren muss angepasst werden.
- Bei der beschriebenen Methode müssen Start- und Endzeitpunkt für beide Vektorfolgen bekannt sein, da die Pfadsuche links oben beginnt und rechts unten endet. Alternativ kann der Startzeitpunkt einer Folge zu Beginn aber auch offenbleiben, indem man nicht nur das erste Feld von der Einzeldistanzmatrix in die kumulierte Distanzmatrix übernimmt, sondern die erste Spalte und die erste Zeile (oder Teile davon). Ebenso kann der Endzeitpunkt einer Folge offenbleiben, indem man nicht das letzte Feld als Gesamtdistanz betrachtet, sondern das Minimum der letzten Spalte und der letzten Zeile (oder von Teilen davon). Diese Anpassung führt in obigem Beispiel zu einem etwas anderen (kürzeren) optimalen Pfad, der in Abb. 2.15 dargestellt ist. Die ersten beiden Werte von $x[n]$ werden hierbei im Ergebnis verworfen. Sollen Anfangs- und/oder Endpunkt beider Folgen variabel sein, so rückt das Start- bzw. Endfeld des Pfades entsprechend noch weiter ins Innere der Matrix.

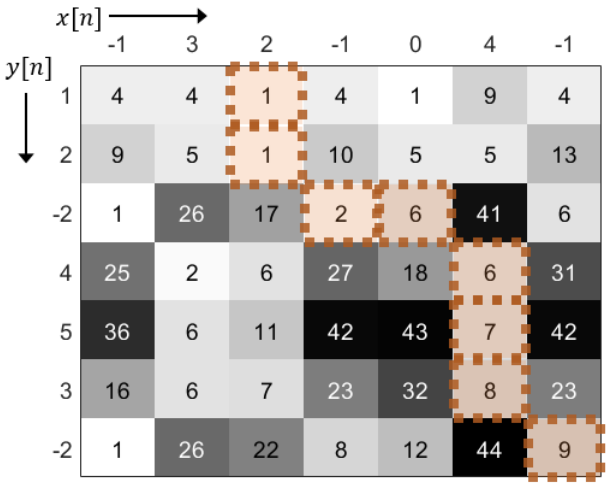


Abb. 2.15 - Beispiel für die kumulierte Distanzmatrix einer DTW-Pfadsuche mit variablem Anfangsfeld

- Bei der Suche des optimalen Pfads führt ein diagonaler Schritt im Mittel zu einer geringeren kumulierten Distanz, als ein horizontaler und ein vertikaler Schritt, auch wenn beide Varianten zum gleichen Feld führen. Außerdem werden bei verändertem Start- oder Endzeitpunkt eventuell weniger Felder aufsummiert. Um dies auszugleichen kann parallel zu jedem $s_{i,k}$ auch noch die Länge des bisher zurückgelegten Pfades in einer weiteren Matrix abgespeichert werden. Die verschiedenen Felder können dann bei der Berechnung des Minimums sowie bei der Bestimmung der Gesamtdistanz durch die Pfadlänge geteilt und damit normalisiert werden.
- Der gewöhnliche Dynamic-Time-Warp-Algorithmus sucht den optimalen Pfad durch die Distanzmatrix. Hierdurch kann es auch passieren, dass ein Pfad gefunden wird, der aufgrund seines Verlaufs nicht erwünscht ist, weil er beispielsweise immer nahe an den Rändern der Matrix entlang verläuft. Um dies zu verhindern, kann die Pfadsuche begrenzt werden, indem nur die Felder innerhalb vorgegebener Bereiche ausgewertet werden. In der Literatur (z.B. [8]) werden hierfür vor allem die in Abb. 2.16 dargestellten Limitierungsbereiche vorgeschlagen. Eine solche Begrenzung verringert gleichzeitig auch die benötigte Rechenzeit, da insgesamt weniger Felder ausgewertet und somit weniger Distanzen berechnet werden müssen.

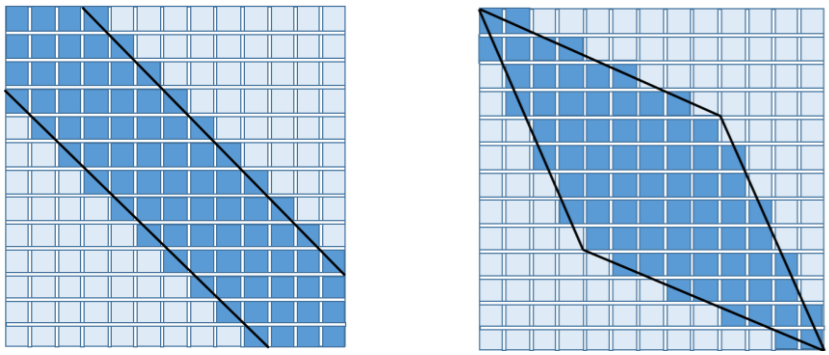


Abb. 2.16 - Beispiele für globale Bereichsbegrenzungen beim DTW-Algorithmus
links: Sakoe-Chiba-Band
rechts: Itakura-Parallelogram

2.5. Künstliche Neuronale Netze (KNN)

Künstliche Neuronale Netze (KNN) stellen eine Implementierungsmöglichkeit für künstliche Intelligenz dar. Sie werden meist dann eingesetzt, wenn klassische Lösungsmethoden versagen, weil ein Problem zu komplex ist oder sich nicht in Algorithmen festhalten lässt. Den Namen verdanken neuronale Netze ihrem Aufbau, welcher an das menschliche Gehirn angelehnt ist: Durch Verknüpfung vieler einzelner Neuronen soll ein hochkomplexes und lernfähiges System entstehen. Während die Vorgänge im Gehirn parallel ablaufen, muss beim KNN die Berechnung aber noch größtenteils seriell erfolgen, es gibt aber mittlerweile auch spezielle Hardware, die für die parallele Berechnung eines KNN ausgelegt ist. Im Folgenden sollen der Aufbau und die grundsätzliche Funktionsweise von neuronalen Netzen näher betrachtet werden.

2.5.1. Aufbau

KNN werden üblicherweise in Schichten aufgebaut. Es gibt dabei immer eine Schicht für Eingangsdaten und eine Schicht für Ausgangsdaten. Dazwischen befinden sich beliebig viele und beliebig große sogenannte „Hidden Layers“, also für die spätere Anwendung versteckte Schichten. Abb. 2.17 zeigt beispielhaft ein KNN mit insgesamt vier Schichten.

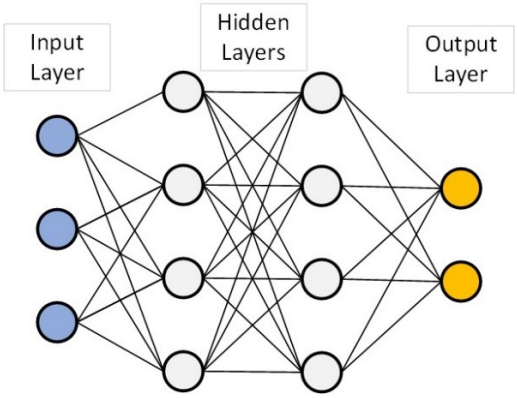


Abb. 2.17 - Schematischer Aufbau eines künstlichen neuronalen Netzes mit vier Schichten

Jede Schicht ist aus Neuronen aufgebaut. Man kann grob sagen, dass die Funktionalität, die das Netz abbilden kann mit der Anzahl der Neuronen in den versteckten Schichten ansteigt. Wenn wie hier alle Neuronen einer Schicht mit allen Neuronen der vorherigen Schicht verbunden sind, spricht man von einem „fully connected layer“, es können grundsätzlich aber auch weniger Verbindungen vorhanden sein.

Jedes Neuron in einem KNN stellt eine eigene kleine Berechnungseinheit dar, welche wie in Abb. 2.18 gezeigt modelliert werden kann.

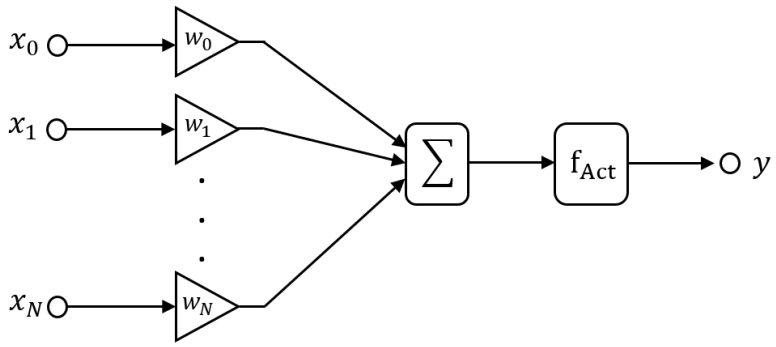


Abb. 2.18 - Schematische Darstellung eines vereinfachten Neuronenmodells

Die Eingänge des Neurons x_0 bis x_N stellen hierbei die Verbindungen zur vorherigen Schicht dar und werden in jedem Berechnungsschritt mit einem individuellen Gewichtungsfaktor multipliziert. Die Ergebnisse der Multiplikation werden dann über alle Eingänge aufsummiert (in Spezialfällen ist hier auch eine andere Operation wie z.B. eine Multiplikation möglich) und einer Aktivierungsfunktion zugeführt. Diese bildet den Wert der Summe auf einen neuen Wert ab. Die Berechnung des Ausgangs kann somit auch wie folgt mathematisch beschrieben werden:

$$y = f_{Act}\left(\sum_{n=0}^{N} x_n \cdot w_n\right) \tag{2.24}$$

Zusätzlich zu den Gewichten kann der Eingangswert der Aktivierungsfunktion auch durch einen Offset-Wert beeinflusst werden. Dieser kann entweder im Neuron selbst modelliert werden (durch Erweiterung des vereinfachten Modells aus Abb. 2.18) oder durch das Einfügen von sogenannten Bias-Neuronen [9]. Ein solches Bias-Neuron ist exemplarisch in Abb. 2.19 dargestellt. Es besitzt keine Eingänge und liefert an seinem Ausgang somit einen konstanten Wert, der zusammen mit der Verbindungsgewichtung den Offset für das damit verbundene Neuron darstellt.

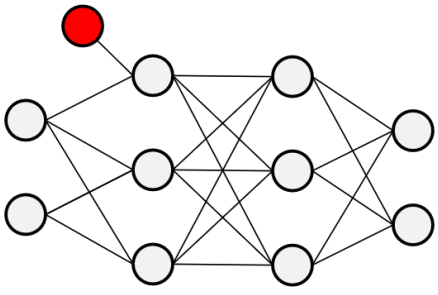


Abb. 2.19 - Schematischer Aufbau eines künstlichen neuronalen Netzes mit einem Bias-Neuron

Der Ausgang des Neurons wird dann entsprechend wieder an die Eingänge der Neuronen in der folgenden Schicht weitergeleitet. Auf diese Weise entstehen bei sehr hoher Neuronenzahl auch sehr komplexe Strukturen, deren exakte Funktionsweise oft nicht mehr vom Menschen nachvollzogen werden kann.

Die Wahl der Aktivierungsfunktion beeinflusst maßgeblich die benötigte Größe bzw. die erreichbare Funktionalität des Netzes sowie die Ausführungszeit für einen Rechenschritt. Die Diagramme in Abb. 2.20 bis Abb. 2.23 zeigen die folgenden typischen Aktivierungsfunktionen:

- **Identische Abbildung**
  Die Aktivierung (bzw. der Ausgang) ist identisch mit der Summe der gewichteten Eingänge

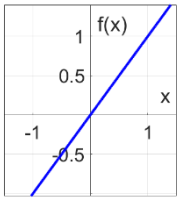


Abb. 2.20 - Identische Abbildungsfunktion

- **Rectified Linear Unit (ReLU)**
  Erweiterung der identischen Abbildung um einen Schwellwert, wodurch negative Ausgangswerte auf null gesetzt werden

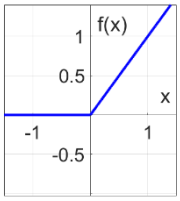


Abb. 2.21 - ReLU-Funktion

- **Sprungfunktion**
  Die Aktivierung ist null für negative Eingangswerte und eins für positive Eingangswerte

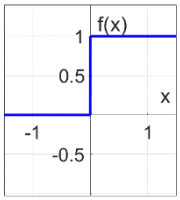


Abb. 2.22 - Sprungfunktion

- **Tangens Hyperbolicus**
  Diese Aktivierungsfunktion stellt einen nichtlinearen Zusammenhang her, der im Vergleich zur ReLU- und der Sprungfunktion überall eindeutig differenzierbar ist

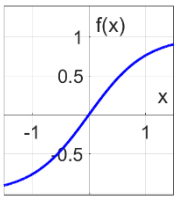


Abb. 2.23 - Tangens Hyperbolicus

2.5.2. Training

Damit ein KNN in einer Anwendung genutzt werden kann, muss dieses zunächst trainiert werden. Hierfür werden meist sehr viele und gut aufbereitete Trainings- und Testdaten benötigt. Aus diesen soll das Netz dann die Zusammenhänge zwischen Eingangs- und Ausgangsdaten möglichst selbständig erlernen. Je nach Anwendung werden grundsätzlich drei Lernarten unterschieden [9]:

- **Supervised Learning** (überwachtes Lernen):
  Für die zu trainierenden Eingangsdaten ist bekannt, was am Ausgang zu erwarten ist. Dieser Wert wird dann für die Korrektur der internen Parameter an das Netz weitergeleitet.
- **Reinforcement Learning** (verstärkendes Lernen)
  Dem Netz wird hier kein exakter Wert für den korrekten Ausgang bezüglich angelegter Eingangsdaten mitgeteilt, sondern nur ob die Ausgabe des Netzes richtig oder falsch ist.
- **Unsupervised Learning** (nicht überwachtes Lernen):
  Hier wird keine Information zu einem korrekten Ausgang vorgegeben. Stattdessen werden Ähnlichkeiten oder Muster in den Eingangsdaten komplett selbständig erlernt.

