Bachelorarbeit

Sergej Sulz

Sprachkommunikation mit einem Service-Roboter

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sergej Sulz

Sprachkommunikation mit einem Service-Roboter

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.Ing. Andreas Meisel
Zweitgutachter : Prof. Dr. rer. nat. Wolfgang Fohl

Abgegeben am 17. Januar 2011

Sergej Sulz

Thema der Bachelorarbeit

Sprachkommunikation mit einem Service-Roboter

Stichworte

Spracherkennung, HTK, Julius, Simon, Sprecherlokalisierung

Kurzzusammenfassung

Bei dieser Arbeit handelt es sich um die Entwicklung eines Sprachinterfaces für den mobilen Roboter SCITOS G5 an der Hochschule für Angewandte Wissenschaften Hamburg. Ziel ist es so eine Sprachkommunikation mit einem Service-Roboter herzustellen und diesem Aufgaben zu erteilen. Dabei spielt die Spracheingabe sowie auch die Sprachausgabe eine große Rolle. Es werden drei verschiedene Verfahren zur Spracherkennung vorgestellt, wobei die Erkennung mittels Hidden Markov Modell implementiert wird. Das System baut auf der kontinuierlichen Spracherkennungs-Engine *Julius* auf. Zur Modellierung wird *Simon* eingesetzt, da es eine komfortable grafische Oberfläche hat, und das trainieren von Wörtern sehr leicht macht. Zur Steuerung und Dialogführung wird ein Zustandsautomat eingesetzt, der leicht verändert und erweitert werden kann. Die Sprachsynthese erfolgt über das Open Source Tool *Mbrola* und wird auch über den Zustandsautomaten gesteuert. Zum Projektende wurde an dem Beispieldialog KIARA eine Erkennungsrate von 95,91% erreicht.

Sergej Sulz

Title of the paper

Voice communication with a service robot

Keywords

Speech recognition, HTK, Julius, Simon, speaker localisation

Abstract

This thesis introduces the development of a speech interface for the service robot SCITOS G5 at Hochschule für Angewandte Wissenschaften Hamburg. The aim is to establish a voice communication with a service robot where speech recognition and speech synthesis play the major role. There are three methods for speech recognition presented, where Hidden Markov Modell is implemented finally. The system is based on the continuous speech recognition engine *Julius*. *Simon* is used for modeling because it has a comfortable graphical user interface, and the training of words is very easy. To control the machine a stateMachine is used, which can be easily edited and expanded. The speech synthesis is performed by the Open-Source tool *Mbrola* which is also controlled by the state machine. At the end of the demo project KIARA a speech recognition rate of 95,91% was achieved.

Meiner süßen Tochter Melina
und meiner lieben Frau Elvira ...

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich während meiner Bachelorarbeit unterstützt haben. Insbesondere bei Prof. Dr.Ing. Andreas Meisel und Prof. Dr. rer. nat. Wolfgang Fohl für die gute Betreuung und das Equipment.

Vielen Dank an Helmut Schimkowski, Dmitri Hammernik und Margarete Daiker für das Korrekturlesen.

Des Weiteren einen herzlichen Dank an meine Tante, die mir im Vorfeld eine Einstiegsmöglichkeit in einem Betrieb ermöglicht hat.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einführung

Die Spracherkennung ist ein fester Bestandteil von Sprachkommunikation mit autonomen Systemen und Maschinen. Dabei ist es das Ziel die Sprachsignale so gut wie möglich in Text umzusetzen und zu verarbeiten. Die Verarbeitung kann unter anderem in folgende Anwendungsgebiete unterteilt werden:

- Büro
  Rechtsanwaltskanzlei, Informationsabfrage
- Telefon-/Auskunftssysteme
  Deutsche Bahn, Kino
- Steuerung von Maschinen und autonomer Systeme
- Medizin
  SpeechMagic, Dragon Medical [Nuaa]
- Behindertenhilfe
  Simon listens Projekt [Sl]

1.1. Motivation und Aufgabenstellung

In dieser Arbeit wird ein Sprachinterface für den mobilen Serviceroboter SCITOS G5 (Abbildung 1.1 auf Seite 14) an der HAW Hamburg implementiert. Der SCITOS G5 wurde von der Firma MetraLabs entwickelt und bietet eine mobile Plattform für Forschungszwecke. Das Gesamtsystem soll eine sprecherunabhängige Sprachkommunikation mit dem Roboter ermöglichen, was die Spracherkennung und die Sprachausgabe miteinschließt. Einem Sprecher soll es so möglich sein, Befehle zu erteilen und Statusmeldungen abzurufen. Aufgrund der Mobilität des Systems muss es robust sein und unter Raumbedingungen zuverlässig arbeiten können. Deswegen werden verschiedene Ansatzmölichkeiten betrachtet, ausgewertet und die geeignetste implementiert.

Abbildung 1.1.: Der SCITOS G5 Service-Roboter an der HAW Hamburg

1.2. Aufbau und Kapitelübersicht

In Kapitel 2 gehe ich zuerst auf die Merkmale und Erzeugung von Sprachsignalen ein. Die dann vorgestellten Verfahren nutzen diese Erkenntnisse um effizient Sprache zu erkennen. Zum Kapitelende wird die Vielfalt der Anwendungsarten und Spracherkennungssysteme vorgestellt und erklärt. Weiterhin wird in Kapitel 3 die Software vorgestellt die zur Spracherkennung und Sprachsynthese dient. Dabei werden kommerzielle und Open Source Projekte betrachtet. Kapitel 4 befasst sich mit der Sprecherlokalisierung. Sie ist für die Störreduzierung sehr wichtig ist. Die Implementierung erfolgt in Kapitel 5. Hier wird die eingesetzte Hardware beschrieben sowie die Vorrausetzungen für das eingesetzte System erläutert. Eine ausführliche Installationsanleitung für die ausgewählte Software ist hier auch zu finden. Zum Schluss des Kapitels wird auf die Programmierung der einzelnen Komponenten eingegangen. Eine Gebrauchsanweisung für den Einsatz und das Trainieren des Spracherkennungssystems ist in Kapitel 6 vorhanden.

2. Spracherkennung

Die Spracherkennung ist der Kern des zu entwickelnden Sprachinterfaces. In diesem Kapitel werden verschiedene Techniken und Verfahren der Spracherkennnung betrachtet und untersucht, welche davon die geeignetste für die Problemstellung dieser Arbeit ist. Mit den vorgestellten Techniken lassen sich viele verschiedene Anwendungen realisieren.

2.1. Einleitung

Beim Erzeugen von Sprache wird die aus der Lunge ausströmende Luft durch die Stimmbänder in Schwingung versetzt. Die Grundfrequenz der Schwingungen liegt bei Männern im Durchschnitt bei 120 Hz und bei Frauen bei 220 Hz. Im Vokaltrakt (Rachen, Mund und Nasenraum) wird der Luftstrom zusätzlich klanglich verändert. [vgl. PK08, S. 13]

Die dadurch entstehenden Laute lassen sich in folgende Klassen unterteilen:

Vokale

Die ausströmende Luft passiert den Vokaltrakt recht ungehindert.
Es entstehen Selbstlaute: A, E, I, O, U

Konsonanten

Die ausströmende Luft wird im Vokaltrakt behindert oder für eine gewisse Zeit ganz gestoppt. Das deutsche Alphabet enthält folgende Konsonanten:

B, C, D, F, G, H, J, K, L, M, N, P, Q, R, S, T, V, W, X, Z

In der Linguistik werden die Laute einer Sprache als Phone bezeichnet. Diese werden in Lautschrift mit speziellen Symbolen dargestellt, die zum Beispiel in einem Wörterbuch zu finden sind. Mehrere Phone schließen sich zu einem Phonem zusammen und ermöglichen das Wort kontextabhängig richtig zu lesen und auszusprechen. Tabelle 2.1 zeigt als Beispiel das Wort *Bahn* und *Bann* [vgl. Eul06, S. 5-10]:

Wort	**Lautsprache**
Bahn	ba:n
Bann	ban

Tabelle 2.1.: Beispiele für Lautsprache

2.2. Merkmalsextraktion

Um Sprache so effizient wie nur möglich zu erkennen, werden dementsprechend auch Merkmale benötigt, die das ermöglichen. Nach näherer Betrachtung eines Sprachsignals kann man feststellen, dass es aus mehreren Blöcken mit unterschiedlicher Länge aber gleichen Merkmalen besteht. Die Berechnung der Merkmale kann entweder auf das ganze Sprachsignal angewendet werden oder sinnvollerweise aufgeteilt auf Blöcke. Die optimale Blockgröße liegt zwischen 15 und 20 ms. Nach der Berechnung werden die Blockgrenzen um 5 ms verschoben. Derartige Verarbeitung lässt das Signal als eine Folge von Blöcken mit gleichbleibenden Merkmalen ansehen. [vgl. Eul06, S. 30-32]:

2.2.1. DFT

Eine Möglichkeit Merkmale aus einem Sprachsignal zu gewinnen ist die Darstellung des Frequenzspektrums durch die Diskrete Fourier-Transformation. Die DFT ist eine Anpassung der Fourier-Transformation für analoge Signale und ist wie folgt definiert:

$$X_k = \sum_{n=0}^{N-1} x_{(n)} e^{-j\frac{2\pi}{N}kn}; k = 0, 1, ..., N-1 \tag{2.1}$$

[vgl. Wen05, S. 155]

In Abbildung 2.1 auf Seite 17 wird zufällig ein Signalausschnitt von 25 ms der Äußerung *„Hallo“* gewählt. Abgebildet sind der Zeitverlauf des Signals und das dazugehörige durch die DFT berechnete Frequenzspektrum.

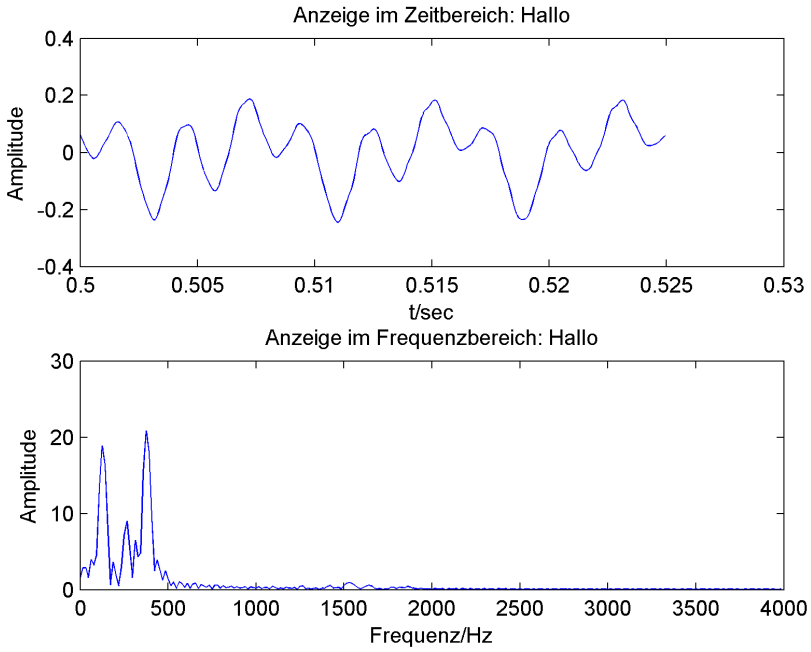

Abbildung 2.1.: Signalverlauf und Frequenzspektrum der Äußerung *„Hallo“*

Damit lassen sich vorkommende Frequenzen in einem Signalfenster berechnen und darstellen. Diese Frequenzen können nun als Merkmal eines Sprachblocks berücksichtigt werden. Für die reale Anwendung geht man ein Schritt weiter und berechnet aus dem Frequenzspektrum das DFT-Cepstrum. Dazu wird der Logarithmus aus dem Spektrum berechnet und erneut mit der Fouriere-Transformation transformiert. Dadurch werden aus multiplikativen Anteilen im Spektrum additive Anteile im Cepstrum. Das hat den Vorteil, dass die störenden Anteile des Signals, welches zum Beispiel ein Mikrofon oder Telefonleitung passiert hat, additiv vorliegen und somit leichter entfernt werden können [vgl. Eul06, S. 43-47].

2.2.2. MFCC

In der Spracherkennung wird anstatt des DFT-Cepstrums auch das Mel-Cepstrum (Mel Frequency Cepstral Coefficients) verwendet. Dabei wird das Mel-Cepstrum nicht wie bei dem DFT-Cepstrum aus dem Frequenzspektrum berechnet, sondern anhand der Mel-Filterdatenbank. Die Mel-Skala (Abbildung 2.2 auf Seite 18) verläuft im Gegensatz zum Frequenzbereich nicht linear und beschreibt die Tonhöhe, die der Mensch wahrnimmt.

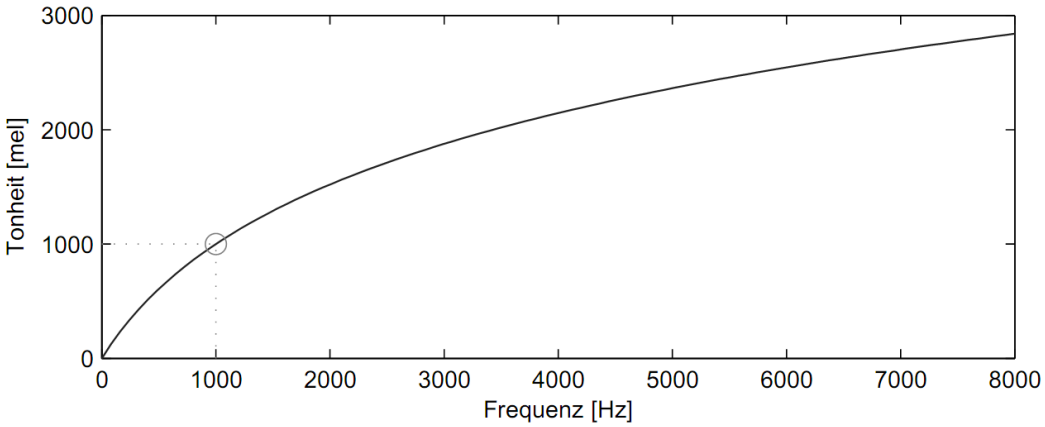

Abbildung 2.2.: Definition der Mel-Skala. Referenzpunkt sind die 1000 Hz, die der Tonheit 1000 mel entsprechen

Die Skalentransformation ist wie folgt definiert:

$$h(f) = 2595 * log_{10}(1 + \frac{f}{700Hz}) \tag{2.2}$$

[vgl. PK08, S. 94-95]

2.3. Verfahren zur Spracherkennung

2.3.1. Kreuzkorrelation

In einem Studienprojekt an der FH Flensburg war es die Aufgabe festzustellen, ob eine ausreichend gute Spracherkennung mit der Kreuzkorrelation zu realisieren ist. Die Kreuzkorellation ist eine mathematische Funktion zum Vergleich von Signalen, die auch zeitlich versetzt sein können. Seien die zwei Signale $X_{1(m)}$ und $X_{2(m)}$, so lautet die mathematische Definition:

$$k_{12}(n) = \sum_{m=-\infty}^{\infty} X_{1(m)} * X_{2(m-n)}; n = -\infty, \ldots, \infty \tag{2.3}$$

[vgl. Wen05, S. 187]

Für die einfache Erkennung von Signalen werden zuerst Vokale auf ihre Frequenzspektren und Eigenschaften untersucht. Bezogen auf einen Sprecher, weisen Vokale ein nahezu kon-

stantes Frequenzspektrum auf (Abbildung 2.3). Die Idee ist nun dabei, dass zuerst Referenzsignale der Vokale aufgenommen werden.