Während des Trainings werden bei den überwachten Lernmethoden folgende Schritte durchlaufen:

1) Initialisierung der internen Parameter mit zufälligen Werten.
   Diese Parameter sind im Wesentlichen die Gewichtungsfaktoren aus Abb. 2.18. Zusätzlich können auch noch verschiedene Schwell- und Offsetwerte eingefügt und trainiert werden.
2) Anlegen von Eingangsdaten, von welchen man weiß, was am Ausgang zu erwarten ist.
3) Berechnung der Ausgangsdaten für die angelegten Eingangsdaten.
4) Korrektur der internen Parameter, sodass sich die Ausgangswerte den korrekten Werten annähern.
5) Wiederholung der Schritte 2-4 mit möglichst großem Datensatz (damit nicht immer auf die gleichen Eingangsdaten trainiert wird), bis das KNN die gewünschte Genauigkeit liefert.

Zur Prüfung, ob das Netz die gewünschte Genauigkeit liefert, sollte in der Regel ein anderer Datensatz verwendet werden, als für das eigentliche Training. Man unterscheidet also Trainings- und Test- bzw. Validierungsdaten. Würde man dies nicht tun, könnte es passieren,

dass das neuronale Netz die Trainingsdaten nur auswendig lernt, ohne die wirklichen Zusammenhänge zwischen Ein- und Ausgangsdaten zu erkennen.

Die Art und Weise der Parameteranpassung während des Lernens wird durch den verwendeten Trainingsalgorithmus bestimmt. Hier gibt es viele verschiedene Ansätze. Die einfachste Variante ist dabei sicherlich Trial-and-Error, also das zufällige Verändern eines Parameters mit anschließender Überprüfung, ob sich der Ausgang verbessert hat. Allerdings sind meist effizientere Verfahren zu bevorzugen, welche in deutlich kürzerer Zeit ein brauchbares Ergebnis liefern. Im Folgenden sind grob die Schritte des Backpropagation-Algorithmus vorgestellt [9], welcher ein weit verbreitetes Lernverfahren darstellt:

- Anlegen von Eingangsdaten und Berechnung der entsprechenden Ausgänge
- Berechnung des Fehlers zwischen den Ausgängen und den korrekten Werten mit einem geeigneten Distanzmaß, z.B. quadratischer Abstand
- Der Fehlerterm wird in entgegengesetzter Richtung zurück zu den Eingängen propagiert. Dabei werden die einzelnen Gewichte mithilfe des Gradientenabstiegsverfahrens so angepasst, dass sich die Fehlerfunktion einem lokalen Minimum annähert. Die Gewichtsänderung bestimmt sich dabei wie folgt [10] und ist zum vorherigen Gewicht hinzuzuaddieren:

$$\Delta w = -\eta \cdot \frac{\partial E}{\partial w} \tag{2.25}$$

$\frac{\partial E}{\partial w}$: Partielle Ableitung des Fehlers nach dem zu ändernden Gewicht

η: Lernrate

In Kapitel 3.4.2 wird mit Resilient Backpropagation (RPROP) eine etwas abgewandelte Form davon als Trainingsalgorithmus verwendet. Während die Gewichtsänderung bei der gewöhnlichen Fehlerrückführung vom Wert der partiellen Ableitung abhängt, so hängt sie bei RPROP nur von dessen Vorzeichen ab. Die Schrittweite der Änderung wird dabei vergrößert, wenn das Vorzeichen des Gradienten im Vergleich zur letzten Berechnung gleichbleibt und man sich somit in die richtige Richtung bewegt. Abb. 2.24 verdeutlicht den Unterschied der beiden Varianten für den Fall, dass es nur ein Gewicht gibt: Beim gewöhnlichen Backpropagation-Algorithmus wird die Schrittweite immer kleiner, da auch die Kurve flacher wird. Beim RPROP-Algorithmus wird die Schrittweite nach den ersten beiden Sprüngen vergrößert, weil man sich in die richtige Richtung bewegt. Danach wird die Schrittweite verkleinert, da sich das Vorzeichen der Steigung umgekehrt hat. Bei beiden Varianten gibt es

keine Garantie, dass es sich bei dem gefundenen Minimum um ein globales Minimum handelt, außer der Fehler wird wirklich zu Null.

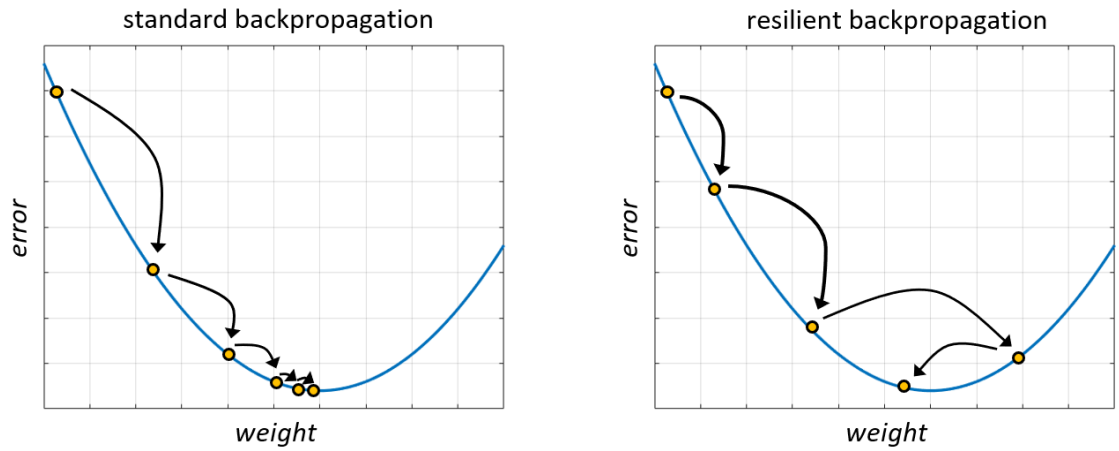


Abb. 2.24 - Schematischer Vergleich des gewöhnlichen Backpropagation-Algorithmus mit resilient Backpropagation

2.5.3. Netztopologien

Je nach seinem Aufbau lässt sich ein neuronales Netz einem oder mehreren der in Tabelle 2.1 enthaltenen Netztopologien zuordnen.

Typ	*Beschreibung*
Feed Forward Neural Network	Alle Verbindungen verlaufen von den Eingängen in Richtung der Ausgänge. Es gibt keine Rückkopplungen.
Recurrent Neural Network	Es gibt mindestens eine Rückkopplung im Netzaufbau
Convolutional Neural Network (CNN)	Es gibt mindestens eine faltende Schicht

Tabelle 2.1 - Typische Topologien von neuronalen Netzen

Hinweis: Tabelle 2.1 stellt lediglich die grundlegendsten bzw. für diese Arbeit relevanten Netztopologien dar. Es existieren zahlreiche weitere Arten von neuronalen Netzen.

Convolutional Neural Network (CNN)

Eine beliebte Netztopologie bilden sogenannte faltende neuronale Netze (Convolutional Neural Network, CNN). Diese werden klassischerweise bei der Objekterkennung eingesetzt, weil sie sehr effizient Muster in Bildern erkennen können. Wenn man es schafft, die Eingangsdaten (also die Merkmale) in grafischer Form darzustellen, lässt sich diese Netztopologie aber grundsätzlich auch in der Spracherkennung verwenden. Eine übliche Form

für die bildliche Darstellung des Frequenzspektrums sind Spektrogramme. Hier werden die in den Frequenzbändern enthaltenen Signalenergien anhand verschiedener Farben über der Zeit aufgetragen. Es ist aber auch möglich die in Kapitel 2.3 beschriebenen MFCC-Merkmalsvektoren auf diese Weise über die Zeit aufzutragen, um sie als Bild zu interpretieren. Abb. 2.25 zeigt beispielhaft eine solche grafische Interpretation. Auf der linken Seite sind die Spektrogramme für zwei Aufnahmen des Wortes „eight“ desselben Sprechers zu sehen. Auf der rechten Seite befinden sich die entsprechend daraus extrahierten MFCC-Merkmale[1]. Man sieht, dass sowohl in den Spektrogrammen als auch in den MFCC-Bildern Muster zu erkennen sind, auch wenn bei Letzteren keine Frequenzen mehr dargestellt werden und deutlich weniger Datenpunkte vorliegen. Diese Muster können entsprechend mit einem CNN erkannt werden.

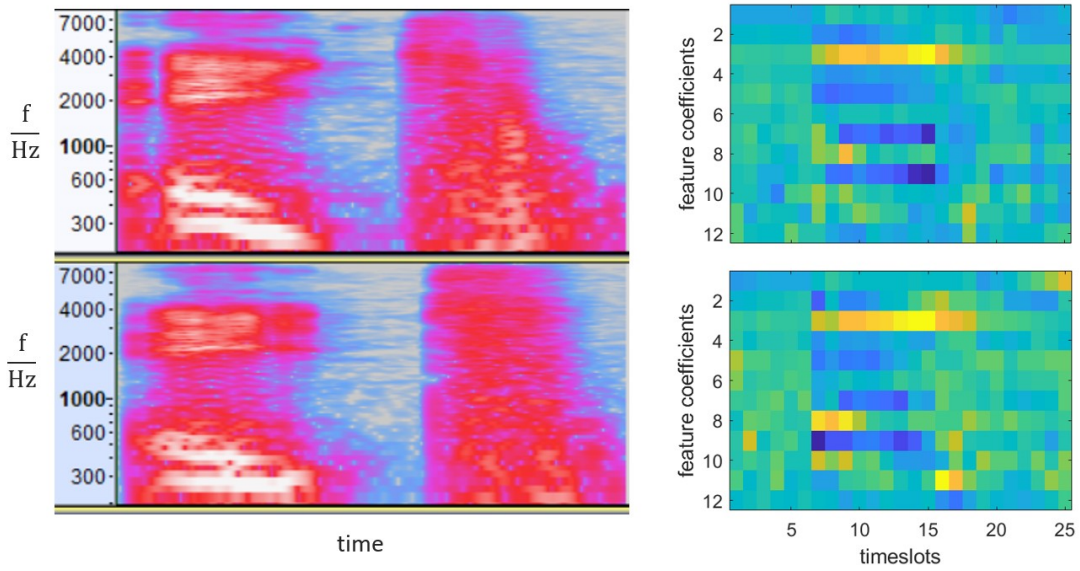


Abb. 2.25 - Spektrogramme (links) und MFCC-Sequenzen (rechts) für zwei Instanzen des Wortes "eight" desselben Sprechers

Hinweis: Die Verwendung von grafischen Darstellungen eines Sprachsignals als Eingangswerte für ein CNN ist nicht unumstritten. In der Literatur finden sich auf der einen Seite zahlreiche erfolgreiche Implementierungen zur Spracherkennung (z.B. [11]), auf der anderen Seite wird deren Effektivität im Vergleich zu gewöhnlichen Objekterkennern (bei realen Bildern) aber auch angezweifelt [12]. Da das beschriebene Verfahren aber grundsätzlich funktioniert, wird es auch in dieser Arbeit eingesetzt (siehe Kapitel 3.4.2).

[1] Ohne zeitliche Ableitungen und der erste Koeffizient wurde verworfen

Als CNN wird ein Netz bezeichnet, welches mindestens einen Convolutional Layer enthält. Dabei werden verschiedene Filter auf die Eingangsmatrix angewendet, um bestimmte Muster darin feststellen zu können. Die Art und Weise der Operation entspricht einer zweidimensionalen Faltung, woraus sich auch der Name der Schicht und der Netztopologie ableiten lässt. Das Ergebnis ist eine sogenannte Feature Map, welche als neues Bild den Eingang für nachfolgende Schichten bilden kann. Das Filter selbst kann ebenfalls als Bild interpretiert werden und sein entsprechendes Muster soll entsprechend in der Eingangsmatrix wiedergefunden werden. Dabei steht ein hoher Wert im Ausgangsbild (in Abb. 2.26 die hellen Felder) für eine hohe Übereinstimmung, ein Wert nahe Null deutet hingegen auf eine sehr geringe Ähnlichkeit mit dem Filtermuster hin. Für die Berechnung des Ausgangsbildes (Feature Map) werden folgende Schritte durchlaufen:

1) Das Filter hat eine feste Pixelgröße (hier 3x3) und wird elementweise mit einem gleichgroßen Ausschnitt des Eingangsbildes multipliziert
2) Die Produkte werden aufsummiert
3) *(optional)* Die Summe wird durch die Anzahl der Felder (hier neun) geteilt, damit der Ausgangswert nicht größer als eins wird
4) *(optional)* Negative Werte werden auf null gesetzt
5) Der Wert wird in die Feature Map geschrieben. Das Feld ergibt sich dabei aus der Position des verwendeten Ausschnitts im Eingangsbild
6) Der Ausschnitt wird um eine feste Schrittweite (hier 1) weiterverschoben und die vorherigen Schritte wiederholt, bis alle Felder der Ausgangsmatrix beschrieben sind

Abb. 2.26 zeigt das Vorgehen beispielhaft für zwei verschiedene Filter auf dem gleichen Eingangsbild. Die beiden als optional gekennzeichneten Schritte wurden hierbei berücksichtigt. Als Resultat ist in den Feature Maps zu erkennen, dass das Eingangsbild oben zentriert eine diagonale Linie und unten links eine vertikale Linie enthält.