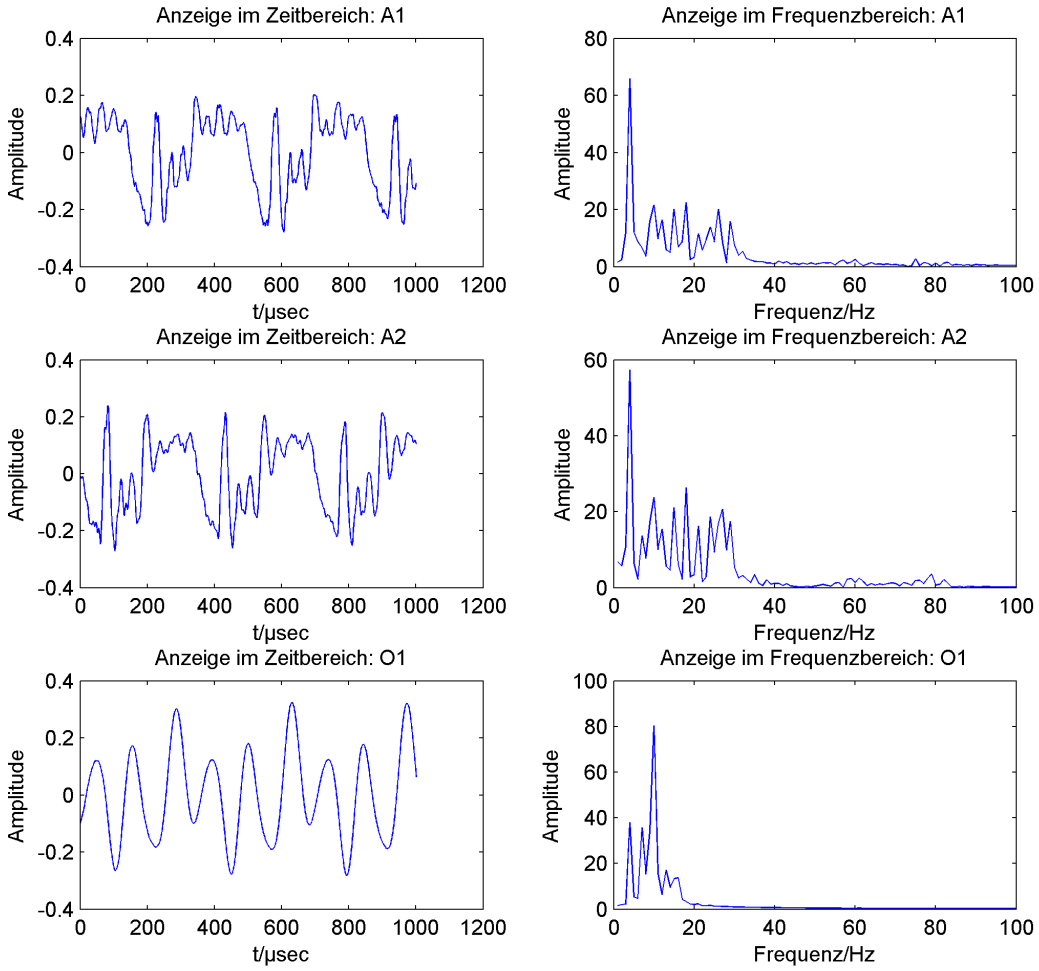

Abbildung 2.3.: Signalverlauf und Frequenzspektrum der aufgenommenen Vokale A und O

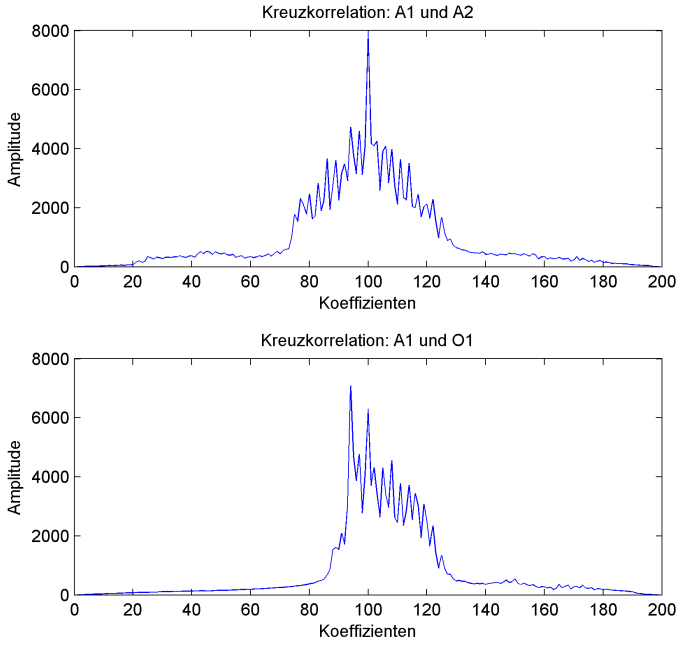

Abbildung 2.4.: Korrelationsvergleich zwischen den Vokalen A-A und A-O

Im zweiten Schritt wird das Frequenzspektrum des Eingangssignals mit dem von mehreren Referenzsignalen verglichen. Dabei entstehen Muster, wie in Abbildung 2.4 zu sehen ist. Des Weiteren tendiert die Korrelationsabbildung von A1 und A2 zu einem Dreiecksmuster. Anhand dem Verhalten des Musters kann man sagen, dass die Signale A1 und A2 mehr zueinander passen, als A1 und O1.

Probleme tauchten auf wenn ganze Worte verglichen wurden. So musste das gesprochene Wort fast exakt so schnell gesprochen sein wie das zuvor aufgenommene Referenzwort. Da dieses System Sprecherabhängig ist, muss der Vergleich auch mit demselben Sprecher durchgeführt werden. Laute wie *TZ* werden schlecht wahrgenommen, da sie in Ihrer Amplitude klein sind und kein eindeutiges Frequenzverhalten aufweisen. Die Kreuzkorrelation ist zwar geeignet Vokale zu vergleichen aber nicht für Ganzwortsignale. Folglich reicht die Kreuzkorrelation nicht aus um Sprache zu erkennen. Das Projekt gab einen sehr guten Einstieg in die Signalverarbeitung und motivierte zum weiter forschen.

Vorteile

- Einfach zu implementieren und zu verstehen

Nachteile

- Beschränkt auf Vokale

- Probleme zwischen E und O
- keine richtige Spracherkennung

2.3.2. Dynamic Time Warp - DTW

Der einfachste Weg eine Spracherkennung zu realisieren ist es zwei Sprachsignale zu vergleichen. Dabei werden generell Referenzsignale aufgenommen, die Merkmale und Muster extrahiert und abgespeichert. Dann werden die Merkmale des unbekannten Signals mit den der Referenzsignale verglichen. Der Vergleich zeitlich versetzter Signale ist allerdings nicht so einfach, so wie bei dem Vergleich mit der Kreuzkorrelation. Hier kommt DTW (Dynamische Zeitverzerrung) zum Einsatz. Wie der Name schon sagt, wird zuerst das aufgenommene Sprachsignal zeitlich verzerrt, sodass es die gleiche Länge wie das Referenzsignal hat. Dann werden jeweils die Differenzen der Merkmale berechnet und die kleinste Distanz als erkanntes Referenzmuster ausgegeben. Die Berechnung des optimalen Pfades basiert auf der dynamischen Programmierung. Der DTW-Algorithmus ist relativ einfach und lässt sich kostengünstig implementieren. Auch die Erkennungsrate liegt bei bis zu 100 % bei einem Wortschatz von ca. 1000 Wörtern.

Die Schwäche liegt im größer werdenden Vokabular und der Sprecherabhängigkeit. Je mehr Wörter desto langsamer wird das Erkennungssystem, dafür aber sprecherunabhängiger. Auch kann mit DTW keine kontinuierliche Sprache erkannt werden. Das Training auf Äußerungen mit mehreren Wörtern ist allerdings möglich. Aufgrund dieser Schwächen kommt dieses System nicht zum Einsatz. [vgl. PK08, S. 312-316] [vgl. Eul06, S. 55-66]

2.3.3. Hidden Markov Modell - HMM

Das Hidden Markov Modell ist ein stochastisches Modell und beschreibt zwei gekoppelte Zufallsprozesse. Dabei ist der erste Zufallsprozess durch Zustände (Markov-Kette) mit Wahrscheinlichkeitsgewichten beschrieben. Der Prozess ist hier von außen nicht sichtbar (verborgen: *hidden*). Ein zweiter Prozess erzeugt Übergänge mit Gewichten nach einer Wahrscheinlichkeitsverteilung. Dieses Model passt bisher am besten zum Modellieren von Sprachsignalen. Dabei wird das Sprachsignal als eine Abfolge von Zuständen angesehen. Ein Zustand stellt einen Laut der Sprache dar. In der Spracherkennung wird von jedem Wort des Vokabulars eine statistische Beschreibung angelegt. Bei der Erkennung wird dann das Wort ausgegeben, welches am besten zu der statistischen Beschreibung passt, oder besser gesagt die höchste Wahrscheinlichkeit hat.

Zum besseren Verstehen werden die Schritte der Spracherkennung am Wort „*Hallo*“ mittels HMM betrachtet. Nach der Sprachaufnahme wird das Audiosignal in Blöcke zerlegt und die einzelnen Merkmale (hier MFCC) berechnet. Die berechneten Merkmale werden in einem Merkmalsvektor zusammengefasst (Abbildung 2.5). Die Merkmalsvektoren können jetzt auf

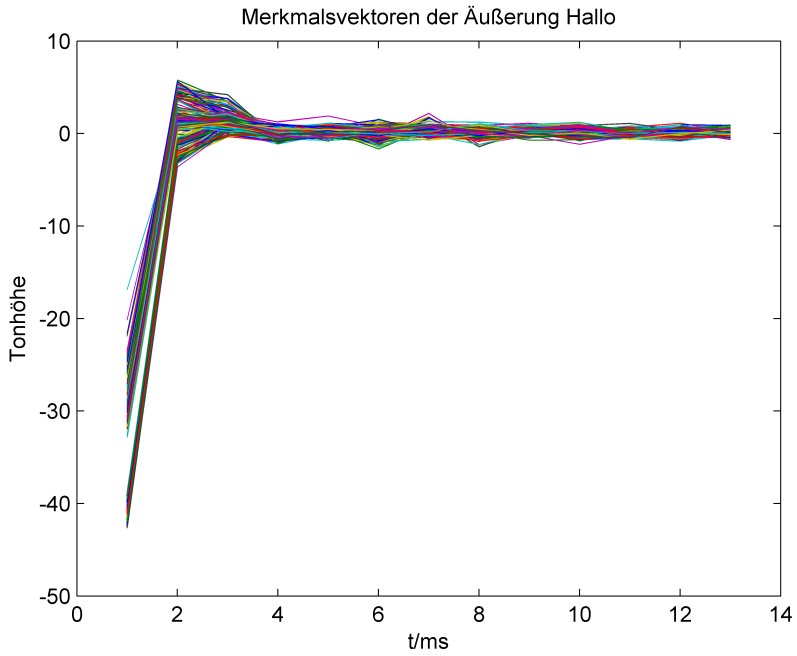

Abbildung 2.5.: Merkmalsvektoren des aufgenommenen Wortes „Hallo“

die linguistische Ebene abgebildet werden. Dabei wird jeder Merkmalsvektor einem Phonem zugeordnet und mit einer statistischen Beobachtung versehen. Danach erfolgt die Gewichtsverteilung der Wahrscheinlichkeiten welche die Merkmalsvektoren wie Zustände miteinander verbindet. Mit jeder weiteren Sprachaufnahme wird das Modell trainiert indem die Merkmalsvektoren und Wahrscheinlichkeiten angepasst werden. Wie in Abbildung 2.6 auf Seite 23 zu sehen ist, kann ein Wort somit als ein endlicher Zustandsautomat angesehen werden. Um den zeitlichen Ablauf einer Aufnahme wiederzugeben, wird der Zustandsautomat von links nach rechts abgearbeitet, sodass die einzelnen Zustände keine Übergänge zu den Vorgängerzuständen haben dürfen. Bindet man den Erkennvorgang an eine Grammatik, so kann man bei Wortketten noch bessere Erkennungsergebnise erzielen.

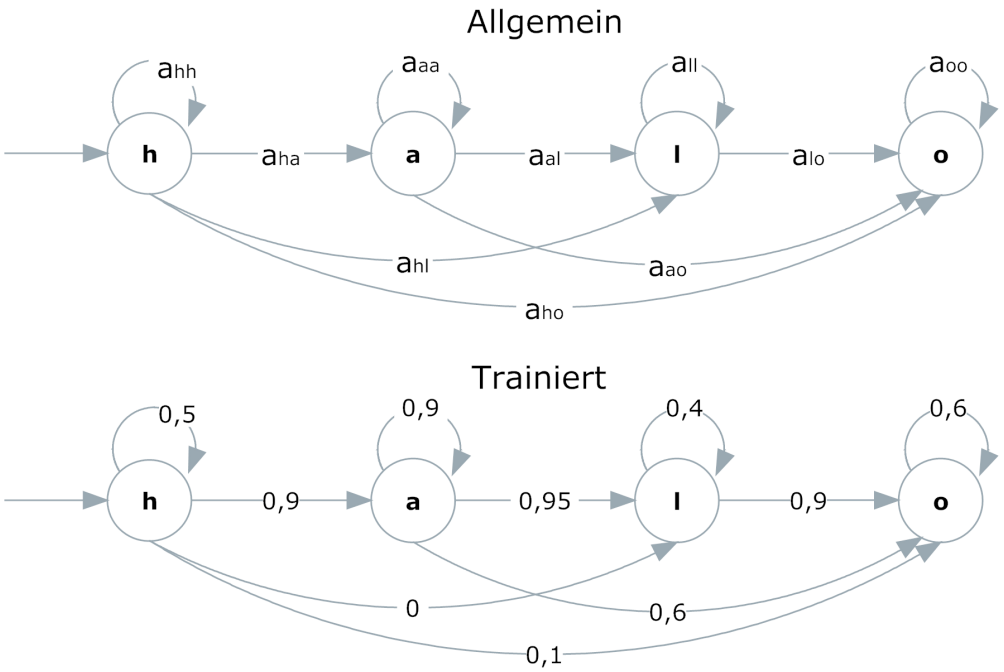

Abbildung 2.6.: Beispiel: Hidden Markov Modell für das Wort *„Hallo“*

Die ideale Wahrscheinlichkeit bei der das Wort „*Hallo*“ erkannt wird kann nach der Formel 2.4 berechnet werden.

$$P(a_{ha}|a_{al}|a_{lo}) = a_{ha} * a_{al} * a_{lo} = 0,9 * 0,95 * 0,9 = 0,77 \tag{2.4}$$

Ein weiterer Vorteil von Hidden Markov Modellen ist, dass der Zustandsautomat auch die zeitliche Variabilität eines Sprachsignals berücksichtigt. So können auch anders ausgesprochene Versionen des Wortes „*Hallo*“ erkannt werden: $\rightarrow$ *haalo*, *halooo*. Dementsprechend verhält sich auch die Wahrscheinlichkeit der jeweiligen Äußerungen.

2.4. Anwendungen

Die Spracherkennung kann man in verschiedene Spracherkennungssysteme einteilen. Tabelle 2.2 zeigt die verschiedenen Anwendungen und dazu das jeweilige Spracherkennungssystem.

System	Anwendung
Einzelworterkennung	Welches Wort wurde gesagt?
Keyword-Spotting	Wurde ein bestimmtes Kommando in einer Äußerung erwähnt?
Kontinuierliche Sprache	Ganze Sätze die fließend gesprochen wurden.
Sprechererkennung	Wer hat die Äußerung gesprochen? Dazu gehört auch die Verifikation.
Sprachenidentifikation	In welcher Sprache wurde gesprochen?

Tabelle 2.2.: Einteilung der Spracherkennungssysteme nach ihren Anwendungen

Die Kernaufgabe ist es eine Äußerung richtig in Worte zu zerlegen und zu verarbeiten. Oft ist es nicht nötig ganze Sätze richtig zu erkennen, so setzt man auf die **Einzelworterkennung**. Dabei geht das System davon aus, dass eine Äußerung genau ein isoliertes Wort ist. Um einzelne Wörter zu erkennen muss das System dementsprechend auch mit einzelnen Wörtern trainiert werden.

Beim **Keyword-Spotting** kann das Schlüsselwort allerdings in einer fließenden Äußerung auftreten. Solche Systeme werden überwiegend zur Kommando-Erkennung eingesetzt.

Die **kontinuierliche Spracherkennung** geht ein Schritt weiter und ermöglicht das Erkennen von ganzen Sätzen die auch fließend gesprochen wurden. Die Art von Spracherkennungssystem wird bei Diktiersoftware wie *Dragon Naturally Speaking* eingesetzt. Hier reicht das Einzelworttraining nicht aus, es müssen auch ganze Sätze trainiert werden.

Zur Verifizierung von Sicherheitsbereichen kann die **Sprechererkennung** eingesetzt werden. Der Sprecher muss sich durch ein vordefiniertes Passwort ausweisen oder das System erkennt den Sprecher anhand von zufällig gewählten Äußerungen.

Sprachenidentifikation ist vorwiegend in Übersetzungsprogrammen zu finden.

Des Weiteren werden Spracherkennungssysteme an der Sprecherabhängigkeit differenziert:

Sprecherabhängig
Jeder Sprecher muss die Äußerungen trainieren.

Sprecherunabhängig
Diese Systeme erkennen Äußerungen von beliebigen Sprechern. Dazu müssen aber auch die Äußerungen aufwändig von mehreren tausend Sprechen trainiert werden.

Mit den beschriebenen Spracherkennungssystemen lassen sich eine Vielfalt von Anwendungen realisieren, insbesondere:

- Gerätesteuerung
  Stereoanlage, Klimaanlage, Handy, PDA, Fernbedienung
- Diktiersysteme
  medizinische Befunde (Diagnose, Radiologie), Rechtsanwaltskanzlei
- Sprachdialogsysteme
  Telefonische Auskunftssysteme, telef. Bestellsysteme, Gewinnspiele

[vgl. PK08, S. 289-292] [vgl. Eul06, S. 15-17]

3. Software

3.1. Spracherkennung

3.1.1. Kommerziell

3.1.1.1. Nuance Communications

Die Hersteller Nuance bietet das sehr bekannte und erfolgreiche Spracherkennungsprogram *Dragon NaturallySpeaking* an. Testberichten [Tes] zufolge ist die Software durchschnittlich gut bewertet und laut Hersteller erreicht *Dragon NaturallySpeaking* eine Erkennungsrate von bis zu 99%. Durch den Zukauf der Firma *ScanSoft* und *ViaVoice* von IBM ist Nuance derzeitig Marktführer auf dem Gebiet der Spracherkennungssoftware. Weiterhin entwickelt Nuance Produkte die auf verschiedene Zielgruppen zugeschnitten und optimiert sind:

Nuance Speech Solutions

Optimierte Lösungen für

- Telekommunikation
- Auskunftsdienste
- Mobile Geräte
- Automobilbereich

Nuance Healthcare Solutions

- Dragon Medical

  Möglichkeit für Ärzte echtzeit in medizinischen Informationssystemen zu navigieren und Befunde zu diktieren.

- SpeechMagic Solution Builder

  Dient zur Erstellung professioneller medizinischer Dokumentation.

Dragon NaturallySpeaking Solution

Erstellung von Dokumenten, PC-Navigation und Suche im Internet für Heimanwender. Professionelle Anwender und Unternehmen profitieren von eigen erstellten Makros, mit denen sich Geschäftsprozesse optimieren lassen. Die Dragon NaturallySpeaking Legal Version ist auf die Rechtsbranche optimiert.

Alle genannten Produkte von Nuance setzen das Windows Betriebssystem voraus. Eine Installation unter Linux ist daher nicht möglich. Das eingesetzte System sollte mindestens 1 GB Arbeitsspeicher aufweisen. Die Installation auf der Festplatte hat einen Umfang von rund 3 GB. [Nuaa]

3.1.1.2. Voice Pro

Nicht ganz unbekannt ist auch das Produkt *Voice Pro* von Linguatec und Microsoft. Auch hier soll die Erkennungsgenauigkeit bei bis zu 99% liegen. Möglich wird dies durch den Einsatz von linguistischer Intelligenz *SpeechCorrect*. Anhand des Kontexts wird bei gleichklingenden Wörtern das richtige Wort ausgewählt. Damit wird das Diktieren von „Stephan *Meier*“ und „*Meyers* Lexikon“ kein Problem mehr. Wie Nuance entwickelt auch Linguatec *Voice Pro* für verschiedene Einsatzbereiche:

Voice Pro

Standard
Einsatz im Heimbereich.

Premium
Zugeschnitten für den beruflichen Alltag.

Premium Wireless
Die professionelle Spracherkennung mit Bluetooth-Headset.

Legal
Professionelle Spracherkennung für Juristen mit 12 Fachgebieten.

Medical
Entwickelt für den Einsatz in Arztpraxen, Kliniken und anderen medizinischen Einrichtungen.

Das bei den Produkten eingesetzte Sprachwörterbuch beinhaltet über 1 Million Wortformen die sich in den Bereichen EDV, Sport, Wirtschaft, Wissenschaft und Technik wiederfinden. Eine weitere Besonderheit ist, dass die Bedienung von Outlook und die Suche im Internet sehr leistungsfähig sind. Die Unterstützung der Betriebssysteme ist hingegen sehr mager. Ab

Voice Pro 12 werden nur Microsoft Windows Vista und Windows 7 unterstützt. Der Arbeitsspeicher sollte nicht weniger als 1 GB aufweisen. Das Installationsvolumen auf der Festplatte beträgt auch ca. 1 GB. [Lin]

3.1.2. Open-Source

3.1.2.1. CMU Sphinx

Die Spracherkennungs-Engine Sphinx wurde an der Carnegie Mellon University entwickelt. Die aktuelle Version ist Sphinx 4 und wurde nach Version 3 komplett neu überarbeitet und in JAVA geschrieben. Die Entwicklung wird von SUN Microsystems und Mitsubishi unterstützt. Basierend auf HMM arbeitet Sphinx sprecherunabhängig und ermöglicht die Spracherkennung mit einem großen Vokabular. Da Sphinx 4 sehr modular aufgebaut ist, lässt es sich sehr einfach durch Plug-Ins erweitern. Ein großes Problem stellt der Zugriff auf die Ressourcen dar. Zur Ausführung benötigt Sphinx 4 eine Java Virtual Machine mit etwa 1 GB Speicher. Außerdem sind die Algorithmen sehr komplex und langsam. [WLK+04]

Wie alle Spracherkenner benötigt auch Sphinx ein Sprachmodell. Deshalb stellen die Entwickler das Programm *Sphinxtrain* zur Verfügung, mit dem man sein Sprachmodel zum eigenen Vokabular anlegen kann. Das modellieren einer Sprache ist mit *Sphinxtrain* wesentlich einfacher als das modellieren mit dem HTK Toolkit. [Car]

3.1.2.2. HTK Toolkit

Mit dem Hidden Markov Toolkit kann man aus Sprachsignalen Hidden Markov Modelle erzeugen. Die Entwicklung wurde zuerst an der Cambridge Universität angefangen, später kommerziell von der Firma *entropics* vertrieben und aktuell steht es wieder zur freien Verfügung. Das Toolkit besteht aus einer Ansammlung von einzelnen Programmen die über Konfigurationsdateien und Startparameter gesteuert werden können. Abbildung 3.1 auf Seite 29 zeigt die Toolkit Architektur mit den dazugehörigen Interface Ein- und Ausgaben.

Als erstes müssen die Sprachdaten vorbereitet werden. Mit *HSLab* können Sprachsignale aufgenommen und bearbeitet werden. Liegen Sprachsignale bereits als Wave-Datei vor können diese mit *HCopy* direkt in das MFCC Format konvertiert werden. Nachdem die Sprachsignale, das Wörterbuch und die Grammatik vorbereitet sind kann das Training mittels *HInit* und *HRest* beginnen, wobei ein Hidden Markov Modell als Ausgabe in eine HMM-Datei geschrieben wird. Die eigentliche Spracherkennung übernimmt das Tool *HVite*. Der allgemeine Ablauf zur Benutzung des HTK-Toolkits ist in Abbildung 3.2 auf Seite 29 dargestellt.

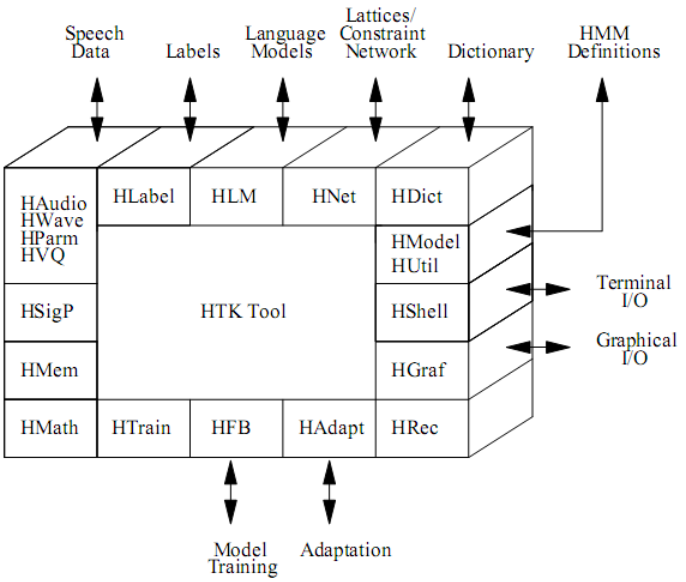

Abbildung 3.1.: HTK Software Architektur

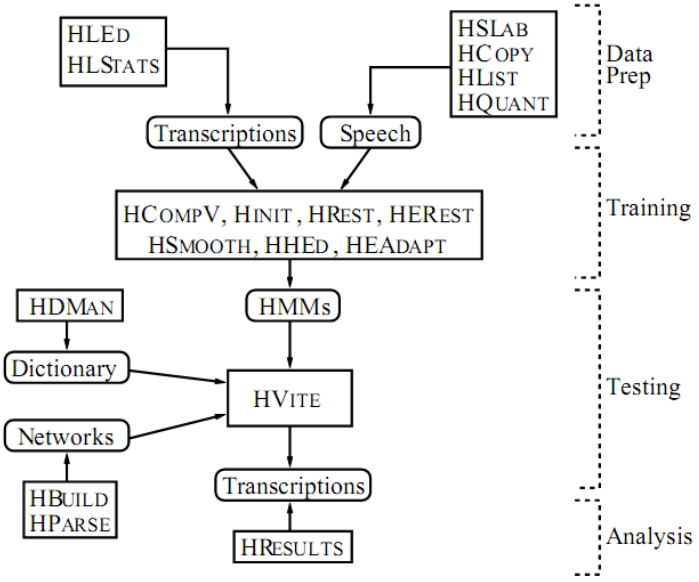

Abbildung 3.2.: HTK Prozessablauf

Am Anfang erscheint die Handhabung des Tools ganz einfach, doch aufgrund der vielen Tools und Einstellungsmöglichkeiten ist dennoch viel Einarbeitungszeit nötig. Die Vorteile des Toolkits sind hingegen:

- sehr flexibel
- wird von den Entwicklern up-to-date gehalten
- optimiert für Spracherkennung
- sprecherunabhängige Modellierung möglich
- sehr gute Dokumentation
- läuft unter Linux und Windows
- programmiert in C

Mehr Informationen erhalten sie in dem ausführlich beschriebenen HTK-Book [YEK+01] nach einer Registrierung auf der Webseite der Cambridge University [Cam].

3.1.2.3. Julius

Julius ist ein LVCSR (Large-Vocabulary Continuous Speech Recognition) Spracherkenner. Ursprünglich 1997 von *Japanese LVCSR* entwickelt, wird das Projekt zurzeit von *Interactive Speech Technology Consortium (ISTC)* fortgeführt. Julius kann nicht nur Einzelwörter erkennen sondern auch ganze Sätze. Insgesamt beziffert der Entwickler das Vokabular mit 65 535 Wörtern. Die Erkennung verläuft in Echtzeit, wobei die Suche in zwei Durchgängen optimiert wird. Deshalb kann eine Erkennungsrate von über 95% erreicht. Leider können mit Julius keine Sprachmodele erstellt werden, deshalb müssen diese mit externen Tools wie HTK modelliert werden.

Nachteile

- Dokumentation überwiegend auf Japanisch
- Sprachmodellierung über externe Tools

Vorteile

- sehr gute Erkennungsraten
- Vokabular bis 65 535 Wörter
- System ist sprecherunabhängig
- sehr performant (weniger als 32 MB Arbeitsspeicher benötigt)

- Echtzeitverarbeitung
- viele Einstellmöglichkeiten über zentrale Datei
- läuft unter Linux und Windows

Ausführliche Informationen erhalten Sie auf der Webseite von Julius [Nag].

3.1.2.4. Simon listens

Das Projekt *Simon listens* soll behinderten Menschen den Umgang mit dem Computer erleichtern. Der Benutzer kann mit der grafischen Oberfläche ganz bequem sein eigenes Sprachmodell erstellen und trainieren. Nach der Installation von simon ist es wichtig das Hidden Markov Toolkit [Cam] zu installieren, denn die Modellierung der Sprache nimmt simon über das Toolkit vor. Die ausführlich beschriebene Installationsanleitung finden Sie im Kapitel 5. Die eigentliche Spracherkennung läuft über die interne Integration von *Julius*.

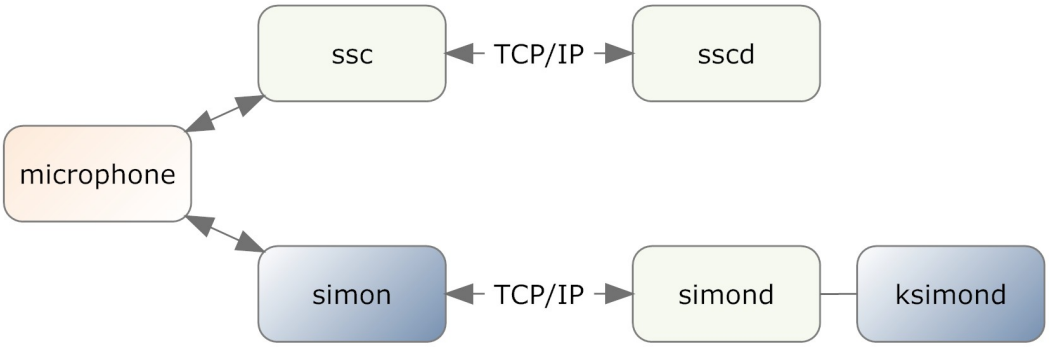

Abbildung 3.3.: Komponentenübersicht und Kommunikationsablauf

Abbildung 3.3 zeigt die Zusammenarbeit und Kommunikation der Simon Komponenten. Die Tools *ssc* und *sscd* ermöglichen das Aufnehmen einer großen Anzahl von Sprachdaten. Die beiden Komponenten *simon* und *simond* bilden eine Server/Client Anwendung. *simond* ist dabei der Erkennungsserver, der mehrere Clients über das Netzwerk bedienen kann. Die grafische Oberfläche *ksimond* erleichtert dem User die Konfiguration des Erkennungsservers *simond* [vgl. Gra09, S. 8]. Der modulare Aufbau von Simon listens verleiht dem Projekt eine hohe Flexibilität, wobei auch Plug-Ins implementiert sind. Damit lassen sich verschiedene Szenarien realisieren. Eine ausführliche Dokumentation und Installationsanleitung finden Sie auf der Homepage von simon listens [Sl].

3.2. Sprachsynthese

Im Sektor der Sprachsynthese sind wesentlich mehr Projekte zu finden als in der Spracherkennung. Hier gibt es zahlreiche kommerzielle und Open-Source Lösungen die sehr natürlich klingende Sprachausgabe ermöglichen. Eine Liste der bekanntesten Projekte finden Sie auf der Webseite von *synthetic speech* [Bur]. Dort sind auch Demoausgaben zu den jeweiligen Projekten aufgeführt, wobei die Beispiele von dem kommerziellen *Nuance Vocalizer* und *AT&T Natural Voices* besonders in ihrer Natürlichkeit beeindruckt haben. Bei den Open-Source Projekten sind BOSS und Mbrola hervorgestochen.

3.2.1. Kommerziell

3.2.1.1. Nuance Vocalizer

Nuance Communications ist nicht nur in dem Bereich der Spracherkennung spezialisiert sondern auch in der Sprachsynthese. Die Softwarelösung *Nuance Vocalizer* wird für Unternehmen und Mobilanwendungen angeboten, wobei über 41 Sprachen inklusive Deutsch unterstützt werden. Seit Jahren nutzt die Deutsche Telekom die TTS-Lösung für ihre Dienste der Telefonauskunft. Bei der Swisscom hat Nuance eine individuelle Lösung implementiert. Dabei können Anrufer per Sprache Zugriff auf Rufnummern und Adressen von Unternehmen und Privathaushalten bekommen. Deshalb legt Nuance besonderen Wert auf die hohe Qualität der Sprachsynthese von Namen und Adressen [Nuab].

3.2.1.2. AT&T Natural Voices

Die Entwicklung von AT&T Natural Voices begann als ein Forschungsprojekt an der AT&T Labs Research. Das fertige Produkt zeigt erstaunliche Ergebnisse. Die Sprachausgabe klingt sehr natürlich und ist kaum von einem realen Sprecher unterscheidbar. AT&T Natural Voices ist in zwei Versionen erhältlich: *Desktop* und *Server*. *Desktop* kostet 295$ und bietet einen SDK (Software Development Kit) für Entwickler. *Server* ist für 995$ zu haben und ermöglicht das Entwickeln von Serveranwendungen mit TTS. Es wird das Betriebssystem Windows und Linux unterstützt. [AT&]

3.2.2. Open-Source

3.2.2.1. BOSS

BOSS (Bonn Open Synthesis System) wird an dem Institut für Kommunikationsforschung und Phonetik entwickelt und steht als Open-Source Projekt frei zur Verfügung. Das Projekt basiert auf der Client/Server Architektur. Dabei wird die eigentliche Sprachsynthese auf dem Server ausgeführt und zum Client transferiert. Diese Arbeitsweise ermöglicht die Ausführung auf Systemen mit wenig Ressourcen. Die Sprachqualität scheint bisher die beste im Bereich Open-Source zu sein. Allerdings ist die Sprecherauswahl klein und auch die Installation nur für erfahrene Anwender, da BOSS für Linux konzipiert wurde. [BWA+05]

3.2.2.2. Mbrola

Das Ziel des TCTS Lab der Faculté Polytechnique de Mons (Belgien) war es ein Projekt zu erwecken, welches die Forschung auf dem Bereich der Sprachsynthese steigern sollte. Mittelpunkt ist das Projekt Mbrola, ein TTS-System auf Basis der Diphone Verkettung. Die nicht-kommerzielle Nutzung ist kostenlos. Zwar ist die Qualität der Sprachausgabe nicht so gut wie bei dem BOSS Projekt, aber dafür ist die Auswahl an Sprechern grösser und die Installation des Systems einfacher. Zurzeit sind 35 Sprachen implementiert, darunter auch die Deutsche Sprache mit 7 Sprechern. Zu beachten ist, dass Mbrola kein richtiger Text-to-Speech Synthesizer ist, sondern als Input Phoneme benötigt, die von einem externen Programm erzeugt werden [TCT]. Die Installationsanleitung finden Sie im Kapitel 5.

4. Sprecherlokalisierung

Bei der Realisierung der Spracherkennung für mobile Systeme muss man auch auf die Raumbedingungen eingehen. Dabei können Störgeräusche, wie das schließen einer Tür oder Klimaanlagen die Spracherkennung erheblich verschlechtern. Auch das Gespräch von mehreren Person in einem Raum führt zur erheblichen Beeinträchtigung des Gesamtsystems. Um diese variablen Einflüsse zu minimieren kam die Idee auf, Signale nach ihrer Quelle zu orten. So könnte man das Spracherkennungsfenster so wählen, dass nur Signale weiterverarbeitet werden, die in Ihrer Quelle den Ursprung vor dem mobilen Service-Roboter haben. In der Abbildung 4.1 sehen Sie die grünen Erfassungsbereiche für die Spracherkennung. Sprecher außerhalb des Bereichs werden ignoriert. Da zwei Mikrofone eingesetzt werden, kann man nicht feststellen, ob das Signal vor oder hinter dem Roboter erzeugt worden ist.

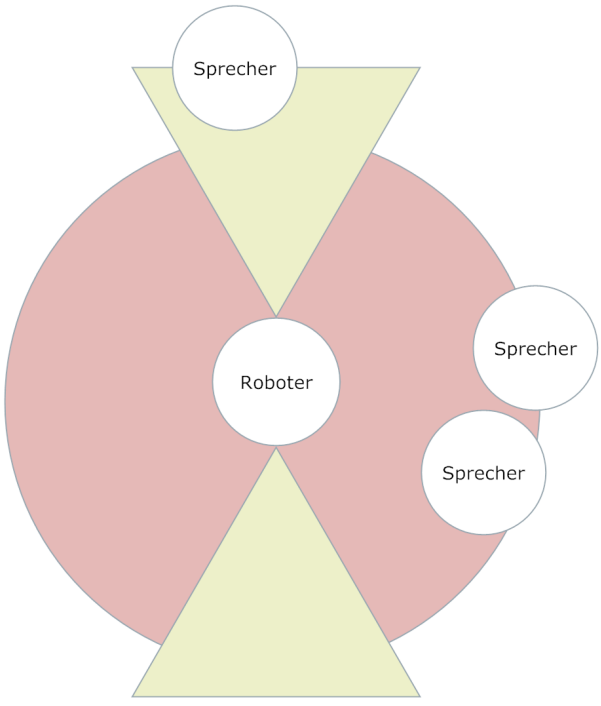

Abbildung 4.1.: Erfassungsbereich für die Spracherkennung. Sprecher im grünen Bereich wird berücksichtigt

Natürlich kann es dann aber immer noch vorkommen, dass sich grade mehrere Personen vor dem Service-Roboter unterhalten. Um dem entgegenzuwirken, wird der Service-Roboter mit einem Dialogsystem ausgestattet, welches per Sprachsynthese nachfragt, ob er gemeint ist. Bei einer Negativen Antwort dreht sich das System so, dass das Gespräch nun außerhalb des Erkennungsfensters stattfindet.