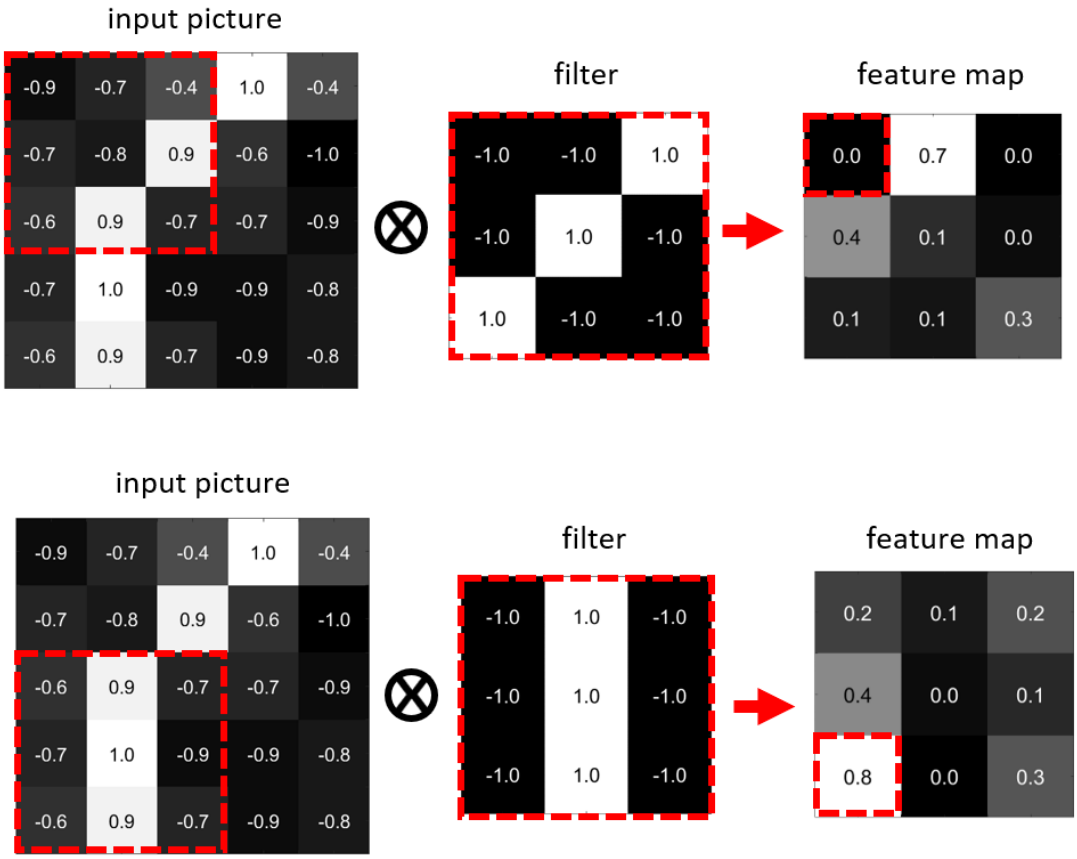


Abb. 2.26 - Beispiel einer zweidimensionalen Faltung mit zwei verschiedenen Filtern

Hinweis: Die Felder des Eingangsbildes und der Feature Maps entsprechen im neuronalen Netz jeweils einem Neuron. Im Gegensatz dazu werden die Felder des Filters als Gewichtungen zwischen einem Neuron der Feature Map und den entsprechenden Neuronen des Eingangsbildes realisiert. Somit müssen die Filtermuster nicht vorher festgelegt sein, sondern können automatisch trainiert werden.

Im Anschluss an ein Convolutional Layer folgt üblicherweise ein sogenannter Pooling Layer, welcher die Aufgabe hat, die Datenmenge des Ausgangsbildes zu verkleinern. Das Verfahren wird auch Subsampling genannt. Besonders weit verbreitet ist dabei das Max-Pooling, bei welchem nur der größte Wert aus mehreren Pixeln weiterverwendet wird. Aber auch andere Operationen wie eine Mittelwertbildung sind möglich. Abb. 2.27 zeigt das Subsampling mit der zuvor erhaltenen Feature Map, einer Größe von 2x2 und einer Schrittweite von Eins.

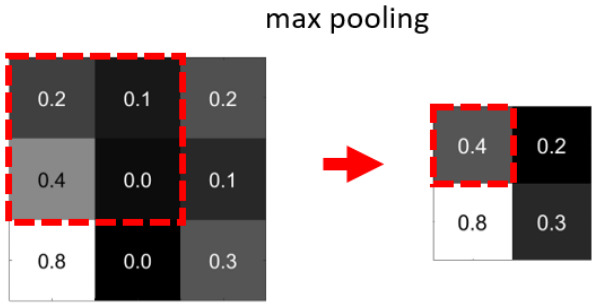


Abb. 2.27 - Beispiel eines Max-Pooling-Filters

Recurrent Neural Network (RNN)

Beim zuvor betrachteten CNN enthalten die Eingangsdaten eine zeitliche Abfolge der Sprachinformationen, wodurch sehr viele Eingangsdaten nötig sind. Einen etwas anderen Ansatz erreicht man durch den Einsatz eines rekurrenten Netzes. Hier sind in den versteckten Schichten Rückkopplungen enthalten, wodurch das Netz selbst intern Informationen speichern kann. Somit reicht es aus, als Eingangsdaten die extrahierten Merkmale eines Datenblocks anzulegen, den zeitlichen Kontext zu anderen Blöcken erlernt das Netz dann automatisch. Grundsätzlich kann eine Rückkopplung innerhalb einer Schicht oder aber auch über mehrere Schichten hinweg erfolgen.

2.6. Message Queuing Telemetry Transport (MQTT)

MQTT ist ein Kommunikationsprotokoll zur Nachrichtenübertragung zwischen verschiedenen Geräten. Es ist ein Client-Server-Modell, wobei es in einem System einen Server (im folgenden Broker genannt) und beliebig viele Clients geben kann. Der Broker übernimmt nur die Aufgabe, Daten zu empfangen und auszuliefern, er sendet keine eigenen Daten. Wie in Abb. 2.28 dargestellt implementiert MQTT das Observer-Pattern, so kann ein Client entweder Nachrichten unter einem bestimmten Thema veröffentlichen (publish), oder ein bestimmtes Thema abonnieren (subscribe). Informationen über die Herkunft einer Nachricht werden in der Regel nicht übermittelt. So müssen die verschiedenen Clients nicht einmal etwas voneinander wissen, da das gesamte Protokoll nachrichtenbasiert ist. Die MQTT-Themen sind Zeichenketten, die ähnlich einer Ordnerstruktur aufgebaut sind, wodurch man auch mehrere Themen als Wildcard abonnieren kann. Der grobe Ablauf der Kommunikation ist in Abb. 2.28 dargestellt. Clients können dabei beliebige Geräte sein, solange sie eine IP-Adresse haben und sich mit dem Broker verbinden können.

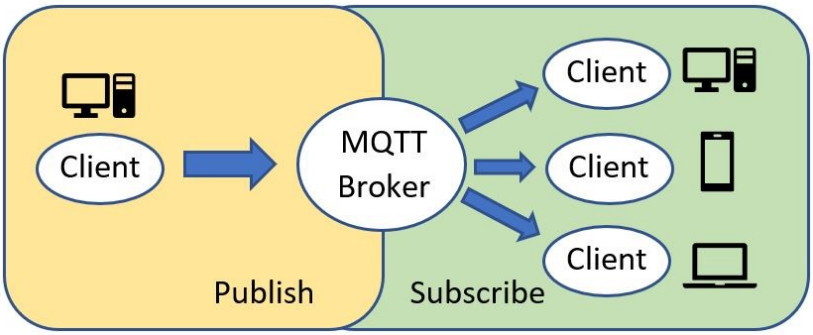


Abb. 2.28 - Publish/Subscribe-Architektur bei MQTT

Quality of Service (QoS)

MQTT bietet drei verschiedene QoS-Level für eine entweder effiziente oder zuverlässige Nachrichtenübertragung:

QoS 0: Nachricht wird maximal einmal übertragen

QoS 1: Nachricht wird mindestens einmal übertragen

QoS 2: Nachricht wird exakt einmal übertragen

MQTT nutzt als darunterliegendes Protokoll meist TCP, welches ebenfalls eine zuverlässige Datenübertragung garantiert. Trotzdem kann es aus zwei Gründen sinnvoll sein, den QoS höher als null anzusetzen [13]:

- TCP garantiert eine korrekte Datenübertragung nur vom Ende eines TCP/IP-Stacks zum Ende eines anderen. Dadurch ist noch nicht zwangsläufig garantiert, dass die Anwendung diese Daten auch richtig weiterverwendet.
- Falls die TCP-Verbindung aus beliebigen Gründen unterbrochen wird, können trotzdem Daten verloren gehen. Um dies zu vermeiden, kann MQTT diese mit einem höheren QoS nach einer bestimmten Zeit erneut senden.

3. Implementierung eines Keyword-Spotting-Systems

In diesem Kapitel wird die Entwicklung und Implementierung des Keyword-Spotting-Systems auf dem UDOO Neo Board und damit der praktische Teil dieser Abschlussarbeit beschrieben. Die Reihenfolge der beschriebenen Komponenten entspricht dabei grob den in den Kapiteln 2.2 bis 2.5 behandelten Themen und wird durch das Schaubild in Abb. 3.1 verdeutlicht. Dieses zeigt den Gesamtaufbau des zu implementierenden Keyword-Spotting-Systems vom Mikrofon bis zur MQTT-Schnittstelle.

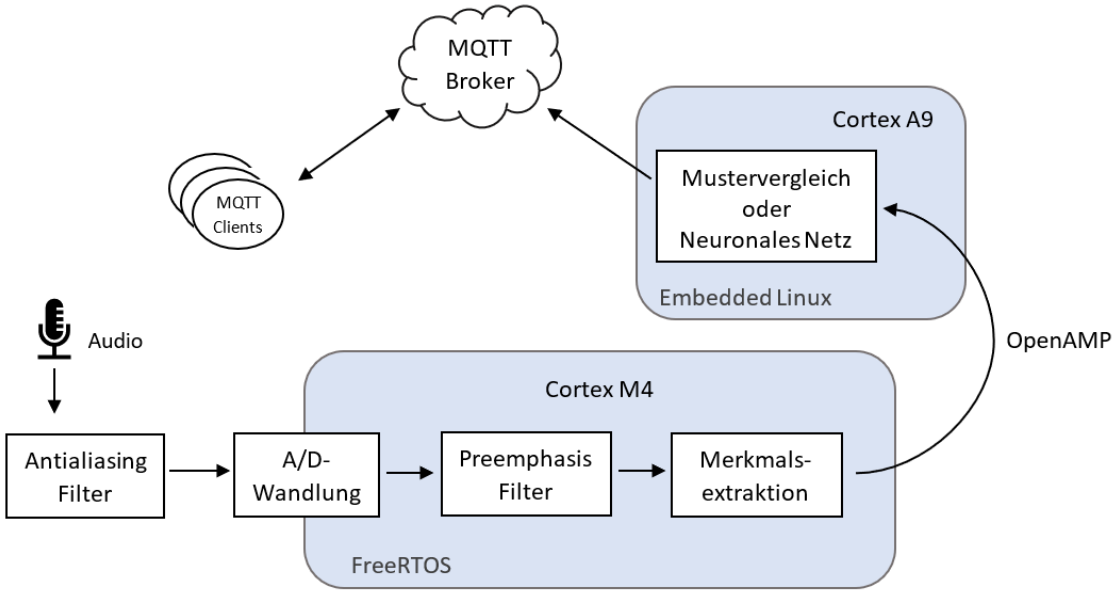


Abb. 3.1 - Gesamtaufbau des Spracherkennungssystems

Die Komponenten Antialiasing-Filter, A/D-Wandlung und das Preemphasis-Filter bilden die Signalvorverarbeitung, um das Sprachsignal für die Merkmalsextraktion aufzubereiten. Die Merkmale werden dann über die OpenAMP- bzw. RPMsg-Schnittstelle an den Cortex-A9 übertragen, auf welchem die eigentliche Spracherkennung stattfindet. Diese wird einmal mit einem Mustervergleich (DTW) zur sprecherabhängigen Erkennung und einmal mit einem neuronalen Netz zur sprecherunabhängigen Erkennung realisiert. Bei beiden Varianten wird bewusst auf eine Detektion des Start- und Endzeitpunkts der Schlüsselworte verzichtet. Stattdessen werden die Sprachinformationen laufend und mit kleinen Zeitverschiebungen zwischen den Blöcken verarbeitet. Um die Echtzeitfähigkeit des Systems zu gewährleisten, ist es deshalb wichtig, dass die Verarbeitung der Audiodaten sehr schnell erfolgt. Dies muss bei der Entwicklung des Systems entsprechend berücksichtigt werden. Die erkannten Wörter werden dann an einen MQTT-Broker gesendet, bei dem sie entsprechend weiterverarbeitet werden können.

3.1. CMSIS-DSP-Bibliothek

Für Anwendungen, die große Datenmengen in kurzer Zeit verarbeiten müssen, werden klassischerweise digitale Signalprozessoren (DSP) verwendet. Trotzdem kann eine Signalverarbeitung auch mit einem Prozessor der Cortex-M-Serie in gewissem Umfang durchgeführt werden. Deren Lizenzgeber ARM stellt zu diesem Zweck eine hochoptimierte Bibliothek namens „CMSIS DSP Library“ zur Verfügung. In diesem Kapitel soll nun die Performance dieser Bibliothek für die Verwendung zur Merkmalsextraktion auf einem Cortex-M4-Microcontroller untersucht werden. Die zugrundeliegenden Berechnungen werden im 32 Bit-Gleitkommaformat durchgeführt, weil die verwendete MCU eine Floating Point Unit (FPU) besitzt.