4.1. Eingesetzte Hardware

Für die Lokalisierung benötigt man mindestens zwei Mikrofone mit guter Raumcharakteristik und Empfindlichkeit. Zur Verfügung standen Mikrofone des Typs C3000B AKG (Abbildung 4.2 [AKG]) und ein USB Interface Tascam US-122 (Abbildung 4.3 [Tas]). Damit können 2 separate Signale, hier linker und rechter Kanal, aufgenommen werden.

Abbildung 4.2.: Mikrofon C3000B von AKG zur Signalaufnahme für Sprecherlokalisierung

Abbildung 4.3.: Eingesetzte USB-Mischpult US-122 von Tascam

4.2. Kreuzkorrelation

Die Lokalisierung beruht auf der Kreuzkorrelationsmethode. In Kapitel 2 wurde beschrieben, dass mittels der Kreuzkorrelation zwei Signale miteinander verglichen werden können. Nicht nur der Vergleich, sondern auch die Phasenverschiebung kann so berechnet werden. Für ein kleines Experiment werden folgende Definitionen festgehalten:

- Schallgeschwindigkeit c beträgt in Luft bei $20°C$ $343\frac{m}{s}$ [SK06, S. 84]
- Mikrofonabstand d liegt bei $30cm$

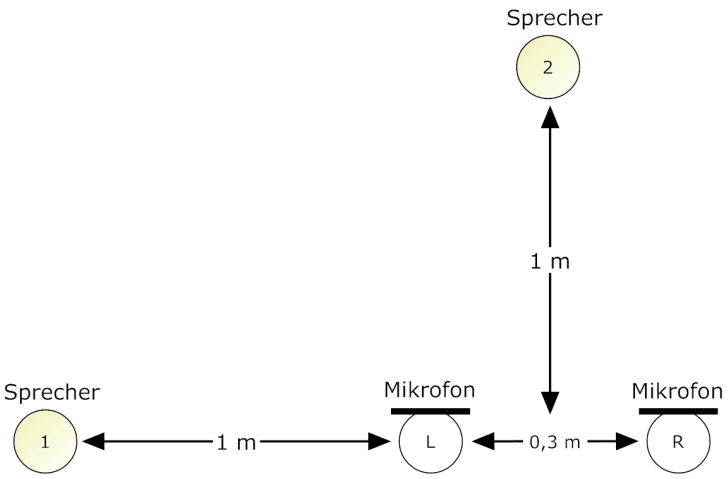

Abbildung 4.4.: Versuchsaufbau und Positionierung der Sprecher

In Abbildung 4.4 ist der Versuchsaufbau dargestellt. Wenn jetzt der Sprecher 1 ein Sprachsignal erzeugt, tritt an den aufgezeichneten Signalen der beiden Mikrofone eine maximale Phasenverschiebung t_{vmax} auf. Diese lässt sich mit einem Dreisatz berechnen:

$$\frac{t_{vmax}}{1s} = \frac{d}{c} \Rightarrow t_{vmax} = \frac{0,3m}{343m} = 874,6\mu s \tag{4.1}$$

Schaut man sich nun den Verlauf und die Korrelation der original aufgezeichneten Signale an (Abbildung 4.5 auf Seite 37), so stellt man fest, dass der höchste Korrelationskoeffizient bei 163 liegt. Würde Sprecher 2 sprechen, läge der höchste Korrelationskoeffizient bei 200, was eine Phasengleichheit der Signale bedeutet. Also beträgt die zeitliche Verschiebung der Signale hier $Samples_v = 200 - 163 = 37 Samples$. Für die Umrechnung von Samples in eine zeitliche Angabe braucht man die Abtastfrequenz, die bei der Aufzeichnung bei $f_a = 44kHz$ lag.

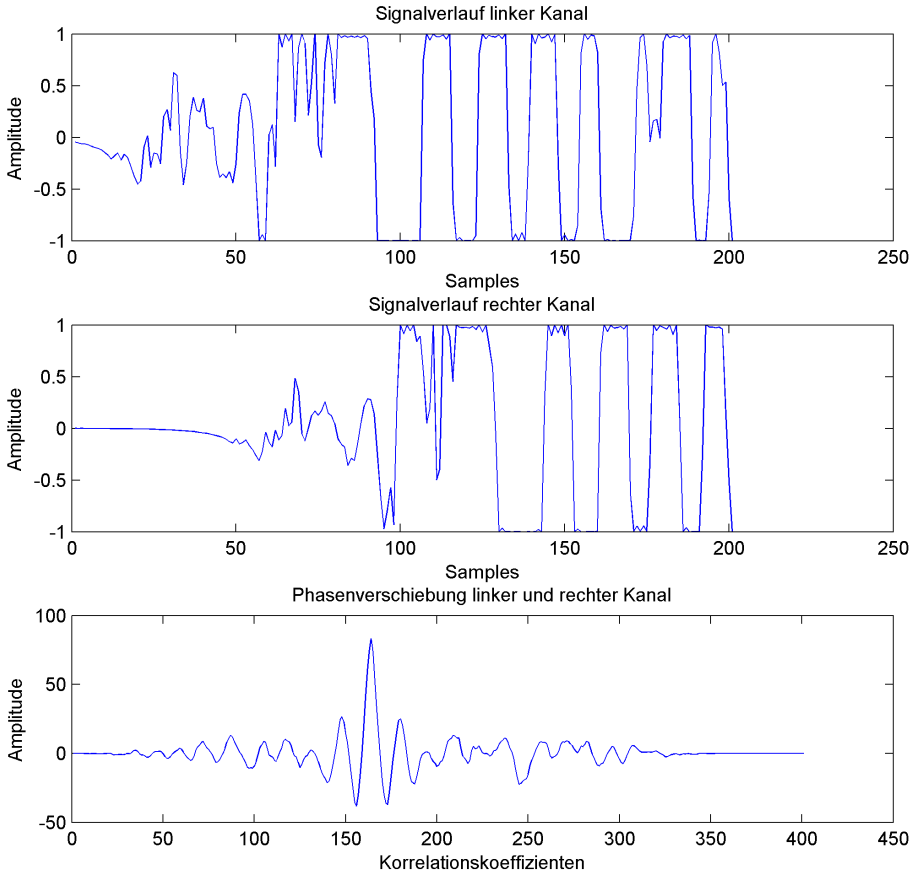

Abbildung 4.5.: Verlauf und Korrelation der vom rechten und linken Mikrofon aufgezeichneten Signale

Nun kann t_v wie folgt festlegt werden:

$$\frac{t_v}{1s} = \frac{samples_v}{f_a} \Rightarrow t_v = \frac{37}{44000} = 845\mu s \tag{4.2}$$

Nach dieser Erkenntnis kann der $90°$ Bereich ($t_{vmax} = 874,6\mu s$) von Sprecher 1 bis Sprecher 2 in zeitliche Abschnitte von $t_{va} = \frac{874,6\mu s}{90} = 9,72\mu s$ eingeteilt und die Richtung des Signals bestimmen werden.

Die Implementierung ist in Kapitel 5 zu finden. Bei Versuchen stellte sich eine Abtastung von 512 Werten als zuverlässig heraus.

4.3. Energieunterschied

Eine weitere Methode um die Richtung eines Signals zu bestimmen ist die Energiedifferenz, die zwischen den beiden Eingangssignalen berechnet wird. Zuerst wird das Signal in einem definierten Aufnahmefenster aufgenommen und für jeden Kanal die Energie als Summe aller absoluten Abtastwerte berechnet. Die Aufnahmen wurden bei derselben Konfiguration aufgenommen wie bei der Kreuzkorrelations-Methode. Dabei betrug der prozentuale Energieunterschied bezogen auf die Kanäle etwa 20%. Vor dem Einsatz müssen die Mikrofone kalibriert werden, sodass sie bei Erzeugung einer mittigen Signalquelle annährend die gleiche Energie aufweisen. Diese Messmethode ist zwar nicht so präzise wie die der Kreuzkorrelation, jedoch schneller in der Berechnung und viel einfacher zu implementieren. Grob lässt sich die Signalquelle bestimmen, ob diese rechts oder links erzeugt worden ist. Auch hier ist die Implementation in Kapitel 5 zu finden.

4.4. VoiceTracker

Im Laufe des Projekts wurde in ein neuartiges Mikrofonarray investiert, welches die Lokalisierung und Filtrierung der Sprache automatisch regelt. Abbildung 4.6 zeigt den VoiceTracker auf einer Aluminium-Halterung, die das Eigengewicht minimal halten soll. Es besteht aus 8 Mikrofonen welche 360 ° Aufnahmen ermöglichen. So kann sich der Sprecher bis zu 7 Meter (laut Hersteller *Acoustic Magic*) von dem VoiceTracker entfernen und mit dem Computer kommunizieren. Dabei wird elektronisch eine Art von Fokus oder Lokalisation auf den Sprecher realisiert. Mit dem Schalter *Field* kann der Erkennungsbreich von $+/-45°$ (narrow-angle mode) bis $+/-90°$ (wide-angle mode) eingestellt werden. Deshalb ist die Lokalisierung auf Softwarebasis nicht mehr nötig und im Hauptprojekt auskommentiert. Weitere Informationen entnehmen Sie bitte der beigelegten Bedienungsanleitung zu dem VoiceTracker.

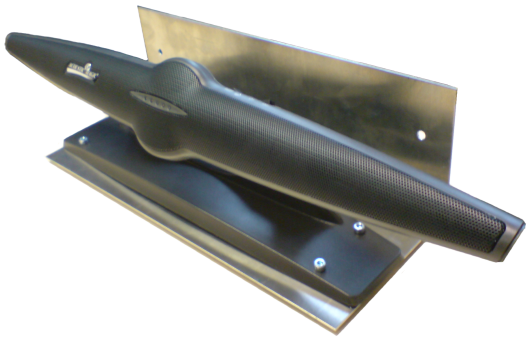

Abbildung 4.6.: VoiceTracker auf einer Aluminium-Halterung

5. Implementierung

5.1. Hardware

5.1.1. SCITOS G5

Auf dem SCITOS G5 Service-Roboter ist ein Computer mit dem Betriebssystem Fedora 12 integriert. Zur Audioaufnahme kann die 3,5 mm Mikrofonbuchse auf der OnBoard-Soundkarte verwendet werden. Zudem stehen USB-Anschlüsse für weitere externe Aufnahmegeräte zur Verfügung. Das Betriebssystem bietet das Treiberpacket ALSA und PulseAudio zur Audio-Aufnahme sowie Audio-Wiedergabe.[Met10]

5.1.2. VoiceTracker

Wie in Kapitel 4 schon erwähnt, besteht der VoiceTracker aus 8 Mikrofonen die sich auf einen Sprecher fokussieren und Störgeräusche herausfiltern. Dieses Mikrofon-Array bietet erstaunliche Ergebnisse und ist speziell für die kabellose Spracherkennung entwickelt worden. Montiert wird das Mikrofon auf der Vorderseite des Service-Roboters. Da der Roboter so wenig wie möglich mit Gewicht belastet werden sollte, dient ein Aluminiumwinkel als Halterung. Um festzustellen wie das Mikrofon auf verschiedene Raumbedingungen und Ereignisse reagiert sind 4 Testszenarien erstellt und analysiert worden. Als Testraum diente das Robot-Vision Labor an der HAW Hamburg. Dort ist wie in den meisten Räumen eine Klimaanlage installiert und bietet ein stetiges Hintergrundrauschen. Der VoiceTracker war während der Experimente auf $+/-45^{\circ}$ Erkennungswinkel und mit eingeschaltetem *LDS* konfiguriert. Die Versuche haben gezeigt, dass in dieser Einstellung die meisten Störgeräusche entfernt wurden.

Szenario 1 - Aufnahmen unter einfachen Raumbedingungen

Durchgeführt wurden einfache Testaufnahmen mit der Klimaanlage im Hintergrund. Alle Aufnahmen wiesen eine „Stille“ auf, was darauf hindeutet, dass der VoiceTracker das Rauschen herausgerechnet hat.

Szenario 2 - Aufnahmen bei eingeschalteten Ultraschallsensoren

Wenn die Ultraschallsensoren aktiv sind, erzeugen sie durch das Umschalten der Betriebsart (senden / empfangen) ein Klackern (Abbildung 5.1). VoiceTracker dämpft das Klackern, kann es aber nicht herausfiltern. Allerdings ist die Amplitude des Signals, bezogen auf den Grundwert 1 sehr niedrig, so dass es auf die Spracherkennung kaum Auswirkungen hat.

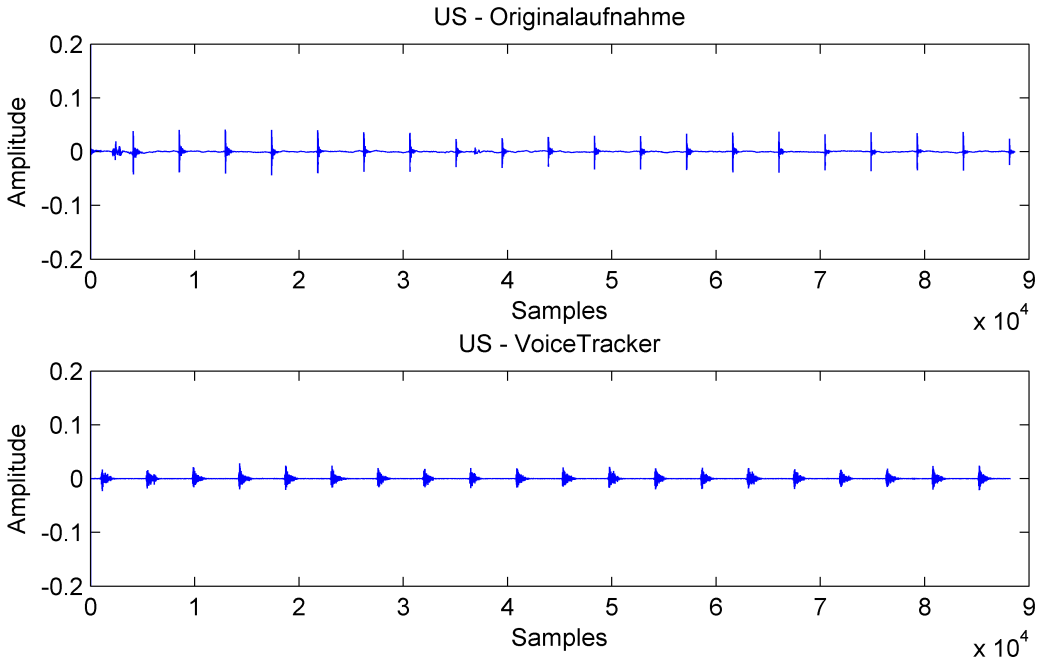

Abbildung 5.1.: Vergleich zwischen VoiceTracker und einfachen Mikrofon bei eingeschalteten Ultraschallsensoren

Szenario 3 - Aufnahmen beim eingeschalteten Motor

Die Abbildung 5.2 zeigt, dass der Motor sehr wenige Störgeräusche verursacht. Somit wäre der Motor keine ernst zu nehmende Störquelle bei der Spracherkennung.

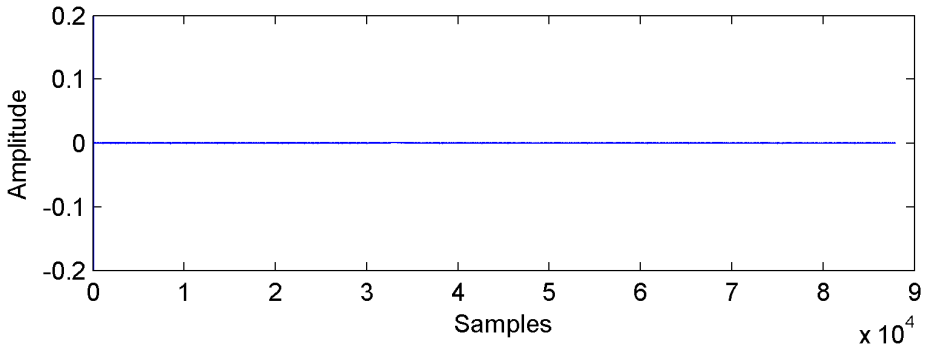

Abbildung 5.2.: VoiceTracker-Aufzeichnung beim eingeschalteten Motor

Szenario 4 - Armbewegungen

Anders als bei den anderen Szenarien ist hier wirklich ein echtes Problem vorhanden. Der VoiceTracker kann die Störgeräusche die durch den bewegenden Arm verursacht werden, nicht herausfiltern. In Abbildung 5.3 ist der Signalverlauf bei maximaler Beweggeschwindigkeit des Roboter-Arms dargestellt. Dabei sieht man, dass die Amplitude an einigen Stellen 20% erreicht. Im Betrieb würde der Spracherkenner versuchen in diesem Signal eine Äußerung zu erkennen. Abhängig davon wie gut das Sprachmodell des Erkenners ist, wird keine gültige Äußerung erkannt. Eine Möglichkeit die Störgeräusche des Arms zu ignorieren sind Parameter in der Konfigurationsdatei welche von der Julius-Engine benötigt wird. Mehr dazu finde Sie im dem Abschnitt 6.3.

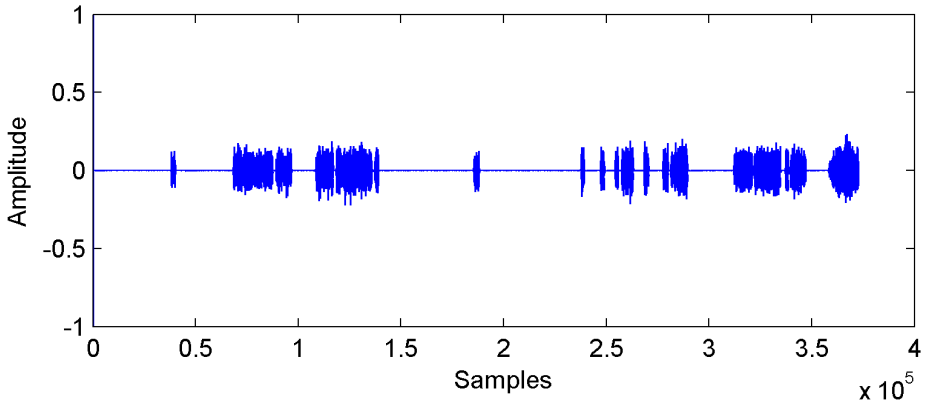

Abbildung 5.3.: VoiceTracker-Aufzeichnung beim Bewegen des Arms

5.2. Installation und Konfiguration

Nachdem in Kapitel 3 die Software die zur Verfügung steht erläutert wurde und was die Vor- und Nachteile dieser sind, kann man folgende Auswahl treffen, die zu der Aufgabenstellung passt:

HTK - Hidden Markov Toolkit
sehr flexibel, mächtig und in C geschrieben

Simon listens - Projekt für Behindertenhilfe
ermöglicht komfortables Erstellen und Trainieren eines Sprachmodells

Julius - large vocabulary continuous speech recognition decoder
performante und quellenoffene Spracherkennung

Mbrola - Sprachsynthese
natürlich klingende Sprachsynthese mit deutscher Sprachunterstützung

Bevor man mit der Installation des Sprachinterfaces und der benötigten Software beginnen kann, müssen einige Voraussetzungen erfüllt sein:

- Fedora 12/13
  Installation auf einer virtuellen Maschine ist auch möglich.
- root Berechtigung
  Einige Module benötigen root Rechte zur Ausführung
- GNU C/C++ Compiler
  `yum install gcc`
  `yum install gcc-g++`
- CMake
  `yum install cmake`
- KDE-Bibiotheken
  `yum install kdelibs-*`
- Flex
  `yum install flex`
- Bison
  `yum install bison`
- ALSA
  `yum install alsa*`

- aplay, Soundausgabe über ALSA
  `yum install aplay`

Es kann vorkommen, dass das Betriebssystem dennoch einige Packete nicht findet. Diese muss man entsprechend der Fehlermeldung beim Kompilieren nachinstallieren.

Wichtig!
Das *Hidden Markov Toolkit* und *Simon* werden ausschließlich für die Modellierung des Sprachmodells benötigt und arbeiten deshalb unabhängig von dem eigentlichen SCITOS Sprachinterface. Die Installation kann so auch unter Windows erfolgen. Dazu sind auf der CD-ROM jeweils auch die Windows-Installationsanleitung und die Installationsdateien vorhanden. Die Windows-Installation wird in dieser Arbeit nicht behandelt, da sonst der Rahmen gesprengt wird.

5.2.1. HTK

Das Grundgerüst zur Sprachmodellierung bildet das an der Cambridge University entwickelte Hidden Markov Toolkit. HTK wird auch von *Simon* zur Modellierung benötigt. Deshalb muss zuerst das gesamte HTK Paket in einen Ordner entpackt werden. Das Archiv finden Sie auf der CD-ROM oder auf der HTK-Webseite [Cam]. Nach dem Entpacken muss HTK mit `configure` konfiguriert werden. Jetzt kann HTK mittels `make install` installiert werden. Dabei werden die fertigen ausführbaren Dateien in dem zuvor definierten Ordner abgelegt.

```
% Entpacken
[fed@localhost $]# tar xzf HTK-3.3.tar.gz
[fed@localhost $]# cd htk
% Zierlordner für Binär-Dateien einstellen
[fed@localhost htk]# ./configure --prefix=/usr
% Kompilieren und installieren
[fed@localhost htk]# make install
```

5.2.2. Simon listens

Die Installation von Simon ist wichtig, da mit diesem Tool die Modellierung der Sprache vorgenommen wird. Außerdem erleichtert Simon die Aufnahme von Äußerungen durch eine GUI. Dabei können neue Wörter leicht hinzugefügt, trainiert sowie entfernt werden. Auch kann eine Grammatik für Sprachmodelle definiert werden. Das Archiv finden sie auf der CD-ROM oder auf der Simon Webseite [Sl]. Entpacken Sie das Archiv in einen beliebigen temporären Ordner und führen dort den folgenden Befehl aus:

```
[fed@localhost simon-3.0.3]# ./build.sh
```

Jetzt wird Simon kompiliert und installiert. Sie finden die Installierten Programme unter *Anwendungen* → *Zubehör*. Die Binären Dateien sind unter `/usr/bin/` abgelegt und können direkt in der Shell aufgerufen werden:

- simon
  Hauptprogram
- ksimond
  Server für die Sprachdatenbank mit GUI
- simond
  Server für die Sprachdatenbank ohne GUI
- sam
  Tool zum Testen der Sprachmodelle

Bevor man Simon startet, sollte man zuerst ksimond ausführen und ein Benutzerkonto anlegen. Um mehr über das *Simon listen* Projekt zu erfahren, lesen Sie bitte die auf der CD-ROM vorhandenen E-Books. Nach der Installation kann der temporäre Ordner mit dem Quellcode gelöscht werden.

5.2.3. Julius

Julius ist eine Spracherkennungs-Engine mit der die Sprachkommunikation mit dem mobilen Roboter SCITOS G5 realisiert wird. Da die *stateMachine* mit der Julius-Engine verbunden werden muss, legen Sie einen Ordner *julius-scitos* auf dem Desktop an und extrahieren dort den Inhalt des Archivs hin. Das Archiv `julius-4.1.5.tar.gz` kann von der Julius-Webseite [Nag] oder von der CD-ROM bezogen werden.

Jetzt kann Julius mit den folgenden Befehlen konfiguriert, kompiliert und installiert werden:

```
% Zielordner für Binäre Dateien festlegen und Audio-Treiber
einstellen
[fed@localhost julius-scitos]# ./configure --prefix=/usr
--with-mictype=alsa
% Kompilieren und installieren
[fed@localhost julius-scitos]# make install
```

Wichtig sind hier die erstellten Julius-Libraries, die man für das Sprachinterface benötigt. Also löschen Sie diesen Ordner nach der Installation unter keinen Umständen.

5.2.4. Mbrola

Das Wort Sprachkommunikation beinhaltet nicht nur die Spracherkennung sondern auch die Sprachsynthese. Damit ist der Service-Roboter SCITOS in der Lage uns den Status mitzuteilen und einen Dialog zu führen. Die Sprachsynthese besteht aus zwei Komponenten:

- txt2pho
- Mbrola

Mbrola ist ein mehrsprachiger Sprachsynthesizer und braucht als Input eine Phone-Datei aus der die Sprache synthetisiert wird. Deshalb wird noch das Tool *txt2pho* benötigt, welches den eingegebenen Text in eine Phone-Datei umwandelt. Mbrola interpretiert die Phone-Datei und erstellt eine Wave-Datei als Output. Diese kann mit einem beliebigen Player (z.B. *aplay*) abgespielt werden. Für weitere Informationen besuchen Sie die Webseite von Mbrola [TCT].

Kopieren Sie zuerst den Ordner *SOFTWARE/MBROLA* von der CD-ROM auf den Desktop. In diesem Ordner befinden sich alle nötigen Binärdateien, sowie die Quellcodes. Des Weiteren befinden sich sechs unterschiedliche Stimmen der deutschen Sprache für Mbrola. Öffnen Sie nun ein Terminal und wechseln Sie in das zuvor auf den Desktop kopierte Verzeichnis *MBROLA*.

Installieren Sie Mbrola mit dem Befehl

```
[fed@localhost TTS]# cp MBROLA/bin/mbrola-linux-i386
/usr/local/bin/mbrola
```

Es folgt das Installieren von txt2pho und den dazugehörigen Tools:

```
[fed@localhost MBROLA]# cd txt2pho
[fed@localhost txt2pho]# cp txt2pho /usr/local/bin/
% txt2pho Konfigurationsdatei anlegen
[fed@localhost txt2pho]# cp txt2phorc /etc/txt2pho
% numfilt kompilieren und installieren
[fed@localhost numfilt-0.1]# cd .. && cd numfilt-0.1
[fed@localhost numfilt-0.1]# gcc -o numfilt numfilt.c
[fed@localhost numfilt-0.1]# cp numfilt /usr/local/bin/
% pipefilt kompilieren und installieren
[fed@localhost numfilt-0.1]# cd .. && cd pipefilt
[fed@localhost pipefilt]# g++ -o pipefilt pipefilt.cc
[fed@localhost pipefilt]# cp pipefilt /usr/local/bin/
% preproc kompilieren und installieren
[fed@localhost pipefilt]# cd .. && cd preproc
[fed@localhost preproc]# make clean && make
[fed@localhost preproc]# cp preproc /usr/local/bin/
```

```
% Arbeitsverzeichnis txt2speech anlegen
[fed@localhost preproc]# cd ..
[fed@localhost txt2pho]# mkdir /usr/local/txt2speech
[fed@localhost txt2pho]# cp -r data/ /usr/local/txt2speech
% Sprachdateien kopieren
[fed@localhost txt2pho]# cd ..
[fed@localhost MBROLA]# cp -r DE-STIMMEN/* /usr/local/txt2speech
% Für den Anwender verfügbar machen
[fed@localhost MBROLA]# chmod -R o+rx /usr/local/txt2speech
```

Nach der Installation ist es wichtig *txt2pho* richtig zu konfigurieren. Dazu öffnen Sie mit *gedit* oder einem anderen Texteditor die in `/etc` liegende Konfigurationsdatei `txt2pho`. Jetzt ändern Sie die folgenden Zeilen:

`DATAPATH=/home/tpo/txt2pho/data/` ändern in
`DATAPATH=/usr/local/txt2speech/data/`

und

`INVPATH=/home/tpo/txt2pho_data/` ändern in
`INVPATH=/usr/local/txt2speech/data/`

Damit ist die Installation erfolgreich abgeschlossen. Für einen Funktionstest geben sie den folgenden Befehl in ein Terminal ein:

```
[fed@localhost  ]# echo "Guten Tag" | txt2pho > /tmp/test.pho
&& mbrola /usr/local/txt2speech/de5/de5 /tmp/test.pho /tmp/test.wav
&& aplay /tmp/test.wav
```

5.2.5. SCITOS-Sprachinterface

Zum Schluss kann das eigentliche SCITOS Sprachinterface kompiliert und installiert werden: Dazu kopieren Sie den SCITOS Interface Ordner *SOFTWARE/SPRACHINTERFACE/SCITOSsi* von der CD-ROM in den auf dem Desktop liegenden Ordner *julius-scitos*:

```
cp //CDROM/SOFTWARE/SPRACHINTERFACE/SCITOSsi /home/fed/
Desktop/julius-scitos/
```

Jetzt gehen Sie im Terminal in den *SCITOSsi* Ordner und führen dort diese Befehle aus:

```
% Kompilieren
[fed@localhost SCITOSsi]# make distclean && make
```

Erstellt wird eine binäre Datei `SCITOSsi`, die das ausführbare Sprachinterface bildet. Das Sprachinterface benötigt noch das Sprachmodell, damit es ausgeführt werden kann. Wie man ein eigenes Sprachmodell anlegt oder erweitert lesen Sie in Kapitel 6. Ein Beispiel-Sprachmodell befindet sich auch auf der CD-ROM in dem Ordner `SOFTWARE/SPRACHINTERFACE/KIARA-Sprachmodell`. Kopieren Sie diesen Ordner auf Ihren Desktop. Verschieben Sie nun die binäre Datei `SCITOSsi` in diesen Ordner und rufen dort mit dem Terminal folgenden Befehl auf:

```
[fed@localhost KIARA Demo]# SCITOSsi -C kiara.conf
```

Die Kommunikation kann jetzt nach dem Zustandsdiagram aus Abbildung 5.10 auf Seite 54 stattfinden. Standardmäßig ist die Sprachausgabe über Mbrola ausgeschaltet. Diese kann vor dem Kompilieren von dem Sprachinterface in der Datei *DEFINES.h* eingeschaltet werden.

5.3. SCITOS-Sprachinterface

Das SCITOS-Sprachinterface ist in C/C++ entwickelt worden und bildet den Kern der Sprachkommunikation. Es baut auf der Spracherkennungs-Engine Julius auf und nutzt die mit Simon erstellten Sprachmodelle für die Erkennung. Der Aufbau besteht aus den Hauptkomponenten

- Spracherkenner
  `SCITOSsi.cpp`
- Nachrichtenschlange
  `msgQueue.cpp`
- Dispatcher
  `dispatcherThread.cpp`
- Zustandsautomat
  `stateContext.cpp, state.cpp, stateImpl.cpp, timerMachine.cpp`
- Sprachsynthesizer
  `sayText.cpp`

Von der Spracherkennung bis zur Ausführung eines Befehls, wird der Ablauf in einem Diagramm (Abbildung 5.4 auf Seite 48) dargestellt. Die eigentliche Spracherkennung läuft über die *JuliusLib*. Erkennt nun der Spracherkenner eine Äußerung, werden die erkannten Worte in ein Nachrichtenpaket verpackt und an die Message Queue gesendet. Der Dispatcher

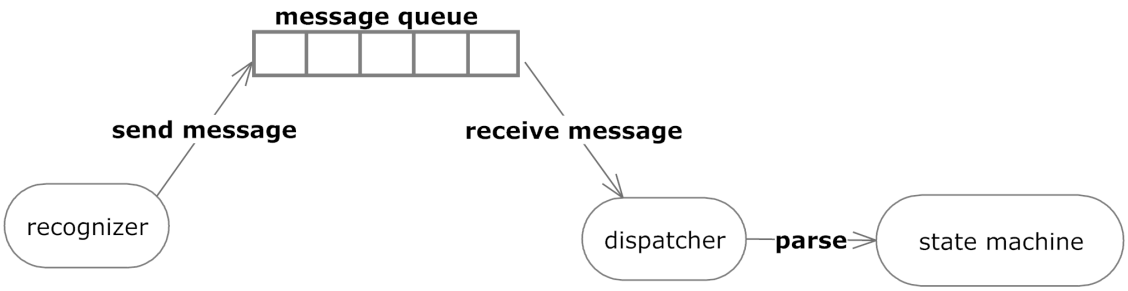

Abbildung 5.4.: Übersicht der Komponenten und deren Arbeitsweise

wartet auf Nachrichten der Warteschlange und sobald eine eintrifft, wird sie mit dem Zustandsautomaten verarbeitet. Der Spracherkenner und der Dispatcher arbeiten jeweils in einem eigenen Thread, daher läuft die Verarbeitung parallel ab.

Da alle Header- und Quellcode-Dateien des Projekts ausführlich kommentiert sind, wird im Folgenden nur der Kern aller Komponenten beschrieben. Eine mit *Doxygen* generierte Dokumentation ist ebenfalls auf der beiligenden CD-ROM enthalten.

5.3.1. Spracherkenner

`SCITOSsi.cpp`

Als Grundlage für die Programmierung des Sprachinterfaces wurde das Beispielprojekt *julius-simple* von Julius herangezogen. Sie finden das Originalprojekt in der Julius-Archiv Datei *julius-4.1.5.tar.gz* auf der CD-ROM. Bei der Implementierung wurden unnötige Verarbeitungsschritte und Ausgaben entfernt und der Dispatcher samt der Zustandsmaschine hinzugefügt. Die neu implementierten Komponenten müssen in der folgenden Reihenfolge definiert und instanziiert werden:

```
1
2       /*  create instances  */
3       TimerMachine::getInstance();
4       SayText::getInstance();
5
6       /*  sound location disabled, using VoiceTracker */
7       //SoundSource::getInstance();
8
9       /* first state  */
10      Z_idle * z1;
11      z1=new Z_idle;
12
```

```
13        /* create context to hold the state pointer */
14        StateContext * sc;
15        sc=new StateContext;
16        sc->setState(z1);
17
18        /*  create and run dispatcher */
19        DispatcherThread disp;
20        disp.start(sc);
```

Kern des Spracherkenners ist die JuliusLib. Nachdem eine Äußerung erkannt worden ist, wird das zuvor registrierte Callback
`static void output_result(Recog *recog, void *dummy)` aufgerufen und die erkannten Sätze und Wörter verarbeitet. Wie in dem unterstehenden Listing zu sehen ist, werden alle erkannten Worte einzeln in Nachrichtenpakete verpackt und an die Nachrichtenschlange verschickt.

```
1         /* send the recognized word with the        *
2          * score and sound direction to dispatcher  */
3         for(i=0;i<seqnum;i++)
4         {
5           /*  NOTE: SoundLocator is disabled, *
6            *  so default direction is MIDDLE. */
7           printf("' \%s"', winfo->woutput[seq[i]]);
8           MsgQueue::getInstance()->sendMsg(winfo->woutput[seq[i]],
9             s->confidence[i],MIDDLE);
10        }
```

Ein Nachrichtenpacket besteht aus dem erkannten Wort, der erkannten Wahrscheinlichkeit und der Signalquelle. Allerdings ist die Sprecherlokalisierung ausgeschaltet, da der VoiceTracker eingesetzt wird. Es wird einfach der Standartwert `MIDDLE` mitgesendet, um zu simulieren, dass das Signal von vorne auf den Roboter erzeugt worden ist. Der Grund dafür ist das neue VoiceTracker Mikrofonarray, welches während des Projekts erworben und eingesetzt wurde. Die Wahrscheinlichkeit des erkannten Wortes wird als `double` im Wertebereich von 0 ... 1 angegeben. Dabei steht der Wert 1 für 100%. Der Spracherkenner läuft in einem eigenen Thread.

5.3.2. Nachrichtenschlange

msgQueue.cpp

Da zur Abarbeitung mehrere Threads arbeiten sollen, ist es nötig Daten zwischen den Threads auszutauschen. Dafür eignet sich eine IPC Message Queue, die Nachrichten von einem Thread entgegen nimmt und an einen anderen weiterleitet. Deshalb muss die Nachrichtenschlange auch Thread-Safe sein. An dem Klassendiagramm in Abbildung 5.5 sieht man, dass die Klasse noch einen Struct `msgbuf` definiert hat. Dieser bildet das Nachrichtenpaket ab, welches über die Nachrichtenschlange gesendet wird.