Um die DSP-Bibliothek verwenden zu können, muss die entsprechende Headerdatei eingebunden und das Zielsystem in Form einer Präprozessordirektive gesetzt werden (z.B. „ARM_MATH_CM4“). Die Bibliothek kann entweder in vorkompilierter Form oder durch die frei verfügbaren Quelldateien verwendet werden.

Die meisten Funktionen der DSP-Bibliothek verwenden als Eingabe- und Rückgabewerte Zeiger auf Vektoren bzw. Arrays. Zudem können auch komplexe Zahlen verarbeitet werden, indem der Realteil einer komplexen Zahl immer in den Arrayfeldern mit geradem Index und der Imaginärteil in den Feldern mit ungeradem Index gespeichert wird. Auf diese Weise werden verschiedene Funktionen für Berechnungen mit einem oder auch mehreren Vektoren zur Verfügung gestellt, wie beispielsweise:

- Elementweise arithmetische Operationen und Betragbildung
- Digitale Filter
- Standardabweichung und Varianz
- Faltung und Korrelation
- Diskrete Fourier- und Kosinustransformationen

Die Funktionen der Bibliothek sind stark auf das entsprechende Zielsystem optimiert, nutzen aber grundsätzlich ähnliche Optimierungsmethoden zur Verarbeitung von Vektoren. Bei den meisten Funktionen wird das sogenannte „Loop Unrolling“ angewendet. Bei vielen Schleifen entsteht das Problem, dass die Anzahl der Anweisungen (auf Assemblerebene) im Schleifenrumpf gering oder ähnlich groß ist wie die der Kontrollanweisungen, die für die Schleife selbst benötigt werden. Es wird also ein großer Overhead zum Überprüfen der Schleifenbedingung und Verändern von Schleifenvariablen generiert. Beim Loop Unrolling werden nun die Operationen im Schleifenrumpf innerhalb eines Schleifendurchlaufs mehrfach ausgeführt und die Schleife dafür entsprechend weniger oft durchlaufen. Dadurch kann der

Overhead stark reduziert werden. Zu beachten ist aber, dass die Codegröße entsprechend ansteigt, was aber auf modernen Systemen meist kein Problem darstellt.

3.1.1. Optimierung der Kopierfunktion

Im Folgenden wird das Ausmaß und der Nutzen der Optimierung beispielhaft für die Funktion „memcpy()“ dargestellt. Diese Standard-C-Funktion kopiert einen Speicherbereich an eine neue Adresse und steht auf nahezu allen C-Plattformen zur Verfügung. Allerdings ist die genaue Implementierung compilerabhängig. Es ist also bei der Verwendung nicht festgelegt, wie sie in Maschinencode umgesetzt wird.

Auch die DSP-Bibliothek bietet Kopierfunktionen für Gleit- und Festkommavariablen. Die verschiedenen Funktionen unterscheiden sich im Wesentlichen darin, ob eine FPU oder auch SIMD-Anweisungen verwendet werden. Der Rechenaufwand der folgenden Kopierfunktionen soll nachfolgend verglichen werden:

- void *memcpy(void* pDst, const void* pSrc, size_t n)
  Wie schon erwähnt ist die Umsetzung hier compilerabhängig. Deshalb sollte das Ergebnis nur als grober Richtwert verstanden werden. In diesem Test wird der gnu11-Compiler verwendet. Verschiedene Optimierungslevel haben keinen Laufzeitunterschied bewirkt.
- void arm_copy_q7(q7_t * pSrc, q7_t * pDst, uint32_t blockSize)
  Die Funktion verwendet folgende Codezeile in einem Schleifenrumpf, um Speicher zu kopieren:
  ```
  *__SIMD32(pDst)++ = *__SIMD32(pSrc)++;
  ```
  Die „Single Instruction Multiple Data“-Anweisung (SIMD) bewirkt eine parallele Abarbeitung von vier 8-Bit-Variablen gleichzeitig, um Rechenzeit zu sparen.
- void arm_copy_f32(float32_t * pSrc, float32_t * pDst, uint32_t blockSize)
  Diese Funktion ist in Bezug auf die Funktionsweise und der Performance identisch mit der Funktion „arm_copy_q31()“, auch wenn sie dem Namen nach für verschiedene Datentypen gedacht sind. Der eigentliche Datentyp kann im Funktionsaufruf durch einen expliziten Cast ohnehin irrelevant gemacht werden. In der Funktion werden durch Loop Unrolling vier 32-Bit-Variablen innerhalb eines Schleifendurchlaufs kopiert, wodurch der Schleifen-Overhead erheblich reduziert wird.

- Die Funktion „arm_copy_f32()“ kann zudem durch eine manuelle Anpassung noch weiter optimiert werden, indem man noch mehr Schleifendurchläufe zusammenfasst. In diesem Fall liegt das Optimum beim Kopieren von 16 Variablen pro Schleifendurchlauf. Allerdings sollte solch eine modifizierte Version nur in Ausnahmefällen genutzt werden, da die Optimierung compiler- und systemabhängig ist. Die für diesen Test modifizierte Funktion befindet sich in Anhang A.1.

Zum Vergleich der vier Methoden wird nun die Anzahl der benötigten Zyklen zum Kopieren verschiedener Datenmengen ermittelt. Das Ergebnis ist in Abb. 3.2 dargestellt und zeigt, dass es möglich ist, über 90% Rechenzeit im Vergleich zu memcpy zu sparen.

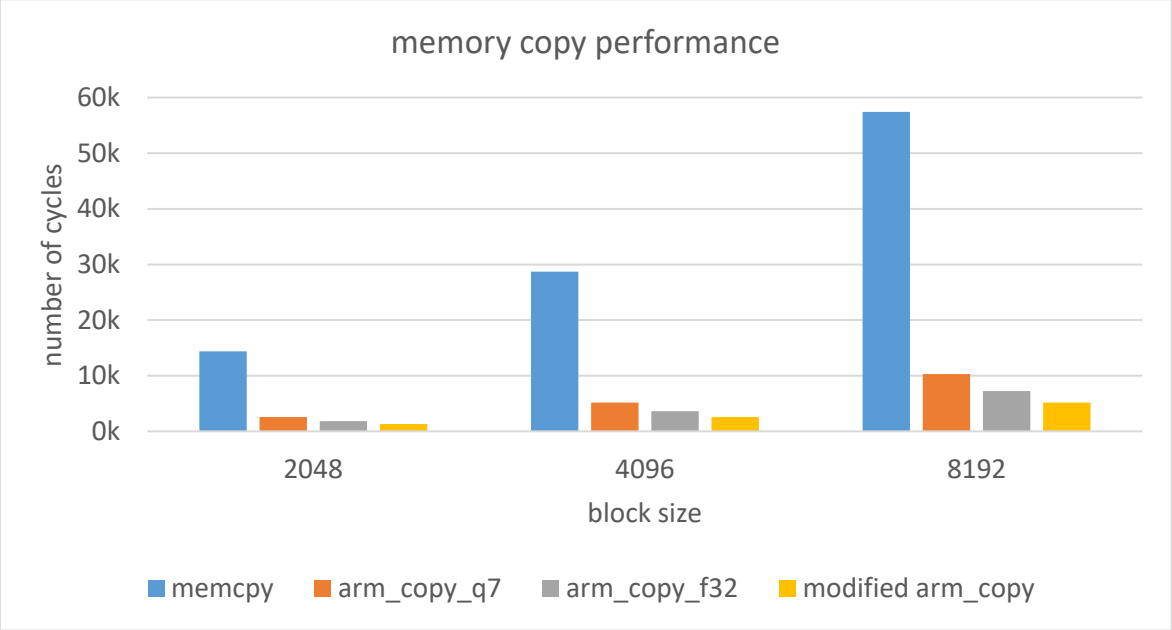


Abb. 3.2 - Performance-Vergleich verschiedener Kopierfunktionen

Hinweis: Bei den beiden schnellsten Kopiervarianten werden die zu kopierenden Variablen zunächst alle in lokale Variablen geladen, bevor sie dann hintereinander abgespeichert werden. Dies verhindert sogenannte „Stall“-Zustände (verschwendeten Zyklen) in der Pipeline. Würde man die Variablen nach dem Laden immer direkt speichern, wäre die Reihenfolge der LDR- und STR-Assemblerbefehle im Maschinencode abhängig vom Optimierungslevel des Compilers, was nicht immer zur besten Performance führt. Bei der Verwendung von zu vielen lokalen Variablen werden je nach Compiler auch die Register der FPU verwendet, was aber auch zu einer Erhöhung der Interrupt-Latenz führen kann [14].

3.1.2. Optimierung der Diskreten Fouriertransformation

Wie in Kapitel 2.3.3 erläutert, ist für die Merkmalsextraktion lediglich die Berechnung des reellen Leistungsspektrums nötig. Im Folgenden wird nun die Laufzeit von drei möglichen Berechnungen des Leistungsspektrums verglichen:

- Berechnung gemäß den Formeln (2.12) und (2.13), wobei Real- und Imaginärteil als eigenständige Arrays abgebildet werden. Um nicht die langsamen trigonometrischen Funktionen aus „math.h“ verwenden zu müssen, wird zudem auf die bereits vorhandenen Lookup-Tabellen der DSP-Bibliothek zurückgegriffen.
- Berechnung über einen Radix-2-FFT-Algorithmus, welcher in Anhang A.2 zu finden ist. Da es sich um eine Fast Fourier Transformation (FFT) handelt, muss die Blocklänge eine Zweierpotenz sein. Die Berechnung des Leistungsspektrums erfolgt dann über eine Funktion der DSP-Bibliothek.
- Berechnung mithilfe der DSP-Bibliothek. Diese implementiert intern ebenfalls einen Fast Fourier Algorithmus. Die Berechnung des Leistungsspektrums erfolgt dabei über vier Funktionsaufrufe:
    - Kopieren der Eingangsdaten mit „arm_copy_f32()“
    - Initialisierung mit „arm_rfft_fast_init_f32()“
    - Komplexe DFT für reelle Eingangsdaten mit „arm_rfft_fast_f32()“
    - Betragsbildung und Quadrieren mit „arm_cmplx_mag_squared_f32()“

Das Kopieren der Eingangsdaten kann je nach Anwendung nötig sein, da intern die Funktion „arm_cfft_f32()“ verwendet wird und diese die Eingangsdaten verändert.

Für den Test wird die benötigte Ausführungszeit gemessen, über mehrere Messungen gemittelt und anschließend in Prozessorzyklen umgerechnet. Das Testergebnis ist in Abb. 3.3 für verschiedene Blocklängen dargestellt. Die logarithmische Achsenskalierung zeigt dabei deutlich, dass die optimierte Bibliothek einer Standardimplementierung in Bezug auf die Ausführungszeit um mehrere Zehnerpotenzen überlegen ist.

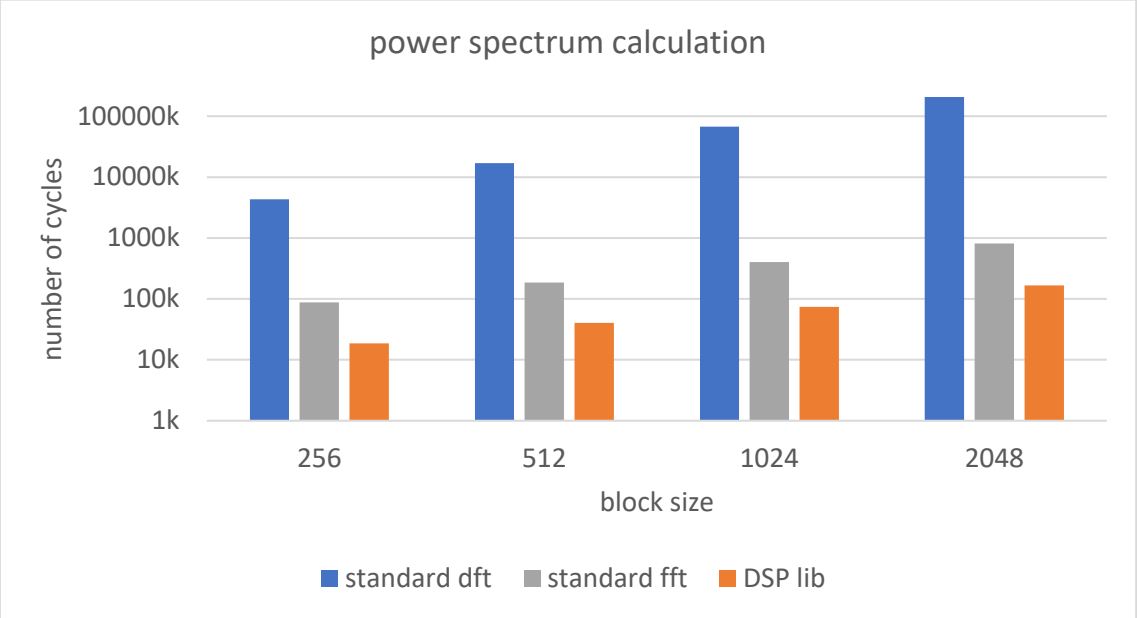


Abb. 3.3 - Performance-Vergleich von drei Varianten zur Bestimmung des Leistungsspektrums

Um den dargestellten Geschwindigkeitsvorteil im Vergleich zu anderen FFT-Implementierungen zu erreichen, nutzt die DSP-Bibliothek intern zahlreiche Lookup-Tabellen, die die Codegröße signifikant steigern. Auf vielen Zielsystemen kann es dadurch zu Problemen kommen, weswegen hier eine manuelle Anpassung nützlich sein kann: Standardmäßig verwendet die DSP-Bibliothek alle Lookup-Tabellen, auch wenn diese gar nicht benötigt werden. Eine Übersicht über diese befindet sich in „arm_common_tables.h“. Falls eine Anwendung wie die verwendete Merkmalsextraktion nur eine einzige Blocklänge bei der FFT-Berechnung verwendet, kann die Funktion arm_rfft_fast_init_f32() manuell nochmal neu implementiert werden, wodurch die Standardimplementierung überschrieben wird. In der darin enthaltenen Switch-Case-Verzweigung können die nicht benötigten Codeabschnitte dabei über Präprozessordirektiven exkludiert werden, damit der Compiler die entsprechenden Lookup-Tabellen wegoptimieren kann. Der folgende Auszug stellt dieses Vorgehen exemplarisch für die Blocklänge von 512 dar:

```
#if LENGTH_FFT == 512     // this line was added
  case 256U:
    Sint->bitRevLength = ARMBITREVINDEXTABLE_256_TABLE_LENGTH;
    Sint->pBitRevTable = (uint16_t *)armBitRevIndexTable256;
    Sint->pTwiddle     = (f32_t *) twiddleCoef_256;
    S->pTwiddleRFFT    = (f32_t *) twiddleCoef_rfft_512;
    break;
 #endif                   // this line was added
```

Zu beachten ist dabei, dass der Integer-Wert im entsprechenden Case (hier 256U) die Hälfte der Blocklänge ist, da sich der Case-Block auf die Berechnung der komplexen FFT bezieht, auch wenn reelle Eingangsdaten vorliegen.