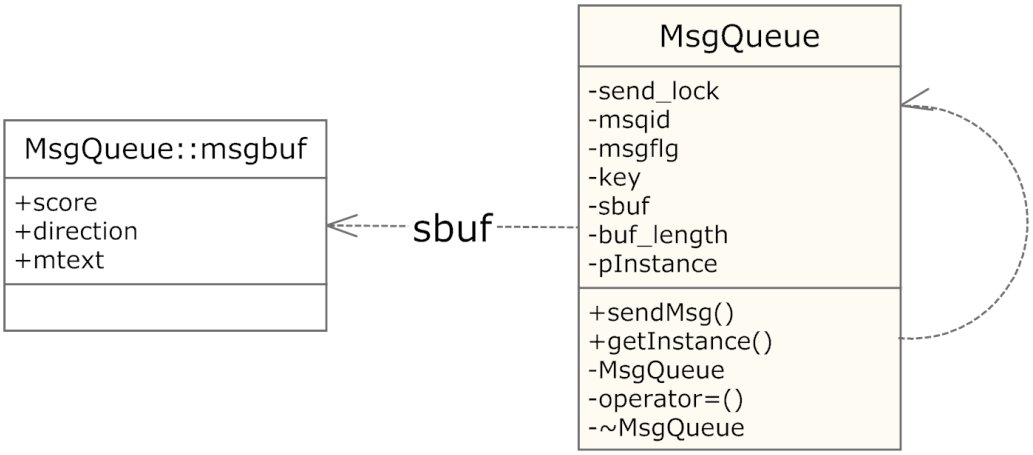

Abbildung 5.5.: Klassendiagramm der Nachrichtenschlange

Des Weiteren ist die MsgQueue-Klasse als Singleton [vgl. HKK05, S. 664] implementiert. So ist die ganze Programmlaufzeit nur ein Objekt vorhanden, über welches die POSIX IPC Message Queue angesprochen werden kann.

5.3.3. Dispatcher

dispatcherThread.cpp

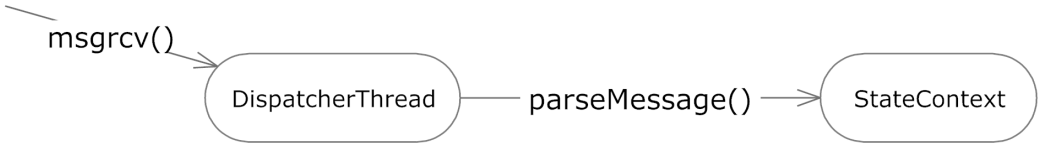

Abbildung 5.6.: Kommunikationsdiagramm vom Dispatcher zum Zustandsautomaten

Nachdem der Spracherkenner eine Nachricht an die Message Queue gesendet hat, holt auf der anderen Seite der Dispatcher die Nachrichten ab und verarbeitet diese mit dem Zustandsautomaten (Abbildung 5.6). Da diese Verarbeitung in einem Thread stattfinden soll, ist *DispatcherThread* von der Klasse *Thread* (`thread.cpp`) abgeleitet. Das UML-Diagramm in Abbildung 5.7 auf Seite 51 zeigt den Zusammenhang der einzelnen Komponenten. Dabei stellt die Klasse *Thread* Grundfunktionen zum Erstellen, Zuweisen und Starten eines Threads zur Verfügung. Der Struct `rbuf` ist der gleiche Typ wie `sbuf` aus der Klasse *MsgQueue*.

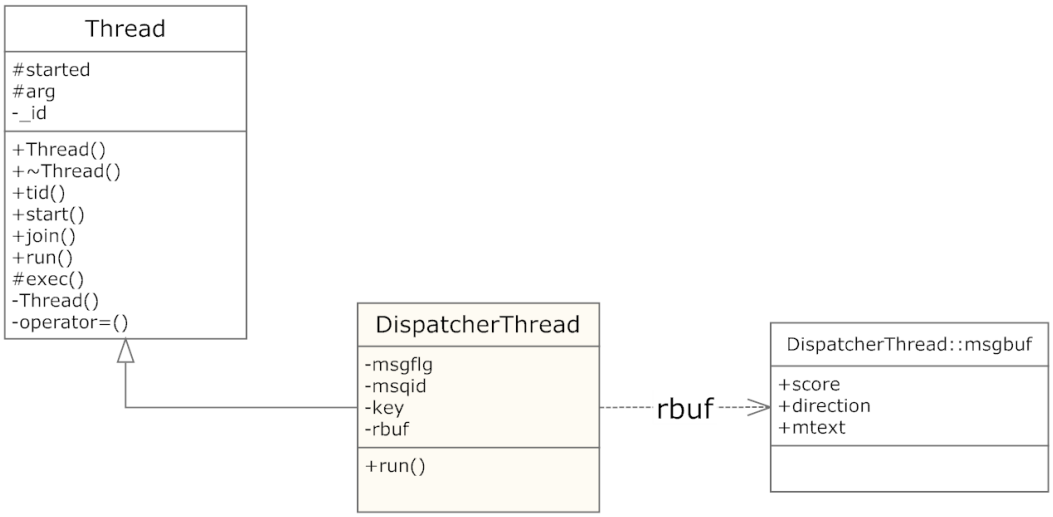

Abbildung 5.7.: Klassendiagramm des Dispatchers

Nach der Instanziierung der *DispatcherThread* Klasse muss beim Aufruf der Methode `start()` unbedingt ein *StateContext* angegeben werden. Der leere Aufruf führt zu einer Ausgabe auf das Terminal:

Richtig!

```
1        /* first state  */
2        Z_idle * z1;
3        z1=new Z_idle;
4
5        /* create context to hold the state pointer */
6        StateContext * sc;
7        sc=new StateContext;
8        sc->setState(z1);
9
10       /*  create and run dispatcher */
11       DispatcherThread disp;
12       disp.start(sc);
```

Falsch!

```
1        DispatcherThread disp;
2        disp.start(NULL);
```

Nun läuft der Thread des Dispatchers und bleibt in einer `while`-Schleife der Methode `run()`. Hier werden die Nachrichten aus der Message Queue abgeholt und verarbeitet. Der Code-Ausschnitt aus `dispatcherThread.cpp`

```
1        /*  check if the message pass the min. score and the direction */
2        if(rbuf.direction==MIDDLE && rbuf.score >= MIN_SCORE)
3        {
4          if (((StateContext *)arg) != NULL)
5            ((StateContext *)arg)->parseMessage(rbuf.mtext);
6          else
7            printf("dispatcher: %s\n", rbuf.mtext);
8        }
```

prüft die ankommende Nachricht auf die Signalrichtung (`direction`) und die Erkennungswahrscheinlichkeit (`score`). Werden die in `DEFINES.h` definierten Grenzwerte eingehalten, wird der Nachrichtentext `mtext` mit dem in *StateContext* gespeicherten Zustand abgearbeitet.

5.3.4. Zustandsautomat

`stateContext.cpp, state.cpp, stateImpl.cpp`

Für die Abarbeitung der spezifischen Nachrichten eignet sich am besten ein Zustandsautomat. Die Implementierung ist sehr einfach gehalten, sodass eine Erweiterung oder Änderung möglich ist. Das Klassendiagramm der Komponenten ist in Abbildung 5.8 dargestellt.

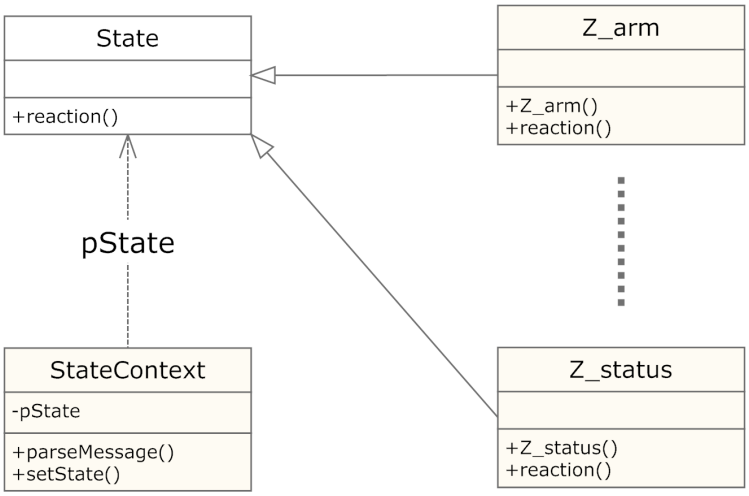

Abbildung 5.8.: Klassendiagramm des Zustandsautomaten und der Zustände

Die Klasse *StateContext* hält mit dem Zeiger `pState` den aktuellen Zustand fest, wobei der Anfangszustand mit `setState()` erst zugewiesen werden muss. Die Methode `parseMessage()` leitet die ankommende Nachricht an den aktuellen Zustand in `pState` weiter und ruft dort die `reaction()`-Methode auf. In der Datei `stateImpl.cpp` sind alle implementierten Zustände zu finden, wobei die Klasse *State* als abstrakte Klasse dient. Das Zustandsdiagramm des Demo-Projekts KIARA ist in Abbildung 5.10 auf Seite 54 zu finden. Der Übergang von einem Zustand zum anderen wird durch den `new`-Operator Aufruf realisiert. Hierbei wird das Objekt des aufrufenden Zustands durch das Objekt des neuen ersetzt (Abbildung 5.9). Das hat den Vorteil, dass der Speicher und die Speicherverwaltung wenig beansprucht werden.

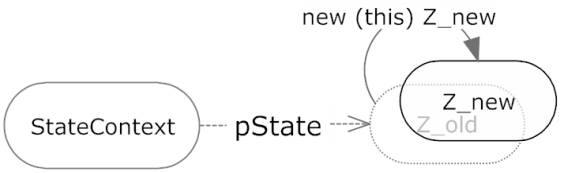

Abbildung 5.9.: Zustandswechsel mit dem `new`-Operator

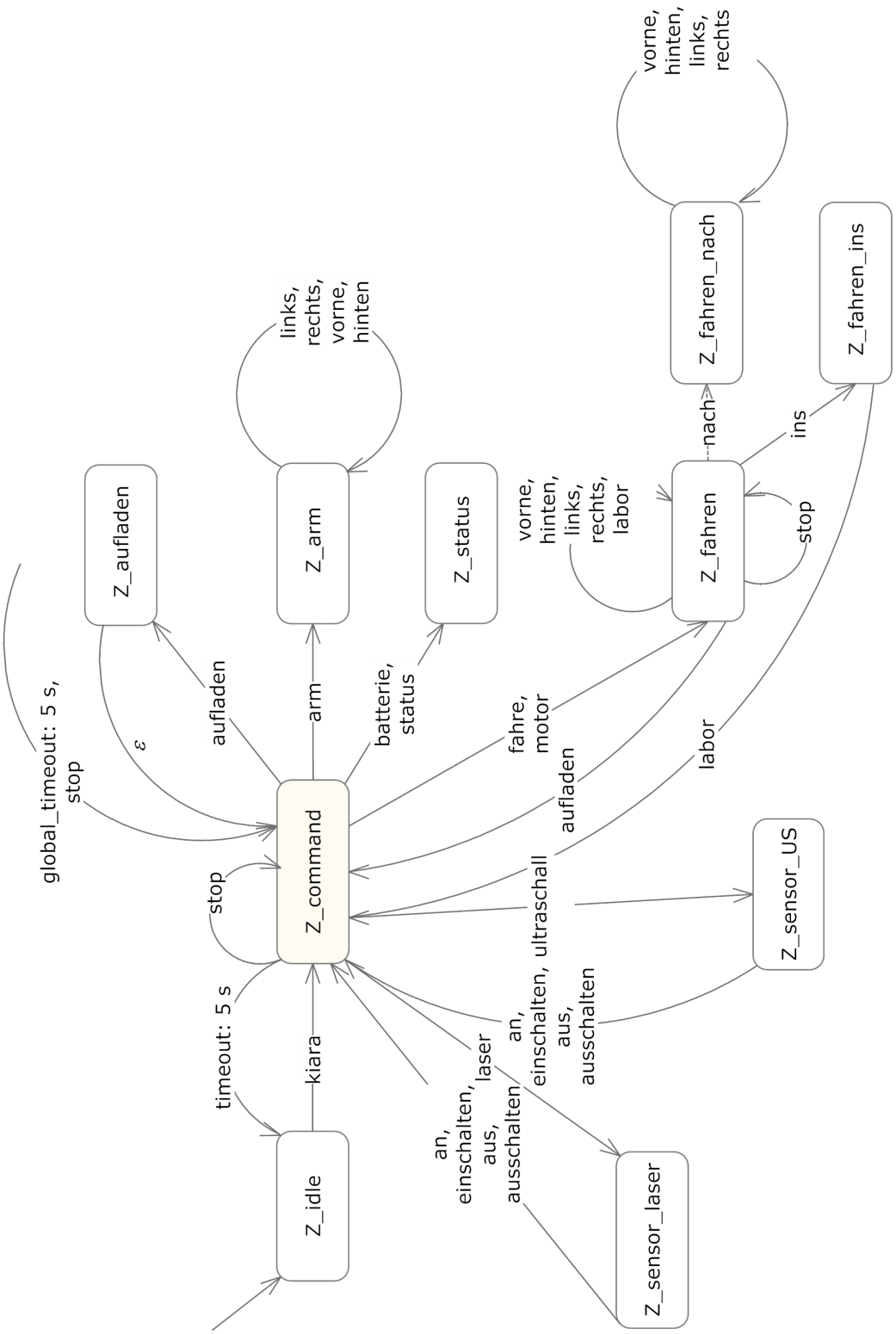

Abbildung 5.10.: Zustandsdiagramm des Demo-Projekts KIARA

Zustandsautomaten erweitern

In diesem Abschnitt wird dem KIARA-Zustandsautomaten (Abblidung 5.10 auf Seite 54) ein neuer Zustand hinzugefügt. Das erweiterte Zustandsdiagramm ist in Abbildung 5.11 dargestellt. Befindet sich der Automat im Befehlszustand `Z_command`, so soll der neue Zustand `Z_neuer_Zustand` durch den Befehl „*test*“ erreicht werden. Zu Testzwecken wird dann eine Äusserung über den Sprachsynthesizer ausgegeben und nach einem Timeout von 2 Sekunden wieder in den Befehlszustand `Z_command` zurückgekehrt.

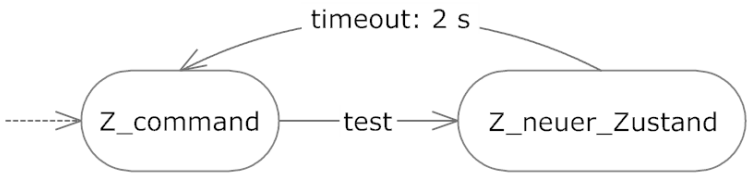

Abbildung 5.11.: Das erweiterte Zustandsdiagramm mit dem neuen Zustand

Um einen neuen Zustand hinzuzufügen muss zuerst ein Eintrag in der Header-Datei `statesImpl.h` vorgenommen werden:

```
...
class Z_neuer_Zustand:public State
{
public:
  Z_neuer_Zustand();
  void reaction(char * Message);
};
...
```

Nun muss die Klasse `Z_neuer_Zustand` in der Quellcode-Datei `statesImpl.cpp` implementiert werden:

```
...
/*  Konstruktor, Eintritt in einen Zustand  */
Z_neuer_Zustand::Z_neuer_Zustand()
{
  TimerMachine::getInstance()->addTimer(2000,"z_neuerZstd_to");
  SayText::getInstance()->say("Neuer Zustand erreicht!");
}
/*  Hier werden die zugestellten Nachrichten verarbeitet  */
void Z_neuer_Zustand::reaction(char * Message)
{
  if(strcmp(Message,"z_neuerZstd_to")==0)
    new (this) Z_command;
}
```

Zum Schluss ist noch die Transition von Zustand `Z_command` zu `Z_neuer_Zustand` nötig:

```
1  void Z_command::reaction(char * Message)
2  {
3    ...
4    if(strcmp(Message,"fahr")==0 || strcmp(Message,"motor")==0)
5    {
6      new (this) Z_fahren;
7    }
8    else
9    /* <Erweiterung> */
10   if(strcmp(Message,"test")==0)
11   {
12     new (this) Z_neuer_Zustand;
13   }
14   else
15   /* </Erweiterung> */
16   if(strcmp(Message,"ultraschall")==0)
17   {
18     new (this) Z_sensor_US;
19   }
20   ....
21 }
```

Das Wort „*test*“ muss auch im Sprachmodell aufgenommen werden. Wie man ein neues Wort einem Sprachmodell hinzufügt ist in Kapitel 6 beschrieben. Achten Sie darauf, dass die Groß- und Kleinschreibung des Wortes im Sprachmodell und im implementierten Zustand gleich sein müssen.

timerMachine.cpp

Zu sehen sind auch Timeouts die in der Klasse *TimerMachine* implementiert sind. Wie *MsgQueue* ist auch *TimerMachine* als Singleton programmiert. Des Weiteren läuft diese Klasse als separater Thread, abgeleitet von der *Thread*-Klasse. Das hat den Vorteil, dass alle Timer zentral verwaltet werden und unabhängig vom Zustandsautomaten ablaufen. Nachdem z.B. ein Zustand mit der Methode `addTimer()` einen Timer mit seiner spezifischen Nachricht erstellt, legt *TimerMachine* die Informationen in einem Array ab. Dann prüft der Thread, nach der in DEFINES.h definierten Zeit $TIMER\_DELAY$, den Ablauf der Timer in dem Array und sendet gegeben falls die Timeout-Nachricht an die IPC Message Queue. Der schematische Ablauf ist in der Abbildung 5.12 gezeigt und das Klassendiagramm in Abbildung 5.13 .