3.2. Signalvorverarbeitung

Als **Antialiasing-Filter** wird ein passiver Tiefpass bestehend aus einem Widerstand (220 Ω) und einem Elektrolytkondensator (0,1 µF) verwendet. Betrachtungen am Oszilloskop zeigen, dass das Filter in Kombination mit dem vorgeschalteten MAX9814-Verstärkermodul die unerwünschten Signalanteile über 8 kHz ausreichend dämpft und auf diese Art Aliasing verhindert. Der verwendete **AD-Wandler** tastet das Signal mit einer Abtastrate von 16 kHz interruptgetrieben ab. Für das korrekte Timing der Abtastung sorgt dabei der Enhanced Periodic Interrupt Timer (EPIT) und das Signal wird mit zehn Bits quantisiert. Abb. 3.4 zeigt den Aufbau und die Verdrahtung des UDOO Neo mit dem Mikrofonmodul und dem Antialiasingfilter.

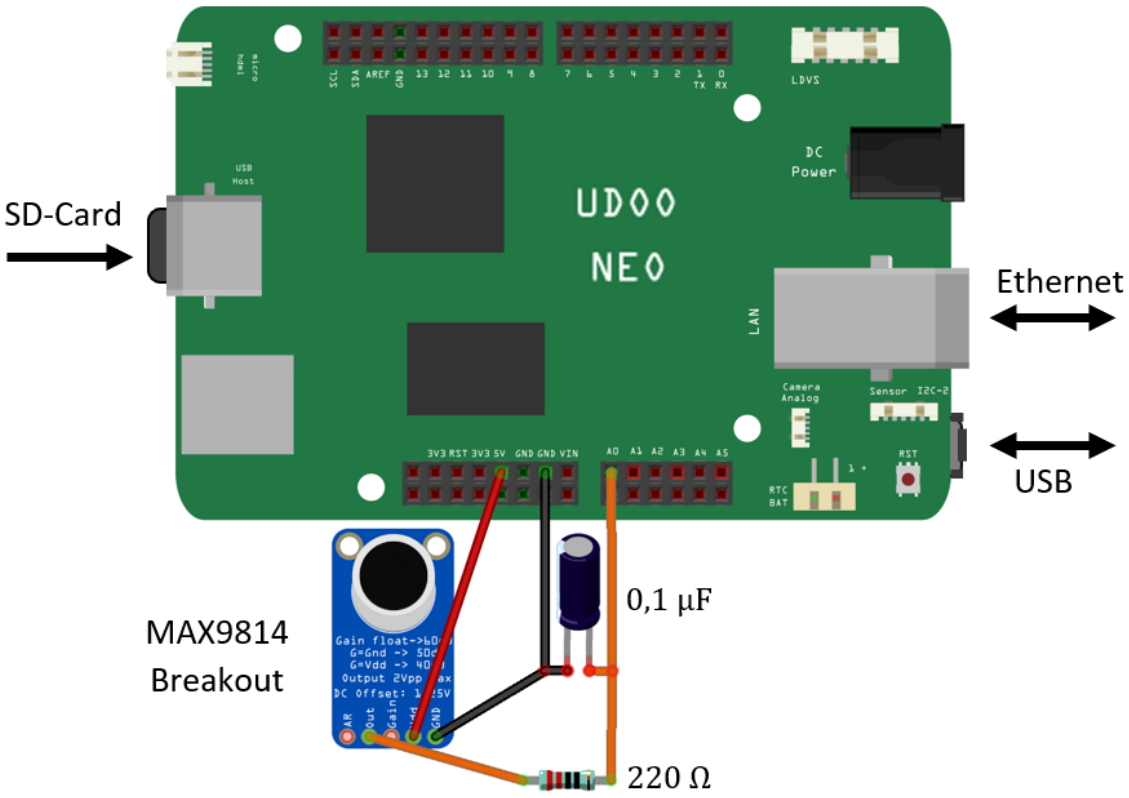


Abb. 3.4 - Verdrahtung des UDOO Neo mit dem Mikrofonmodul

Das in Kapitel 2.2.4 vorgestellte **Präemphase-Filter** bewirkt eine zunehmende Verstärkung für hohe Frequenzen. In bestimmten Szenarien kann die Spracherkennung allerdings dadurch auch beeinträchtigt werden, da die Frequenzen über 5 kHz kaum Informationen über das Sprachsignal enthalten. Deshalb soll hier exemplarisch ein digitales Filter entworfen werden, welches den Gleichanteil des Signals entfernt und zudem die mittleren Frequenzen anhebt, ohne die hohen Frequenzen zu sehr zu verstärken. Das Filter soll also bis Frequenzen von ca.

5 kHz den Amplitudengang des Präemphase-Filters abbilden, darüber hinaus aber nicht weiter verstärken, da in den höheren Regionen kaum mehr Informationen über das Sprachsignal enthalten sind.

Hinweis: Grundsätzlich sind an dieser Stelle verschiedene Hochpassfilter möglich. Der folgende Abschnitt soll dabei vor allem die Vorgehensweise für einen passenden Filterentwurf vorstellen und wurde im fertigen System verwendet, weil sich die Erkennung dadurch im Vergleich zum gewöhnlichen Präemphase-Filter leicht verbessert hat.

Das Filter wird wie folgt anhand des Pol-Nullstellen-Diagramms (Abb. 3.5) entworfen:

- Zur vollständigen Entfernung des Gleichanteils muss eine Nullstelle bei $z = 1$ liegen.
- Um das Verhalten des Präemphase-Filters nachzubilden wird hier ebenfalls ein Pol in den Ursprung der z-Ebene gelegt.
- Einer von zwei weiteren konjugiert komplexen Polen sollte näherungsweise einen Winkel entsprechend der Frequenz haben, die maximal verstärkt werden soll. Je näher diese beiden Pole am Einheitskreis liegen, desto größer ist die maximale Verstärkung. Damit das Filter stabil bleibt, müssen die Pole aber auf jeden Fall innerhalb des Einheitskreises liegen. Der Winkel des Pols in der oberen z-Ebene lässt sich dabei wie folgt bestimmen:

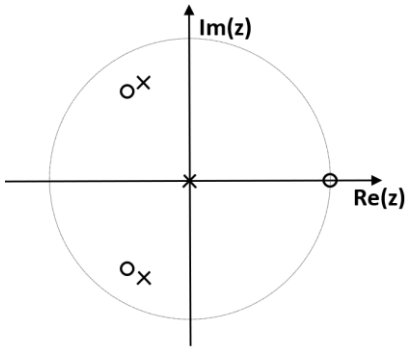


Abb. 3.5 - Pol-Nullstellen-Diagramm des entworfenen Filters

$$\varphi = \frac{f}{f_s} \cdot 2\pi \tag{3.1}$$

f: Frequenz der maximalen Verstärkung (näherungsweise), maximal halbe Abtastfrequenz

f_s: Abtastfrequenz, hier 16 kHz

- Um den Einfluss der beiden zusätzlichen Pole für tiefe Frequenzen zu neutralisieren und hohe Frequenzen zu dämpfen, werden direkt daneben zwei konjugiert komplexe Nullstellen gelegt, deren Betrag identisch dem der Pole ist.
- Um das Präemphase-Filter möglichst exakt nachzubilden, wird noch ein konstanter Vorfaktor eingefügt.

In Abb. 3.6 ist der Amplitudengang des entworfenen Filters (dicke Linie) und zum Vergleich des gewöhnlichen Präemphase-Filters (gestrichelt) dargestellt. Um den Amplitudengang des angepassten Filters zu realisieren, ergibt sich dabei folgende z-Übertragungsfunktion:

$$H(z) = 0{,}8 \cdot \frac{(z-1)\cdot\left(z-0{,}77e^{2{,}18j}\right)\cdot\left(z-0{,}77e^{-2{,}18j}\right)}{z\cdot(z-0{,}77e^{2{,}01j})\cdot(z-0{,}77e^{-2{,}01j})}$$

$$= 0{,}8 \cdot \frac{1-0{,}119z^{-1}-0{,}288z^{-2}-0{,}593z^{-3}}{1+0{,}655z^{-1}+0{,}593z^{-2}} \qquad (3.2)$$

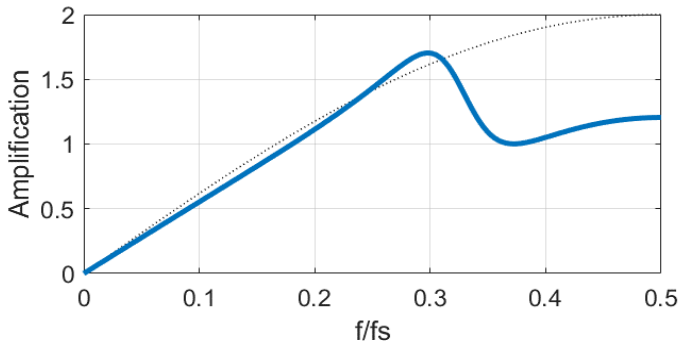


Abb. 3.6 - Übertragungsfunktion des verwendeten digitalen Hochpassfilters

Durch Anwendung der z-Rücktransformation ergibt sich folgende Differenzengleichung, die auch mit weniger Speicherbausteinen (wie in Abb. 3.7 gezeigt) realisiert werden kann:

$$y[n] = 0{,}8\cdot x[n] - 0{,}095\cdot x[n-1] - 0{,}23\cdot x[n-2] - 0{,}474\cdot x[n-3]$$
$$- 0{,}655\cdot y[n-1] - 0{,}593\cdot y[n-2] \qquad n \in \mathbb{Z} \qquad (3.3)$$

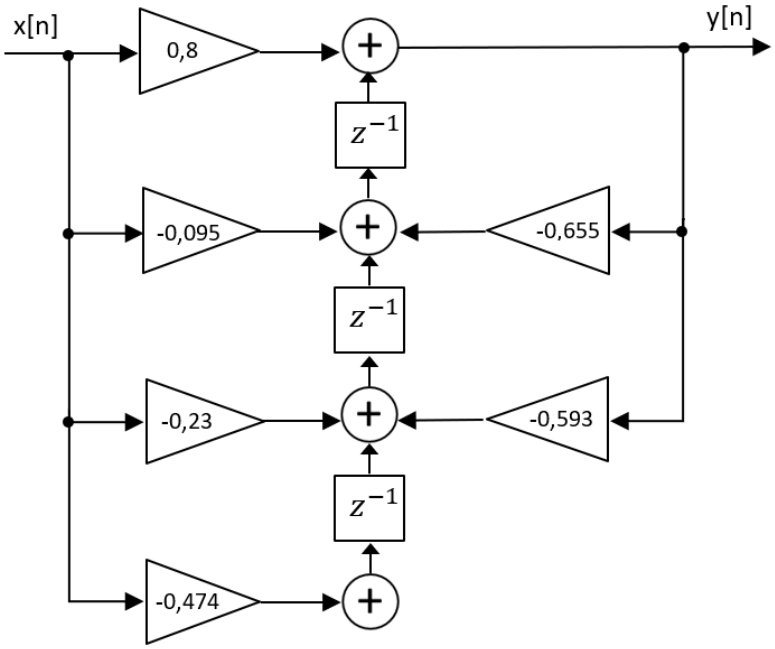


Abb. 3.7 - Erste kanonische Filterstruktur des entworfenen digitalen Hochpassfilters

3.3. MFCC-Merkmalsextraktion

Die Merkmalsextraktion wird grundsätzlich wie in Kapitel 2.3 beschrieben durchgeführt. Tabelle 3.1 zeigt dabei die verwendeten Parameter und fasst nochmal den groben Ablauf zusammen:

Parameter		*Beschreibung*
Blocklänge	512 Samples	Anzahl der Abtastwerte in einem Verarbeitungsblock, entspricht bei 16 kHz einer Blockdauer von 32 ms
Blockverschiebung	160 Samples	Die Blöcke überlappen sich, da die Blockverschiebung geringer ist als die Blocklänge. Die Blockverschiebung entspricht 10 ms
Fensterfunktion	Hamming	Wird mit jedem Block multipliziert, um den Leckeffekt zu reduzieren
Frequenzauflösung des Spektrums	31,25 Hz	Wird definiert durch die FFT-Länge (512) und die Abtastfrequenz (16 kHz)
Frequenzbereich der Filterbank	0 Hz bis 8 kHz	Gibt den Bereich der verwendeten Dreiecksfilter an
Anzahl Dreiecksfilter	30	Entspricht der Anzahl der resultierenden Frequenzbänder
Anzahl MFCC-Koeffizienten	13	Entspricht der Anzahl der berechneten DCT-Koeffizienten (der Rest wird verworfen)
Fenster für Delta-Berechnung	5	Gibt an, über wie viele Blöcke die zeitliche Ableitung berechnet wird

Tabelle 3.1 Übersicht der verwendeten Parameter zur MFCC-Merkmalsextraktion

Als Fensterfunktion wird das Hamming-Fenster verwendet. Dieses ist in Abb. 3.8 dargestellt und lässt sich wie folgt berechnen:

$$w[n] = 0{,}54 - 0{,}46 \cdot cos\left(\frac{2\pi \cdot n}{N_b - 1}\right) \quad , \quad 0 \leq n < N_b \qquad (3.4)$$

N_b: Blocklänge

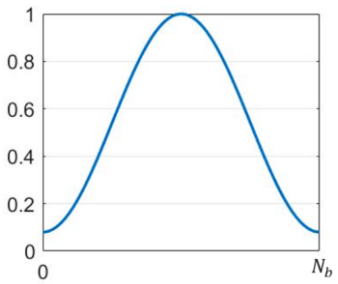


Abb. 3.8 - Hamming-Fenster

Bei Verwendung des DTW-Algorithmus wird der Mustervergleich der Merkmale zudem noch etwas robuster gegenüber stationärem Rauschen gemacht, indem vor der diskreten Kosinustransformation der Mittelwert der spektralen Energien von den aktuellen Werten abgezogen wird („Spectral Subtraction“). Dadurch ist es nicht mehr zwingend nötig, dass das

Referenzmuster und die zu vergleichende Sequenz in der gleichen Umgebung aufgenommen werden, weil stationäre Hintergrundgeräusche (zumindest teilweise) eliminiert werden. Bei der KNN-Erkennung bringt dieser Ansatz keine Verbesserung, da auch schon in den Trainingsdaten unterschiedliche akustische Umgebungen vorliegen.

Laut der angegebenen Parameter ergeben sich also 13 MFCC-Koeffizienten und 13 Delta-Koeffizienten. Allerdings wird der erste MFCC-Wert verworfen, da dieser ein Indikator für die Gesamtenergie des Spektrums darstellt, es ergeben sich folglich pro Block 25 Fließkommawerte, die über die OpenAMP-Schnittstelle an den Cortex-A9 weitergeleitet werden. Entsprechend der verwendeten Blockverschiebung wird bei aktiver Kommunikation alle 10 ms ein Datenpaket gesendet.

3.4. Spracherkennungsmodul

Auf der Cortex-A9-Seite werden die Informationen der Merkmalsvektoren empfangen und ausgewertet. Die eigentliche Spracherkennung wird in zwei Varianten entwickelt, welche in diesem Kapitel vorgestellt werden. Abb. 3.9 zeigt die grafische Oberfläche, auf der sich die entsprechende Variante auswählen lässt. Das Bedienkonzept der Anwendung ist in Anhang A.3 erläutert.

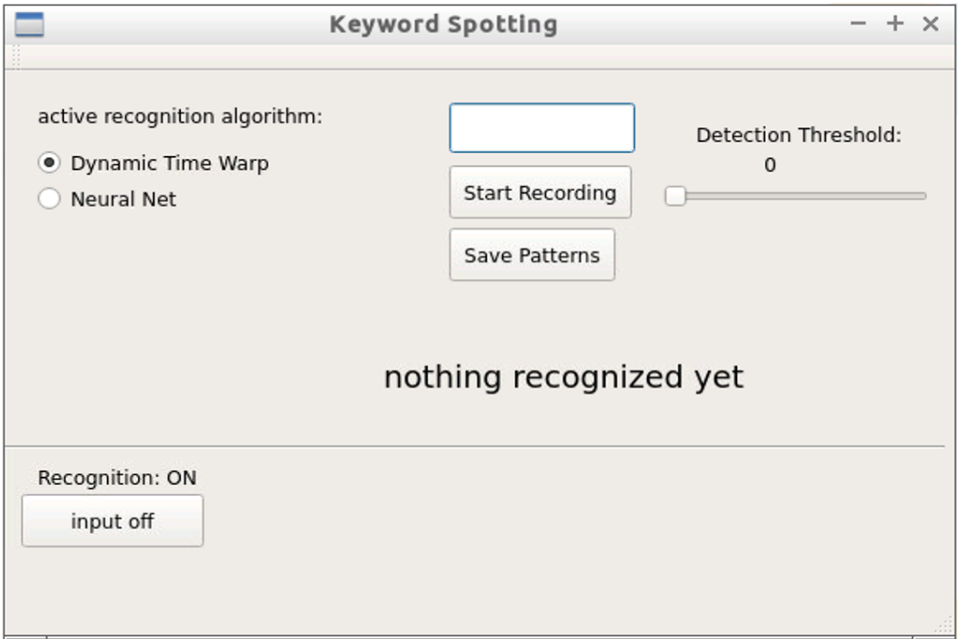


Abb. 3.9 - Grafische Oberfläche der implementierten Demoanwendung

3.4.1. Mustervergleich mit DTW

Der Mustervergleich wird wie in Kapitel 2.4 beschrieben mit Dynamic Time Warp durchgeführt. Für den Einsatz des DTW-Algorithmus auf einem System mit begrenzten Ressourcen können verschiedene Optimierungen durchgeführt werden, die nachfolgend erläutert werden. Der Quellcode der Implementierung ist zudem in Anhang A.5 zu finden.

Zunächst einmal wirkt sich schon die funktionale Anpassung durch eine Bereichslimitierung positiv auf Rechenzeit und Speicherverbrauch aus. Hier soll die in Abb. 3.10 schematisch dargestellte Begrenzung verwendet werden, da diese im Vergleich zum Itakura-Parallelogram (Abb. 2.16) auch einen variablen Anfangszeitpunkt ermöglicht.

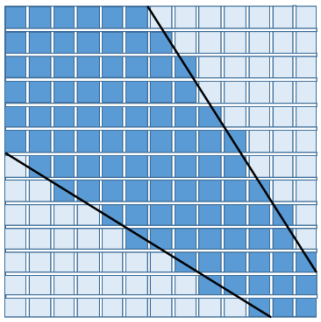


Abb. 3.10 - Schematische Darstellung der verwendeten DTW-Bereichsbegrenzung

Der DTW-Algorithmus kann aufgrund seines hohen Rechenaufwands in der Echtzeit-Spracherkennung zu Problemen führen. Denn je mehr Wörter unterschieden werden müssen, desto öfter muss er durchlaufen werden. Gerade wenn für ein Wort mehrere Muster abgespeichert sind (Multi-Template-DTW) kann es schnell zu Engpässen bei der Rechenzeit kommen. Deshalb sollte der Algorithmus nicht immer komplett durchlaufen werden, wie die beiden folgenden Punkte zur **Laufzeitoptimierung** zeigen.

- Wenn die kumulierte Distanz zu einem (nicht zu frühen) Zeitpunkt der Berechnung in einer kompletten Zeile oder in einer kompletten Spalte sehr hoch ist, so kann das Referenzwort als nicht erkannt angenommen und die Berechnung abgebrochen werden. Dadurch handelt man sich zwar potenziell Fehler ein, da nicht mehr alle möglichen Pfade betrachtet werden, aber durch einen passenden Schwellwert überwiegt der Vorteil einer schnelleren Berechnung.
- Die Berechnung der Distanz zwischen den einzelnen Merkmalsvektoren benötigt den Großteil der Rechenzeit des Dynamic Time Warp. Wenn diese Berechnung erst während der Pfadsuche (und nicht vorher) erfolgt, so kann sie immer dann weggelassen werden, wenn alle relevanten Vorgängerfelder sowieso schon eine sehr hohe kumulierte Distanz haben. Durch einen passenden Schwellwert können so ganze Berechnungszweige eliminiert werden. Auch hier können Erkennungsfehler nicht ausgeschlossen werden.

In der Implementierung des Keyword-Spotting-Systems wird lediglich der zweite Punkt zur Laufzeitoptimierung umgesetzt, da dieser die erste Variante überflüssig macht. Der Algorithmus wird dann zwar immer bis zum letzten Feld durchlaufen (und nicht vorher abgebrochen), aber dies macht nahezu keinen Unterschied aus, da die Berechnung der Einzeldistanzen trotzdem verhindert wird.

Wenn der DTW-Algorithmus auf einem System mit begrenzten Ressourcen läuft, kann zudem eine **Speicheroptimierung** sinnvoll sein, da die Matrizen der Einzeldistanzen und der kumulierten Distanzen bei längeren Sequenzen recht schnell eine kritische Größe erreichen. Diese sind aber nur zwingend nötig, wenn der optimale Pfad durch eine Rückwärtssuche nachverfolgt werden soll. Wenn die Anwendung wie hier nur am Distanzmaß der zu vergleichenden Sequenzen interessiert ist, so kann auf die Speicherung der Matrizen verzichtet werden. Stattdessen kann ein eindimensionales Array als Zwischenspeicher verwendet werden. Im einfachsten Fall hat dieses Array die Größe einer Zeile und speichert die kumulierten Distanzen der vorherigen Zeile. Die Einzeldistanzen zwischen zwei Merkmalsvektoren werden dabei erst berechnet, wenn der entsprechende Wert benötigt wird und nicht schon zu Beginn. Ein weiteres Array der gleichen Länge kann die entsprechenden Pfadlängen speichern. Bei der Verwendung der Bereichsbegrenzung aus Abbildung Abb. 3.10 kann die Größe dieser Arrays zudem auf die maximale Breite des Bereichs in einer Zeile reduziert werden.

Beim Einsprechen der Schlüsselworte sollte ein geeigneter Schwellwert für die Detektion eingestellt werden. Grundsätzlich funktioniert die Erkennung mit dem voreingestellten Schwellwert. Ein optimaler Wert muss aber je nach Schlüsselwort und Sprecher experimentell ermittelt werden. Auf diese Weise ergibt sich in Tests auf der Zielplattform eine sehr hohe Genauigkeit und Zuverlässigkeit bei der Mustererkennung, wenn das zu erkennende Wort und das Referenzmuster von der gleichen Person gesprochen werden.

3.4.2. Neuronales Netz

Für die sprecherunabhängige Erkennung wird alternativ ein neuronales Netz mithilfe der Software MemBrain entwickelt. Als Topologie dient ein CNN, um Muster in den Eingangsdaten unabhängig von ihrem zeitlichen Auftreten detektieren zu können. Der Netzaufbau ist in Abb. 3.11 dargestellt. Ausgehend von einem Eingangsbild (Merkmale über der Zeit aufgetragen) werden nacheinander ein *convolutional layer*, ein *pooling layer* und zwei *fully connected layer* durchlaufen, bevor die Ausgangsschicht erreicht ist. Am Ausgang deutet ein Neuron durch einen Aktivierungswert nahe eins an, dass das entsprechende Wort erkannt ist, und durch einen Wert nahe null, dass das Wort nicht erkannt ist.

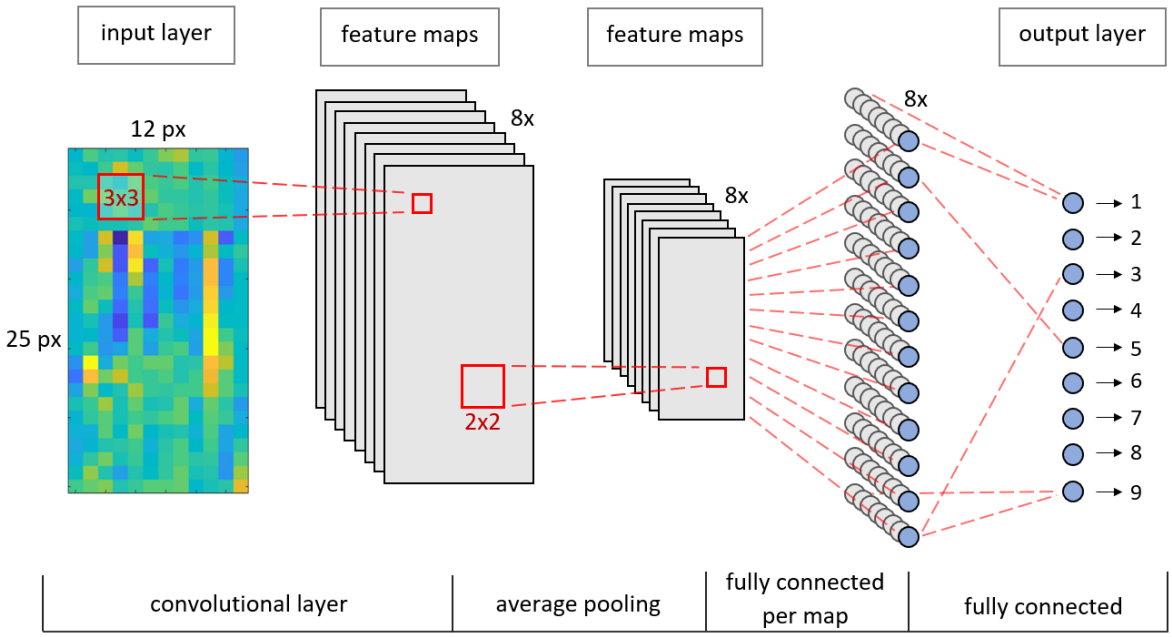


Abb. 3.11 - Schematischer Aufbau des verwendeten neuronalen Netzes

Als Pooling-Verfahren wird hier nicht das übliche Max-Pooling eingesetzt, weil sich dieses nicht ohne Weiteres in MemBrain implementieren lässt. Stattdessen kann aber eine ähnliche Funktion durch das Average-Pooling realisiert werden, wenn vorher alle negativen Werte auf null gesetzt werden. Dies geschieht durch eine ReLU-Aktivierungsfunktion im Convolutional Layer.