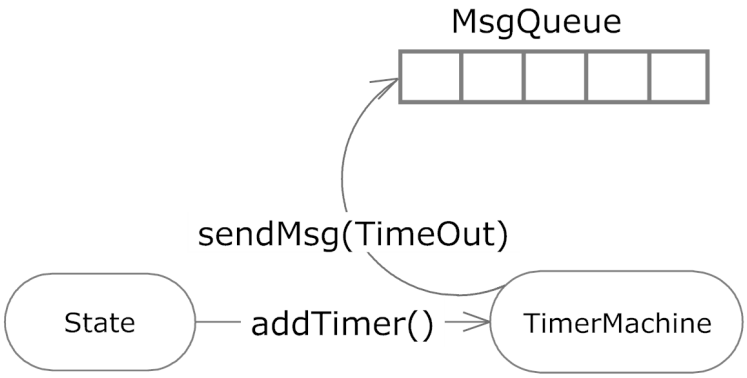

Abbildung 5.12.: Kommunikationsdiagramm von TimerMachine zum Zustandsautomaten

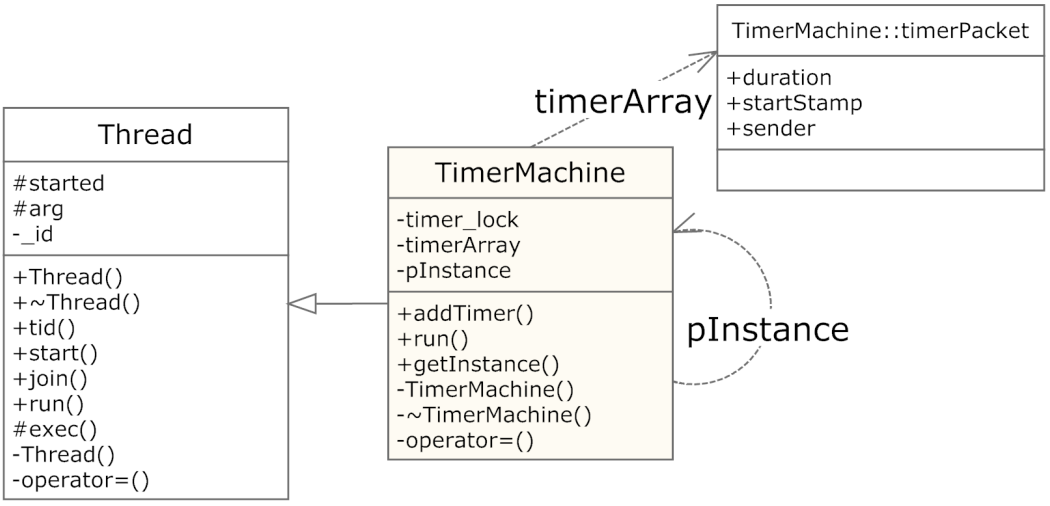

Abbildung 5.13.: Klassendiagramm des Zeitgebers TimerMachine

5.3.5. Sprachsynthesizer

sayText.cpp

Der Sprachsynthesizer ist die am einfachsten implementierte Komponente. Für den globalen Zugriff wurde es in dem Singleton Design Pattern programmiert. Damit diese Komponente funktionieren kann, muss unbedingt die Installation von *Mbrola* auf Seite 45 durchgeführt werden. Deshalb ist der Sprachsynthesizer über die Variable `TTS` in `DEFINES.h` standardmäßig ausgeschaltet.

In der Klassenansicht (Abbildung 5.14) ist zu erkennen, dass der Kern in der Funktion `say()` liegt. Beim Aufruf der Funktion muss der Text als ein `char`-Array übergeben werden.

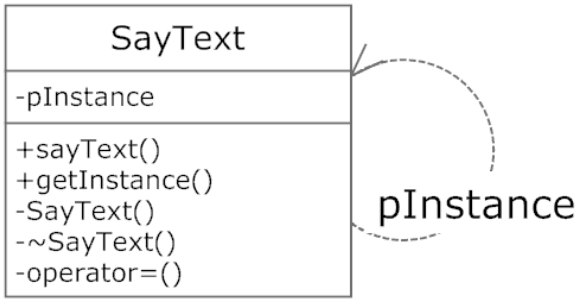

Abbildung 5.14.: Klassendiagram von SayText

Dabei wird ein Systembefehl generiert, der die Tools `echo, txt2pho, mbrola` und `aplay` kombiniert und mit `system()` ausführt. Sind Änderungen geplant, achten Sie bitte darauf, dass der Befehl `system()` blockierend und nicht-blockierend ausgeführt werden kann. Blockierend bedeutet dass der Prozess bis zum Beenden der Funktion angehalten wird. Das Blockieren an dieser Stelle ist wichtig, damit die synthetisierte Ausgabe nicht von einer anderen überlagert werden kann. Folgendes Listing zeigt die Nutzung des Befehls:

blockierend

```
1         system("mkdir test"');
```

nicht-blockierend

```
1         system("&mkdir test");
```

5.4. Sprecherlokalisierung

5.4.1. Kreuzkorrelation

main.cpp, soundSource.cpp

Die Lokalisierung wird hauptsächlich in der Klasse `SoundSource` realisiert.
Methode `int SoundSource::getDirection()` nimmt über den ALSA-Treiber eine definierte Menge an Frames auf und leitet den Puffer weiter an die Funktion `int SoundSource::sndSource(short * buf, int size)` um die Richtung mittels der Kreuzkorrelation zu berechnen. Als Rückgabewert wird `RIGHT, LEFT, MIDDLE` und `NO_CHANGE` zurückgegeben. Die Werte sind in der Datei `DEFINES.h` definiert. Der Projektordner befindet sich in dem Ordner `SOFTWARE/SPRECHERLOKALISIERUNG/xcorr_locate` auf der CD-ROM. Beim Kompilieren muss die ALSA Bibliothek angegeben werden:

```
g++ -o main main.cpp soundSource.cpp -lasound
```

5.4.2. Energieunteschied

power_locate.cpp

Im Prinzip ist das Programm genau so aufgebaut wie das der Kreuzkorrelation, nur dass sich die Berechnung auf die aufgenommene Signalenergie bezieht. Es werden die absoluten Werte berechnet, sodass auch die negativen Anteile des Signals berücksichtigt werden. Außerdem ist noch die Klasse `ProgressBar` dabei. Sie ermöglicht die Anzeige der ermittelten Signalrichtung in einer Konsole. Das Projekt ist in dem Ordner `SOFTWARE/SPRECHERLOKALISIERUNG/power_locate` auf der CD zu finden. Auch hier muss beim Kompilieren die ALSA Bibliothek angegeben werden:

```
g++ -o power_locate power_locate.cpp progressBar.cpp -lasound
```

6. Anwendung

In diesem Kapitel wird auf die Schritte eingegangen, die notwendig sind um ein Sprachmodell für das SCITOS-Sprachinterface zu erstellen oder das bereits vorhandene Demo-Projekt KIARA zu erweitern. Dazu muss die Installation aus Kapitel 5 bereits erfolgreich durchgeführt worden sein.

6.1. Sprachmodell erzeugen

Bevor man überhaupt mit der Sprachmodellierung beginnt, sollte man sich zuerst Gedanken machen, welche Worte erkannt werden sollen und wie darauf zu reagieren ist. Am besten lässt sich das in einem Zustandsdiagramm darstellen. Hier wird Schritt für Schritt das Sprachmodell für das Zustandsdiagramm (Abbildung 5.10 auf Seite 54) des Demo-Projekts KIARA erzeugt. Die Modellierung geschieht mit Simon. Starten Sie *Simon* und stellen Sie die Verbindung zu dem *simond* Server her.

6.1.1. Schattenwörterbuch HADIFIX BOMP

In Simon besteht die Möglichkeit mit einem Schattenwörterbuch zu arbeiten. Es beinhaltet die Lautschrift zu den vorhandenen Wörtern. Außerdem bietet das HADIFIX BOMP Schattenwörterbuch Terminalinformationen, welche hilfreich für die Grammatikbildung sind. Da HADIFIX BOMP [PWU] einer restriktive Lizenz unterliegt muss es über Simon heruntergeladen werden. Dazu navigieren Sie in Simon auf *Vokabular* → *Wörterbuch importieren*. Folgen Sie dem Assistenten bis in das Menü *HADIFIX Wörterbuch importieren* und wählen die Option *Das HADIFIX BOMP automatisch herunterladen und installieren*. Nach Eingabe Ihrer Daten und Akzeptieren der Lizenzbestimmungen wird es heruntergeladen und installiert. Eine Kopie befindet sich ebenfalls auf der beigelegten CD-ROM im Ordner `SOFTWARE/SIMON`.

6.1.2. Wort hinzufügen

Jedes vorkommende Wort in unserem Zustandsdiagramm muss einzeln hinzugefügt werden. Es ist empfehlenswert eine Liste (Tabelle 6.1) mit dem gesamten Vokabular zu erstellen, die einen guten Überblick verschafft und später zum Erstellen der Grammatik dient.

an	aus	einschalten	ins	laser	nach	stop
arm	ausschalten	fahr	kiara	links	rechts	ultraschall
aufladen	batterie	hinten	labor	motor	status	vorne

Tabelle 6.1.: Das gesamte Vokabular aus dem KIARA Zustandsdiagramm

Um ein Wort hinzuzufügen betätigen Sie in Simon den Button *Wort Hinzufügen* und geben das entsprechende Wort ein. Im kommenden Fenster werden nun das Wort, das dazugehörige Terminal und die Aussprache angezeigt, soweit es im Schattenlexikon verfügbar ist. Die Terminale sind für die Grammatik wichtig, denn diese ermöglicht uns Wortketten und Sätze zu bilden. Mit der Aussprache ist die Lautschrift gemeint, welche in die statistischen Berechnungen der HMMs einfließt. Die Liste aller verfügbaren Terminale im HADIFIX BOMP Lexikon ist in Tabelle 6.2 auf Seite 62 dargestellt [RB08, S. 42]. Im nächsten Schritt wird die Lautstärke des Mikrofons geprüft. Danach folgen Sie dem Assistenten um Aufnahmen zu erstellen. Sprechen Sie dabei ganz normal und verständlich, nicht extra langsam oder zu schnell. Nach dem Hinzufügen des Wortes wird Simon das Sprachmodell, auch wenn es nur ein Wort ist, kompilieren und in dem Benutzerordner ablegen. Sie finden den Ordner abhängig vom Betriebssystem in:

Windows

```
%appdata%\.kde\share\apps\simond\models\USERNAME\active
```

Linux

```
~/.kde/share/apps/simond/models/USERNAME/active
```

Wiederholen Sie die Vorgänge für alle Wörter aus dem Vokabular. Das aktuelle Vokabular in Simon kann durch den Reiter *Vokabular* aufgerufen werden. Sollte ein Wort nicht im Schattenlexikon vorhanden sein, kann es trotzdem hinzugefügt werden. Als Beispiel wählen wir das Wort *„einschalten“* welches nicht im HADIFIX BOMP Lexikon vorhanden ist, dafür aber die Teilworte *„ein“* und *„schalten“*. Nun nimmt man die Lautsprache der Teilworte und fügt diese zusammen zu `gls aI n S a l t @ n = gls aI n + S a l t @ n .` Jetzt fehlt nur noch das richtige Terminal. Da das Wort *„einschalten“* ein Verb ist wählt man `VRB`.

ADD	Additive, Präpositionen oder Postpositionen
ADV	Adverbien
ART	Artikel
INJ	Interjektion
KNJ	Konjugation
MOD	modales Formwort
NAM	Namen
NEG	Negativpronomen, Negation
NOM	Nomen
NUM	Numerale
PPR	Personalpronomen
PRO	Pronomen
QUAL	Qualifier, dienen zum Qualifizieren andere Wortformen
QAN	QUAL, Adnominal
QAV	QUAL, Adverb: QUAL, die syntaktisch nur die Position des Adverbs einnehmen
QPN	QUAL, Prädikativ, Nomen
QPV	QUAL, Prädikativ, Adverb: QUAL, die prädikativ oder adverbial funktionieren
QAL	Attributive Alternanten der QPV
REL	Relativpronomen
REFL	Reflexivpronomen
RFW	reflexives Formwort, Pronomen in reflexiver Funktion
SBJ	Subjunktion = unterordnendes Wort
TME	Tmesis fähiges Präverb, Zerlegen von Verbkonstruktionen
VEH	Hilfsverb 'haben'
VEM	Modalverben
VES	Hilfsverb 'sein'
VEW	Hilfsverb 'werden'
VRB	Verben

Tabelle 6.2.: Liste der Wortarten aus dem HADIFIX BOMP Lexikon [RB08]

6.1.3. Grammatik erzeugen

Die Grammatik beschreibt die Reihenfolge der Wörter in der diese vorkommen können. Bezogen auf das KIARA Zustandsdiagramm und Vokabular sind in Tabelle 6.3 auszugsweise zugelassene Wortkombinationen mit den dazugehörigen Terminalen abgebildet.

kiara	fahr	nach	rechts
NAM	VRB	ADD	ADD
kiara	fahr	ins	Labor
NAM	VRB	ADD	NOM
ultraschall	einschalten		
NOM	VRB		
laser	aus		
NOM	ADD		

Tabelle 6.3.: Beispiel für gültige Wortkombinationen und Terminale

Simon bietet unter dem Reiter *Grammatik* den Button *Satz hinzufügen* mit dem man die Terminale einer Wortkette eingeben kann umso eine Grammatik zu erzeugen. Natürlich wäre die manuelle Eingabe jeder Wortkombination aus dem KIARA Zustandsdiagramm sehr aufwendig, deshalb kann man durch das Importieren von Grammatik aus einem Text oder einer Textdatei viel Zeit sparen. Legen sie dazu eine Textdatei an und schreiben dort alle möglichen Wortkombinationen betreffend dem Zustandsdiagramm auf. Die fertige Textdatei zu dem KIARA Zustandsdiagramm ist bereits vorhanden → `CD-ROM/DOKUMENTATION/SPRACHINTERFACE/KIARA Grammatik.txt`. Dabei sind auch Einzelworte pro Zeile zugelassen. Navigieren Sie mit dem Button *Importieren* im *Grammatik*-Menü zu dem Import-Assistenten und befolgen dort die Hinweise zum Importieren einer Textdatei. Nun wird Simon alle Terminale nach dem Satzbau automatisch anlegen und daraus eine Grammatik generieren.

6.1.4. Training

Nachdem nun alle Wörter hinzugefügt worden sind und eine Grammatik vorhanden ist, kann das Sprachmodell trainiert werden. Das Training ist für die Verbesserung der Spracherkennung notwendig. Es gibt drei Möglichkeiten das Sprachmodell zu trainieren:

- Einzelworttraining
- Satztraining
- Texttraining

Bevor der Spracherkenner überhaupt was erkennen kann, muss das zu erkennende Wort mindestens 10-mal aufgenommen worden sein. Dann kann das System als sprecherabhängig angesehen werden. Für ein Sprecherunabhängiges System werden Aufnahmen des gesamten Vokabulars von etwa 100 Sprechern empfohlen. Die Anzahl der Aufnahmen zu jedem Wort finden Sie in der Spalte *Erkennungsrate* im dem *Vokabular*-Menü.

Einzelworttraining

Gehen Sie in den Reiter Vokabular. Nun wählen Sie das Wort aus, welches trainiert werden soll und bestätigen die Auswahl mit dem Button *Zum Training hinzufügen*. Mit dem Button *Trainiert die ausgewählten Wörter* wird das Training gestartet. Das Einzelworttraining eignet sich nur für Systeme die einzelne Kommandos erkennen sollen. Für unseren Fall ist es also nicht nötig Einzelworttraining zu betreiben. Es wird erst dann wichtig, wenn die Erkennungsraten von bestimmten Wörtern verbessert werden sollen. Dazu mehr im Abschnitt 6.3.

Satztraining

Anders als das Einzelworttraining eignet sich das Satztraining durchaus für unser Projekt. Nach der Auswahl der Wörter, die trainiert werden soll, muss man das Häkchen
Sätze generieren setzen. Beim Training generiert Simon nun Sätze aus den ausgewählten Wörtern, die auf die definierte Grammatik zutreffen. Das ermöglicht dem System die Erkennung von Wortketten.

Texttraining

Mindestens genau so mächtig wie das Satztraining ist das Texttraining. Dabei wird das Training anhand eines Textes oder einer Textdatei generiert. Da bereits alle Wortkombinationen in eine Textdatei geschrieben sind (siehe Abschnitt 6.1.3) können diese in das Training importiert werden. Dazu rufen Sie im *Training*-Menü über den Button *Text hinzufügen* den Import-Assistenten auf und importieren dort die vorhandene Textdatei. Nach dem Importieren kann das Training des gesamten Vokabulars und Wortkombinationen mit *Start Training* gestartet werden. Das Texttraining ist deshalb die beste und effizienteste Methode um ein Sprachmodell zu trainieren.

6.1.5. Modell testen

Nachdem das Sprachmodell hinreichend trainiert ist kann der erste Spracherkennungsversuch mit Simon gestartet werden. Zuerst muss Simon beigebracht werden die erkannten Wörter als Tastatureingaben auszugeben. Dazu befolgen Sie folgende Schritte um das *Diktion* Plug-In zu aktivieren:

Menü *Aktionen verwalten* aufrufen : *Kommandos* → *Plug-Ins verwalten*

Plug-In *Diktion* hinzufügen : *Hinzufügen* → *Diktion* → *OK*

Jetzt kann man die Spracherkennung mit Simon durch die Schaltfläche *Aktivieren* starten. Öffnen Sie einen beliebigen Texteditor (Notepad, gedit, etc.) und sprechen einige Äußerungen aus dem trainierten Vokabular. Wenn alles richtig eingestellt ist und die Erkennungsrate für die Vokabeln bei mindestens 10 Aufnahmen liegt, sollte Simon die erkannten Worte in den Texteditor als Tastatureingabe ausgeben. Damit lässt sich das Sprachmodell auf seine Qualität testen bevor es mit in dem SCITOS-Sprachinterface eingesetzt wird.