Die weiteren Parameter des Netzes sind in Tabelle 3.2 dargestellt und werden in MemBrain entsprechend umgesetzt.

Schicht	*Parameter*		*Beschreibung*
Convolution	Filtergröße	3x3	Ein Pixel/Neuron in der Feature Map ist mit neun Pixel/Neuronen in der Eingangsmatrix verbunden
	Schrittweite	1	Damit überlappt sich ein verschobener Block mit dem Vorgänger um zwei Pixel
	Aktivierungsfunktion	ReLU	Die Summe der gewichteten Eingänge wird an den Ausgang übernommen, negative Aktivierungen werden auf null gesetzt
Pooling	Filtergröße	2x2	Durchschnittsbildung aus vier Pixeln
	Schrittweite	2	Die Pooling-Blöcke überlappen sich dadurch nicht
	Aktivierungsfunktion	ReLU	Die Summe der gewichteten Eingänge wird an den Ausgang übernommen, negative Aktivierungen werden auf null gesetzt
Fully Connected	Aktivierungsfunktion	Logistic	Ähnlich zum Tangens Hyperbolicus, jedoch liegt die Aktivierung zwischen null und eins

Tabelle 3.2 Übersicht über die Parameter des verwendeten neuronalen Netzes

Als Trainings- und Test-Daten wird das „Google Speech Commands Dataset“ verwendet [15], welches ca. 65000 Audiodateien mit einer Dauer von jeweils einer Sekunde enthält. Diese wurden von mehreren tausend Personen in Form von 30 verschiedenen englischen Worten eingesprochen. Für die zu erstellende Demoanwendung wird der Ausgang des neuronalen Netzes auf die Zahlen von eins bis neun trainiert. Da für das Training aber nicht einfach Sprachdaten eingegeben werden können, werden in einem Zwischenschritt zunächst die Merkmale aus den Audiosignalen extrahiert[2], wobei zur Datenreduktion immer nur eine halbe Sekunde der Dateien verwendet wird (das entsprechende Wort sollte natürlich in der halben Sekunde vollständig vorkommen). Nach ca. drei Stunden Training[3] mit dem RPROP-Trainingsalgorithmus lässt sich keine weitere Verbesserung der Genauigkeit auf den Validierungsdaten mehr feststellen, weswegen das Training dann abgebrochen wird. Mithilfe des integrierten Code-Generators lässt sich dann C-Code generieren, welcher sehr einfach in die Qt-Anwendung des Cortex-A9 integriert werden kann. Die Entscheidung, ob eine

[2] ohne zeitliche Ableitung und der erste Koeffizient wird verworfen
[3] ausgeführt auf einem i5-6500-Prozessor

gesprochene Zahl für einen Eingangsdatenblock als erkannt angenommen werden soll, erfolgt im Wesentlichen über drei Funktionsaufrufe:

- *NeuralNetApplyInputAct()* zum Anlegen der Eingangsdaten
- *NeuralNetThinkStep()* zum Berechnen der Ausgänge
- *NeuralNetGetOutputAct()* zum Auslesen der Ausgangsdaten

Zusätzlich wird dann noch eine Mittelung über mehrere Zeitschlitze durchgeführt, um schnelle Schwankungen der Ausgänge zu vermeiden.

Beim Testen dieser Variante auf der Zielplattform zeigt sich, dass die meisten gesprochenen Zahlen in leiser Umgebung zuverlässig erkannt werden. Lediglich das Wort „six“ wird nicht immer registriert und oft fälschlicherweise auch bei rauschähnlichen Lauten erkannt. Um diese Probleme zu lösen und die Erkennung insgesamt noch zuverlässiger zu machen, können folgende Verbesserungen in Betracht gezogen werden:

- Verwendung der gesamten Audiodateien aus der Datenbank und nicht nur eine halbe Sekunde pro Datei, da gerade auch die Bestimmung der besten Position des Zeitfensters automatisiert nicht optimal möglich ist.
- Herausfiltern von unbrauchbaren Dateien aus der Datenbank (manuell oder automatisiert), da manche Dateien nur Bruchteile des zu erkennenden Wortes enthalten. Diese Dateien verschlechtern dann entsprechend das Training des neuronalen Netzes.
- Wiederholung oder Hinzufügen von Audiodateien mit weiblichen Sprechern, da diese deutlich seltener in der Datenbank vorkommen, wodurch deren Erkennung auch schlechter funktioniert.
- Hinzufügen von verschiedenen Rauschquellen und Umgebungsgeräuschen, wie es in [16] vorgeschlagen wird.
- Verwendung eines tieferen neuronalen Netzes (bei Bedarf auch mit anderer Topologie), um noch komplexere Zusammenhänge zwischen Eingangs- und Ausgangsdaten zu erkennen. Eventuell wird dann der Einsatz einer anderen Software nötig, da die grafische Oberfläche von MemBrain bei sehr großen Netzen an ihre Grenzen stößt.

Vor allem die ersten beiden Punkte können die Genauigkeit der Worterkennung noch erheblich verbessern, da abgeschnittene Wörter im Trainingsdatensatz einen stark negativen Einfluss auf das neuronale Netz haben. Variationen in der Netztopologie konnten hingegen keine Verbesserung bei den gewählten Trainingsdaten bewirken.

4. Fazit

In dieser Abschlussarbeit wurde eine Anwendung entwickelt, die Sprachbefehle erkennt und entsprechende Anweisungen an einen MQTT-Broker schickt. Es konnte gezeigt werden, dass ein Spracherkennungssystem auch mit begrenzten Ressourcen auf einem Embedded System möglich ist und nicht zwingend in die Cloud ausgelagert werden muss. In der entwickelten Anwendung kann für die Spracherkennung zwischen zwei Verfahren ausgewählt werden. Die erste Variante führt einen Mustervergleich der aufgenommenen Merkmalsvektoren mit Dynamic Time Warp durch, um die Ähnlichkeit zwischen den Sequenzen im laufenden Betrieb und den Referenzmustern zu bestimmen. Da die MFCC-Merkmale selbst vom Sprecher abhängen, werden in der Regel nur Wörter von Personen erkannt, die diese vorher auch als Referenzmuster eingesprochen haben. Dies kann in einer realen Anwendung ein Vorteil sein, wenn das System nur vom Besitzer oder einem bestimmten Anwender kontrolliert werden soll, oder aber ein Nachteil, wenn es von verschiedenen Personen verwendet werden soll. In der zweiten Variante erfolgt die Erkennung stattdessen mithilfe eines neuronalen Netzes, wodurch auch verschiedene Sprecher erkannt werden. Die Anwendung erkennt hier grundsätzlich neun verschiedene Worte, ist aber insgesamt auch weniger zuverlässig als mit dem DTW-Algorithmus. In Anhang A.5 befinden sich zwei Videos der Demoanwendung für die beiden Erkennungsverfahren, um die Funktionsweise nochmals anschaulich zu demonstrieren.

Während der Entwicklung traten leider auch einige Probleme im Zusammenhang mit der verwendeten Toolchain und Zielplattform auf, welche im Rahmen dieser Arbeit teilweise gelöst oder teilweise auch nur umgangen wurden. So muss beispielsweise das gesamte System immer neu gestartet werden, bevor die Kommunikation der beiden Prozessorkerne funktioniert. Eine vollständige Liste der aufgetretenen Probleme mit den entsprechenden Lösungen befindet sich in Anhang A.4. Rückblickend betrachtet wäre es wohl einfacher gewesen, die gewünschte Funktionalität auf einem separaten Cortex-M4-Mikrocontroller in Kombination mit einem Raspberry-Pi (oder einem vergleichbaren Single-Board-Computer) zu implementieren. Gerade auch die Dokumentation des UDOO Neo Boards selbst lässt sehr zu wünschen übrig, da es kein richtiges User Manual gibt, sondern lediglich eine vergleichsweise kleine Online-Dokumentation.

5. Ausblick

Abschließend soll nun noch ein kurzer Ausblick über die weiteren Möglichkeiten der Embedded Spracherkennung gegeben werden, welcher über die vorgestellte Anwendung hinaus geht. Die beiden betrachteten Erkennungsverfahren (DTW und KNN) wurden auf dem Cortex-A9 des UDOO-Neo Boards implementiert, da dieser eine deutlich größere Rechenleistung mit sich bringt als der Cortex-M4. Aufbauend auf der CMSIS-DSP-Bibliothek bietet das Unternehmen ARM seit Januar 2018 aber auch eine Bibliothek zur effizienten Berechnung von neuronalen Netzen auf Cortex-M-Prozessoren an. Dadurch wird es möglich, die Echtzeitspracherkennung auch allein auf dem Cortex-M4 zu implementieren. Als Toolchain werden hierzu basierend auf Google´s Tensorflow-Framework verschiedene Python-Skripte bereitgestellt. Durch diese ist es möglich, vortrainierte oder eigene mit Tensorflow erstellte Modelle auf Cortex-M-Mikrocontroller zu portieren [17]. Das Konzept wurde im Rahmen dieser Thesis ansatzweise anhand der in [17] beschriebenen Beispiele getestet. Die resultierende Genauigkeit auf der Hardware ist dabei zwar nicht perfekt, jedoch lassen sich die meisten Worte recht zuverlässig und ebenfalls in Echtzeit erkennen. Der entsprechende Aufbau des Spracherkennungssystems ist in Abb. 5.1 dargestellt.

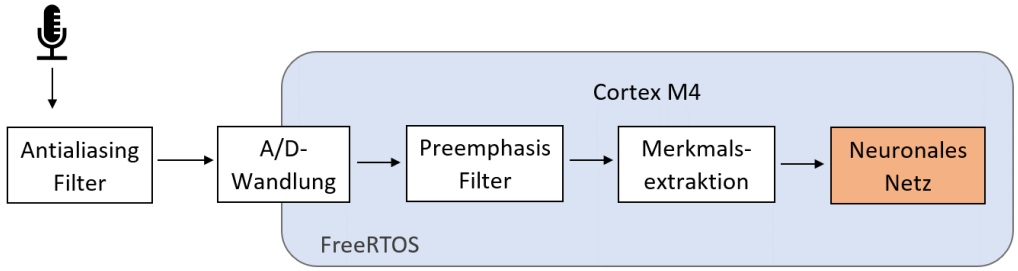


Abb. 5.1 - Aufbau eines Spracherkennungssystems unter Verwendung der CMSIS-NN-Bibliothek

Eine solche Realisierung auf einem Cortex-M4 ist natürlich gerade für Low-Cost-Anwendungen sehr interessant. Es kann deshalb lohnend sein, sich weiterhin mit dem Thema zu beschäftigen, zumal die Software noch recht neu ist. In näherer Zukunft ist auch mit weiteren Verbesserungen und Erweiterungen der Bibliothek zu rechnen.

Literaturverzeichnis

[1] Maxim Integrated, August 2016. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX9814.pdf. [Zugriff am 20. Dezember 2018].

[2] B. Pfister und T. Kaufmann, Sprachverarbeitung - Grundlagen und Methoden der Sprachsynthese und Spracherkennung, Berlin: Springer-Verlag, 2017, pp. 8-100.

[3] E. B. Brixen, „Facts About Speech Intelligibility," DPA Microphones, 20. Januar 2016. [Online]. Available: https://www.dpamicrophones.com/mic-university/facts-about-speech-intelligibility. [Zugriff am 5. Februar 2019].

[4] R. G. Lyons, „DSP Tricks - DC-Removal," 11. August 2008. [Online]. Available: https://www.embedded.com/design/configurable-systems/4007653/DSP-Tricks-DC-Removal. [Zugriff am 5. Februar 2019].

[5] Y. A. Ibrahim, J. C. Odiketa und T. S. Ibiyemi, „Preprocessing Technique In Automatic Speech Recognition For Human Computer Interaction: An Overview," 2017. [Online]. Available: http://anale-informatica.tibiscus.ro/download/lucrari/15-1-23-Ibrahim.pdf. [Zugriff am 5. Februar 2019].

[6] J. Lyons, „Mel Frequency Cepstral Coefficient (MFCC) tutorial," [Online]. Available: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/. [Zugriff am 5. Februar 2019].

[7] „Wikipedia - Diskrete Kosinustransformation," 2017. [Online]. Available: https://de.wikipedia.org/wiki/Diskrete_Kosinustransformation. [Zugriff am 13 Dezember 2018].

[8] M. Morel, C. Achard, R. Kulpa und S. Dubuisson, „Time-series averaging using constrained dynamic time warping with tolerance," 7. November 2017. [Online]. Available: https://hal.sorbonne-universite.fr/hal-01630288/document. [Zugriff am 5. Februar 2019].

[9] G. D. Rey und K. F. Wender, Neuronale Netze - Eine Einführung in die Grundlagen, Anwendungen und Datenauswertung, 3. Hrsg., Bern: Hogrefe-Verlag, 2018.

[10] „Wikipedia - Backpropagation," 2019. [Online]. Available: https://de.wikipedia.org/wiki/Backpropagation. [Zugriff am 3. Februar 2019].

[11] H. Lee, Y. Largman, P. Pham und A. Y. Ng, „Unsupervised feature learning for audio classification using convolutional deep belief networks," Stanford University.

[12] D. Rothmann, „What's wrong with CNNs and spectrograms for audio processing?," 26. März 2018. [Online]. Available: https://towardsdatascience.com/whats-wrong-with-spectrograms-and-cnns-for-audio-processing-311377d7ccd. [Zugriff am 5. Februar 2019].

[13] HiveMQ, „MQTT Essentials - Quality of Service," 16 Februar 2015. [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/. [Zugriff am 8. Januar 2018].

[14] T. Lorenser, „The DSP capabilities of ARM Cortex-M4 and Cortex-M7 Processors," ARM Limited, 2016.

[15] P. Warden, „Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," April 2018. [Online]. Available: https://arxiv.org/pdf/1804.03209.pdf. [Zugriff am 5. Februar 2019].

[16] Y. Zhang, N. Suda, L. Lai und V. Chandra, „Hello Edge: Keyword Spotting on Microcontrollers," Stanford University, 2018.

[17] N. Suda, „ML-KWS for MCU," 2018. [Online]. Available: https://github.com/ARM-software/ML-KWS-for-MCU. [Zugriff am 20. Januar 2019].

[18] ARM, „CMSIS - DSP Source Code," 2018. [Online]. Available: https://github.com/ARM-software/CMSIS_5/tree/develop/CMSIS/DSP/Source. [Zugriff am 26. November 2018].

[19] Nayuki, „Free small FFT in multiple languages," 2018. [Online]. Available: https://www.nayuki.io/page/free-small-fft-in-multiple-languages. [Zugriff am 7. Dezember 2018].

Anhang

A.1 Optimierte Kopierfunktion

Der ursprüngliche Code steht auf Github zur Verfügung [18]. Die Funktion *arm_copy_f32()* wurde wie folgt angepasst, um möglichst schnell abzulaufen:

```
void arm_copy_own(float32_t * pSrc, float32_t * pDst, uint32_t blockSize)
{
  uint32_t blkCnt; // loop counter
  float32_t in1, in2, in3, in4, in5, in6, in7, in8;
  float32_t in9, in10, in11, in12, in13, in14, in15, in16;
  // loop Unrolling
  blkCnt = blockSize >> 4U;

  // First part of the processing with loop unrolling.Compute 16 outputs
  // at a time.
  // a second loop below computes the remaining 1 to 15 samples.
  while (blkCnt > 0U)
  {
    // Copy and then store the results in the destination buffer
    in1 =  *pSrc++; in2  =  *pSrc++; in3 = *pSrc++; in4  = *pSrc++;
    in5 =  *pSrc++; in6  =  *pSrc++; in7 = *pSrc++; in8  = *pSrc++;
    in9 =  *pSrc++; in10 = *pSrc++; in11 = *pSrc++; in12 = *pSrc++;
    in13 = *pSrc++; in14 = *pSrc++; in15 = *pSrc++; in16 = *pSrc++;

    *pDst++ = in1;  *pDst++ = in2;  *pDst++ = in3;  *pDst++ = in4;
    *pDst++ = in5;  *pDst++ = in6;  *pDst++ = in7;  *pDst++ = in8;
    *pDst++ = in9;  *pDst++ = in10; *pDst++ = in11; *pDst++ = in12;
    *pDst++ = in13; *pDst++ = in14; *pDst++ = in15; *pDst++ = in16;

    // Decrement the loop counter
    blkCnt--;
  }

  // If blockSize is not a multiple of 16, compute any remaining output
  // samples here.
  // No loop unrolling is used.
  blkCnt = blockSize % 16U;


  while (blkCnt > 0U)
  {
    // Copy and then store the results in the destination buffer
    *pDst++ = *pSrc++;

    // Decrement the loop counter
    blkCnt--;
  }
}
```

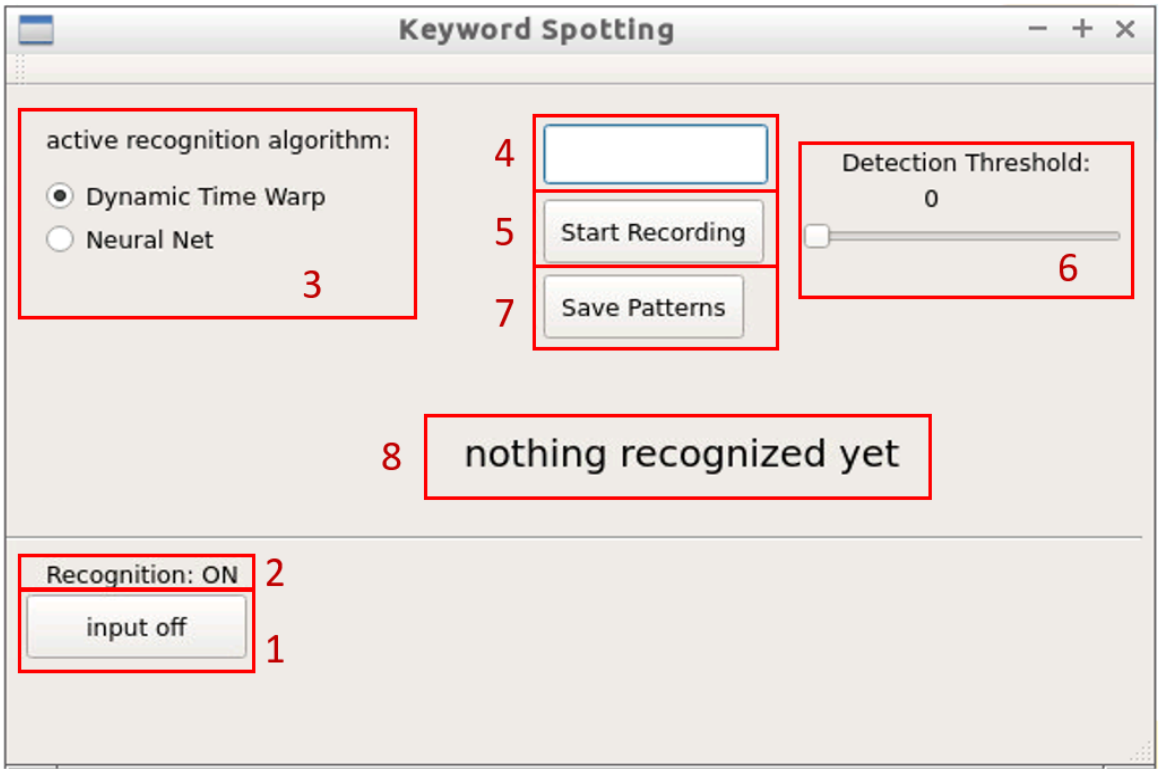
A.2 Radix-2 FFT-Algorithmus

Der verwendete Code basiert auf einer unter der MIT-Lizenz veröffentlichten FFT-Implementierung [19], wurde aber dahingehend angepasst, dass keine dynamische Speicherallokierung erfolgt.

```
Fft_transformRadix2(f32_t* pfData, size_t n)
{
	// Length variables
	Bool status = False;
	int levels = 0;  // Compute levels = floor(log2(n))
	for (size_t temp = n; temp > 1U; temp >>= 1)
	{
		levels++;
	}
	while ((size_t)1U << levels != n); // n is not a power of 2, error

	// Trignometric tables are already initialized with sin and cos values
	f32_t *cos_table = &afBufferSinCos[0];
	f32_t *sin_table = &afBufferSinCos[LENGTH_FFT/2];

	// Bit-reversed addressing permutation
	for (size_t i = 0; i < n; i++)
	{
	    size_t j = reverse_bits(i, levels);
	    if (j > i)
	    {
	        f32_t temp = pfData[2*i];
	        pfData[2*i] = pfData[2*j];
	        pfData[2*j] = temp;
	        temp = pfData[2*i+1];
	        pfData[2*i+1] = pfData[2*j+1];
	        pfData[2*j+1] = temp;
	    }
	}

	// Cooley-Tukey decimation-in-time radix-2 FFT
	for (size_t size = 2; size <= n; size *= 2)
	{
	    size_t halfsize = size / 2;
	    size_t tablestep = n / size;
	    for (size_t i = 0; i < n; i += size)
	    {
	        for (size_t j = i, k = 0; j < i + halfsize; j++, k += tablestep)
	        {
	            size_t l = j + halfsize;
	            f32_t tpre =pfData[2*l]*cos_table[k]+pfData[2*l+1]*sin_table[k];
	            f32_t tpim =-pfData[2*l]*sin_table[k]+pfData[2*l+1]*cos_table[k];
	            pfData[2*l] = pfData[2*j] - tpre;
	            pfData[2*l+1] = pfData[2*j+1] - tpim;
	            pfData[2*j] += tpre;
	            pfData[2*j+1] += tpim;
	        }
	    }
	    if (size == n)  // Prevent overflow in 'size *= 2'
	        break;
	}

	status = True;
	return status;
}
```

A.3 Bedienung der Anwendung

Der folgende Screenshot zeigt die grafische Oberfläche der fertigen Demoanwendung.



Beschreibung der Bedienelemente:

1) Ein- und Ausschalten der Kommunikation zwischen den beiden Prozessorkernen. Bei ausgeschalteter Kommunikation ist automatisch auch die Erkennung deaktiviert.
2) Angabe, ob das Keyword-Spotting aktuell aktiv ist, oder ob es aufgrund der angehaltenen Interprozessorkommunikation oder sonstigen Fehlern ausgeschaltet ist.
3) Über das Optionsfeld *active recognition algorithm* lässt sich auswählen, ob für die Spracherkennung der DTW-Algorithmus oder das Neuronale Netz verwendet werden soll. Bei letzterem werden sprecherunabhängig vortrainierte Worte (hier die englischen Zahlen von eins bis neun) erkannt. Bei der Auswahl von DTW ist die Vorgehensweise in den folgenden Punkten erklärt erklärt.
4) Textfeld für die Eingabe des zu erstellenden oder zu ändernden Schlüsselwortes.
5) Der erste Klick auf den Button startet die Aufnahme eines neuen Schlüsselwortes, ein zweiter Klick beendet diese und speichert das Schlüsselwort unter der in 4) eingegebenen Zeichenkette ab.
6) Über den Regler kann der Erkennungsschwellwert für ein in 4) eingegebenes Schlüsselwort ausgelesen und verändert werden.

7) Abspeichern aller vorhandenen Schlüsselworte auf der SD-Karte. Diese werden beim Start der Anwendung automatisch geladen. Zum Löschen der Daten muss die Datei *SpeechRecognition_save.txt* im Dateisystem gelöscht werden.
8) Ausgabe erkannter Wörter/Kommandos.

A.4 Probleme während der Entwicklung

Während der Entwicklung traten einige Probleme in Bezug auf die verwendete Toolchain und Hardware auf. Die folgende Tabelle gibt einen Überblick über die größten Probleme und ihre Lösungen:

Problem	*Lösung*
Cortex-M4-Seite	
Verwendung von %f in einer sprintf-Anweisung führt zu einem HardFault	Anstatt die float-Werte in einem String zu übertragen, werden sie binär an den Cortex-A9 gesendet
Fehlerhafte FreeRTOS-Heapgröße (durch Ausprobieren) oder andere kritische Speicherüberschreitungen führen zur dauerhaften Unbrauchbarkeit des RPMsg-Moduls	Die SD-Karte mit dem Betriebssystem muss neu geflasht werden
Interprozessorkommunikation / RPMsg	
Beim Senden der binär übertragenen float-Werte geht die Synchronisation verloren, da Bytes mit dem Inhalt Carriage Return (dezimal 13) verschluckt werden	Minimales Ändern der entsprechenden float-Werte vor dem Senden
Initialisierung der Kommunikation schlägt beim manuellen Flashen des M4 immer fehl	Kopieren des M4-Programms in den Boot-Ordner des UDOO Neo und anschließender Reboot des Gesamtsystems
Kommunikation hängt sich ab einer gewissen empfangenen Datenmenge auf	Vor dem Start der eigentlichen Anwendung muss der RPMsg-Kanal über die Konsole (minicom-Anweisung) einmalig geöffnet und wieder geschlossen werden
Cortex-A9-Seite (UDOObuntu und Qt)	
Nach kurzer Zeit ist der gesamte Speicher auf der SD-Karte belegt (ca. 3,1 GB sollten eigentlich frei sein)	Ausschalten des Logging-Moduls des Betriebssystems, da dieses alle über RPMsg empfangenen Nachrichten in Dateien abspeichert
Aufgrund einer fehlerhaften Qt-Konfiguration können viele Methoden der Klasse QString nicht verwendet werden	String-Verarbeitung erfolgt teilweise über gewöhnliche char-Arrays

A.5 Inhalt der CD-ROM

- **Ordner \Code**

  Im Unterordner „UDOO Neo M4“ befindet sich das Eclipse-Projekt für die Programmierung des Cortex-M4 inklusive der verwendeten Quelldateien. Hier befindet sich auch das Modul zur Merkmalsextraktion (featureExtraction).

  Im Unterordner „Qt“ befindet sich das Qt-Projekt für die grafische Oberfläche auf dem Cortex-A9. Darin enthalten sind unter anderem der implementierte DTW-Algorithmus (in der Klasse FeatureSequence) und der mit MemBrain generierte Code des neuronalen Netzes.

- **Ordner \Datenblätter**

  Dieser Ordner enthält das Datenblatt des verwendeten Mikrofonverstärkers und des UDOO Neo Boards.

- **Ordner \Dokumentation**

  Der Ordner enthält eine pdf-Version dieser Arbeit.

- **Ordner \Funktionsdemo**

  Dieser Ordner enthält zwei Videos zur Demonstration der beiden Erkennungsverfahren. Im Video zum DTW-Algorithmus wurden verschiedene Kommandos eingesprochen, die bei erfolgreicher Erkennung über den Broker an einen Client geschickt und dort visualisiert werden. Im Video zum neuronalen Netz wird die Erkennung der vortrainierten Zahlen von eins bis neun demonstriert.