6.2. Sprachmodell mit dem SCITOS-Sprachinterface nutzen

In diesem Abschnitt ist zu lesen, wie man das erzeugte oder schon vorhandene Sprachmodell in das SCITOS-Sprachinterface importiert. Da Simon nur ein Client ist, läuft die Verwaltung über den Server simond. Simond speichert die Sprachmodelle abhängig von dem Betriebssystem in:

Windows

```
%appdata%\.kde\share\apps\simond\models\USERNAME\active
```

Linux

```
~/.kde/share/apps/simond/models/USERNAME/active
```

Ein Sprachmodell besteht aus dem Hidden Markov Model und der dazugehörigen Grammatik. Die HMMs sind implementiert in den Dateien

`hmmdefs`

`hmmdefs_loc`

`tiedlist`

und die Grammatik in

`model.dfa`

`model.dict`

`model.grammar`

`simple.voca`

Für den Import in das SCITOS-Sprachinterface kopieren Sie die HMM-Dateien in Ordner `SCITOSsi\hmm` und die Grammatik-Dateien in `SCITOSsi\grammar`. Damit wäre der Import in wenigen Schritten abgeschlossen. Jetzt können Sie die Spracherkennung durch die Eingabe des folgenden Befehls starten:

```
[fed@localhost KIARA Demo]# SCITOSsi -C kiara.conf
```

Die Datei `kiara.conf` ist eine Konfigurationsdatei für die Julius-Engine. Die Konfigurationsparameter sind intern ausführlich beschrieben. Weitere Informationen finden Sie in dem Julius Book [Lee10].

6.3. Optimierung

Um die Qualität der Spracherkennung zu verbessern werden in dieser Sektion einige Möglichkeiten vorgestellt. Die Optimierung kann auf folgende Bereiche angesetzt werden:

- richtige Sprachaufnahme
- effizientes Training
- angepasste Grammatik
- optimale Konfiguration

Bei **Sprachaufnahmen** achten Sie darauf, dass sie zu dem VoiceTracker einen Mindestabstand von einem Meter einhalten. Sprechen sie dabei ganz normal, nicht zu laut, leise, schnell oder langsam. Eine verständliche Aussprache erhöht die Qualität der Erkennung. Eine Probeaufnahme kann helfen die richtige Aufnahmeposition und Aufnahmelautstärke zu finden. Nutzen Sie dafür ein Aufzeichnungstool Ihrer Wahl und nehmen eine Äußerung auf. Eine optimale Aufnahmelautstärke ist in Abbildung 6.1 gezeigt. [vlg. Gra09, S. 16]

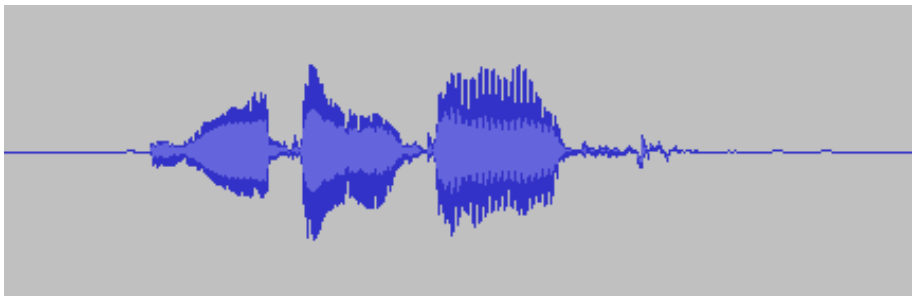

Abbildung 6.1.: Optimale Lautstärke einer Sprachaufnahme

Achten Sie darauf, dass vor und nach einer Äußerung eine Stille von etwa 2 Sekunden folgen muss.

Versuchen Sie nach jeder **Training**seinheit die Erkennungsrate durch das Aktivieren der Spracherkennung in Simon zu testen. Eine weitere Möglichkeit bietet das im Simon Packet enthaltende Tool sam. Mit sam lassen sich die vorhandenen Sprachmodelle auf ihre Erkennungsraten prüfen. Nach dem Start von sam wird über die Schaltfläche
Verändere simons Modell das aktuelle Sprachmodell geladen. Mit dem Button *Modell testen* kann das Sprachmodell auf die Qualität überprüft werden. Sollte bei einem Wort oder Wortkette eine niedrige Erkennungsrate, meist kleiner als 50%, vorhanden sein, so muss dies trainiert werden. Dazu muss man wieder ein Training mit Simon auf das bezogene Wort oder Wortkette anwenden.

Eine weitere Methode die Spracherkennung zu verbessern ist es eigene Terminale in die **Grammatik** einzubringen. In dem Demo-Projekt KIARA sind folgende Terminale aus der Tabelle 6.4 eingepflegt.

ACT	Aktoren
CMD	Kommandos
ZSTD	Zustand, z.B. ein / aus
SENS	Sensoren
DIRECT	Richtung (Direction)
PDIRCET	Präfix der Richtung (ins,nach)
ORT	Labor, Raum,...
NAM	Name

Tabelle 6.4.: Terminale des KIARA Demo-Projekts

Das hat den Vorteil, dass Wortketten eindeutiger identifiziert werden können. Nehmen wir dazu die Äußerung „kiara laser aus“ als Beispiel. Demnach wäre die Grammatik nach den HADIFIX BOMP Terminalen `NAM NOM ADD` und nach den neu erstellten `NAM SENS ZSTD`. Tabelle 6.5 auf Seite 69 zeigt den Vergleich der beiden Grammatiken. Die KIARA Terminale erlauben die präzisere Definition der Grammatik und gleichzeitig eine Steigerung der Erkennungsrate. Nach der Grammatik mit
HADIFIX BOMP Terminalen werden Wortketten gültig die nach dem KIARA Zustandsdiagramm nicht akzeptiert werden oder sogar unvorhergesehenes Verhalten hervorgerufen werden kann.

Um Störgeräusche auf Softwarebasis zu vermeiden kann man mit **Konfigurationsparametern** in einer `.jconf`-Datei die Empfindlichkeit und Akzeptanz eines Signals steuern. Die wichtigsten sind [vgl. Lee10, S. 23]:

`-lv` *`tres`*
: Steuert den Schwellwert für den Energiepegel der zwischen 0 und 32767 liegen kann. Voreinstellung ist der Wert 2000. Abhängig von den Raumbedingungen kann beim hohen Aufkommen von Geräuschen der Wert höher gewählt werden.

`-rejectshort` *`msec`*
: Sollte die Äußerung oder das eigentliche Signal kürzer als die angegebene Zeit in Millisekunden sein, wird der Input ignoriert und die Erkennung abgebrochen.

`-powerthres` *`thres`*
: Dieser Parameter bestimmt die Energie die mindestens vorhanden sein muss, bevor ein Erkennungsversuch gestartet wird. Beachten Sie, dass die Option `-enable-power-rejec` bei der Kompilierung angegeben werden muss.

Grammatik

NAM	SENS	ZSTD
kiara	ultraschall	aus
kiara	ultraschall	an
kiara	ultraschall	einschalten
kiara	ultraschall	ausschalten
kiara	laser	an
kiara	laser	aus
kiara	laser	einschalten
kiara	laser	ausschalten
NAM	NOM	ADD
kiara	motor	an
kiara	motor	aus
kiara	motor	einschalten
kiara	motor	ausschalten
kiara	batterie	an
kiara	batterie	aus
kiara	batterie	einschalten
kiara	batterie	ausschalten
kiara	ultraschall	aus
kiara	ultraschall	an
kiara	ultraschall	einschalten
kiara	ultraschall	ausschalten
kiara	laser	an
kiara	laser	aus
kiara	laser	einschalten
kiara	laser	ausschalten
etc...		

Tabelle 6.5.: Vergleich zwischen HADIFIX BOMP und KIARA Terminalen

7. Auswertung und Aussicht

Nachdem das Sprachinterface fertiggestellt worden ist konnte innerhalb einer kurzen Zeit und intensiven Training eine sprecherabhängige Erkennungsrate von 95,91% erreicht werden. Abbildung 7.1 zeigt die Erkennungsergebnisse für das Sprachmodell KIARA. So ist eine sprecherunabhängige Spracherkennung bei etwa 100 Sprechern möglich. Bis auf die optimale Geräuschfiltrierung sind alle aufgestellten Anforderungen an ein mobiles Sprachinterface erfüllt worden.

Die Software ist auf einem virtuellen PC implementiert und getestet worden, deshalb wäre die Installation und Ausführung auf dem SCITOS G5 System noch zu überprüfen. Das sollte allerdings kein Problem darstellen, weil das virtuelle Betriebssystem nach SCITOS G5 Vorgaben angepasst ist. Zudem ist auch das Demo-Projekt KIARA enthalten, das nach wenigen Installationsschritten einsatzbereit ist. Für die Steuerung der SCITOS-Aktoren müssen noch die metralabs-Bibliotheken eingebunden werden und die Zustände dementsprechend angepasst sein.

Leider konnten keine Geräuschproben unter realen Arbeitsbedingungen aufgezeichnet werden, da der Service-Roboter noch nicht voll einsatzbereit gewesen ist. Deshalb muss dieser Bereich weiter untersucht werden. Bei Einzelaufnahmen von Störgeräuschen jedoch zeigte der eingesetzte VoiceTracker gute Filtereigenschaften. Für den Betrieb des VoiceTrackers muss auf dem SCITOS G5 noch eine Spannungsquelle von 6 V realisiert werden.

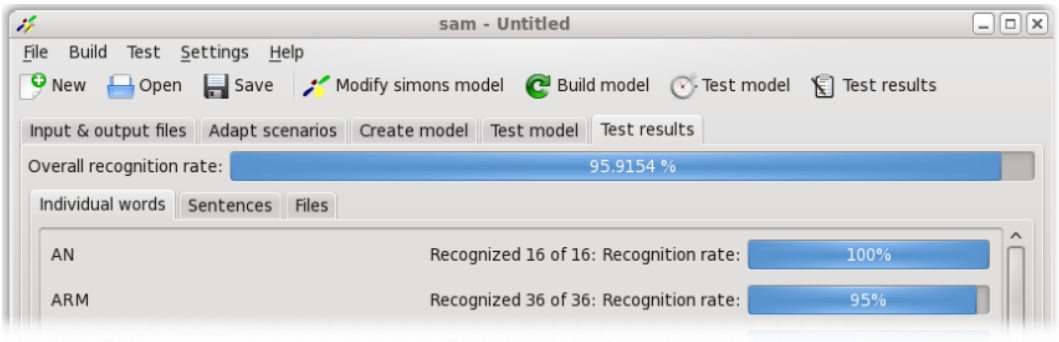

Abbildung 7.1.: Sprachmodell-Evaluierung mit dem Tool *sam*

Literaturverzeichnis

[AKG] AKG: *C 3000 B*. http://www.buyfromchucks.com/images/products/small/C3000B.jpg, Abruf: 04.01.2011

[AT&] AT&T: *AT&T Natural Voices*. http://www.wizzardsoftware.com/speech_overview.php, Abruf: 25.11.2010

[Bur] BURKHARDT, Felix: *Deutsche Sprachsynthese*. http://ttssamples.syntheticspeech.de/deutsch/index.html, Abruf: 25.11.2010

[BWA+05] BREUER, Stefan ; WAGNER, Petra ; ABRESCH, Julia ; BRÖGGELWIRTH, Jörg ; ROHDE, Hannes ; STÖBER, Karlheinz: *Bonn Open Synthesis System (BOSS) 3 - Documentation and User Manual*. Institut für Kommunikationsforschung und Phonetik, 2005. – 49 S. http://www.sk.uni-bonn.de/forschung/phonetik/sprachsynthese/boss/BOSS_Documentation.pdf

[Cam] CAMBRIDGE.UNIVERSITY.ENGINEERING.DEPARTMENT: *HTK Speech Recognition Toolkit*. http://htk.eng.cam.ac.uk/, Abruf: 23.11.2010

[Car] CARNEGIE.MELLON.UNIVERSITY: *CMU Sphinx Speech Recognition Toolkit*. http://cmusphinx.sourceforge.net/, Abruf: 23.11.2010

[Eul06] EULER, Stephen: *Grundkurs Spracherkennung - Vom Sprachsignal zum Dialog - Grundlagen und Anwendung verstehen*. Veiweg-Verlag, 2006. – 200 S. http://www.springerlink.com/content/978-3-8348-0003-9. – ISBN 978–3–8348–0003–9

[Gra09] GRASCH, Peter: *Das simon Handbuch*. „simon listens“ gemeinnütziger Verein zur Entwicklung, Verbreitung und Implementierung der Spracherkennungssoftware simon, 2009. – 112 S. http://simon.gibolles.com/doc/0.3/simon/de/index.pdf

[HKK05] HEROLD, Helmut ; KLAR, Michael ; KLAR, Susanne: *C++, UML und Design Patterns*. Addison-Wesley, 2005. – 1176 S. – ISBN 978–3–827–32267–8

[Lee10] LEE, Akinobu: *The Julius book*. 2010. – 67 S. http://jaist.dl.sourceforge.jp/julius/47534/Juliusbook-4.1.5.pdf

[Lin] LINGUATEC: *Linguatec - Voice pro 12*. http://www.linguatec.net/products/stt/voice_pro, Abruf: 23.11.2010

[Met10] METRALABS: *SCITOS G5 - Embedded PC and Operating System*. 2010. – Benutzerhandbuch

[Nag] NAGOYA.INSTITUTE.OF.TECHNOLOGY: *Open-Source Large Vocabulary CSR Engine Julius*. http://julius.sourceforge.jp/en_index.php, Abruf: 24.11.2010

[Nuaa] NUANCE: *Nuance - Produkte*. http://www.nuance.de/produkte/, Abruf: 11.11.2010

[Nuab] NUANCE: *Nuance Vocalizer*. http://www.nuance.de/news/20090825_vocalizer5.asp, Abruf: 24.11.2010

[PK08] PFISTER, Beat ; KAUFMANN, Tobias: *Sprachverarbeitung - Grundlagen und Methoden der Sprachsynthese und Spracherkennung*. Springer-Verlag, 2008. – 483 S. http://www.springerlink.com/content/978-3-540-75909-6. – ISBN 978–3–540–75909–6

[PWU] PHILOSOPHISCHE.FAKULTÄT.DER.RHEINISCHEN.FRIEDRICH-WILHELMS-UNIVERSITÄT.BONN: *Sprache und Kommunikation*. http://www.sk.uni-bonn.de/forschung/phonetik/sprachsynthese/bomp, Abruf: 02.12.2010

[RB08] RÄHM, Jan ; BENTHIN, Falko: Zwiegespräch - SPRACHERKENNUNGSSYSTEM SIMON LISTENS. In: *LinuxUser* (2008), 07. – http://www.linux-community.de/Internal/Artikel/Print-Artikel/LinuxUser/2008/07/Zwiegespraech, Abruf: 03.12.2010

[SK06] STUART, Herbert ; KLAGES, Gerhard: *Kurzes Lehrbuch der Physik*. Springer-Verlag, 2006. – 324 S. http://www.springerlink.com/content/978-3-540-23146-2/. – ISBN 978–3–540–23146–2 (Print) 978–3–540–29728–4 (Online)

[Sl] SIMON-LISTENS: *simon listens - Gemeinsamer Verein für Forschung und Lehre*. http://www.simon-listens.org/, Abruf: 11.11.2010

[Tas] TASCAM: *US-122*. http://tascam.de/pics/us-122.jpg, Abruf: 04.01.2011

[TCT] TCTS.LAB: *The MBROLA Project*. http://tcts.fpms.ac.be/synthesis/mbrola.html, Abruf: 26.11.2010

[Tes] TESTBERICHTE.DE: *Testberichte.de - Nuance Dragon NaturallySpeaking Version 11*. http://www.testberichte.de/p/nuance-tests/dragon-naturallyspeaking-version-11-testbericht.html, Abruf: 03.01.2011

[Wen05] WENDEMUTH, Andreas: *Grundlagen der digitalen Signalverarbeitung - Ein mathematischer Zugang*. Springer Verlag, 2005. – 268 S. http://www.springer.com/engineering/signals/book/978-3-540-21885-2. – ISBN 978–3–540–21885–2

[WLK+04] WALKER, Willie ; LAMERE, Paul ; KWOK, Philip ; RAJ, Bhiksha ; SINGH, Rita ; GOUVEA, Evandro ; WOLF, Peter ; WOELFEL, Joe: *Sphinx-4: A Flexible Open Source Framework for Speech Recognition*. SUN Microsystems Inc., 2004 http://labs.oracle.com/techrep/2004/smli_tr-2004-139.pdf

[YEK+01] YOUNG, Steve ; EVERMANN, Gunnar ; KERSHAW, Dan ; MOORE, Gareth ; ODELL, Julian ; OLLASON, Dave ; VALTCHEV, Valtcho ; WOODLAND, Phil: *The HTK Book*. Microsoft Corporation, Cambridge University Engineering Department, 2001. – 284 S. http://htk.eng.cam.ac.uk/prot-docs/htk_book.shtml

A. CD Inhalt

```
/
├── DOKUMENTATION ... Dokumentation im PDF und HTML Format
│   ├── BILDER ... In der Arbeit eingesetzte Bilder
│   ├── HTK
│   ├── JULIUS
│   ├── MBROLA
│   ├── QUELLEN ... Internetquellen im PDF Format
│   ├── SIMON
│   ├── SPRACHINTERFACE
│   │   └── html
│   └── SPRECHERLOKALISIERUNG
│       ├── power_locate
│       └── xcorr_locate
├── SOFTWARE ... Installationsdateien und Quellcode der
│                eingesetzten Programme
│   ├── HTK
│   │   ├── Linux
│   │   └── win32
│   ├── JULIUS
│   │   └── QUICKSTART
│   ├── MBROLA
│   │   ├── bin
│   │   ├── DE-STIMMEN
│   │   └── txt2pho
│   ├── SIMON
│   ├── SPRACHINTERFACE
│   │   ├── KIARA-Sprachmodell
│   │   └── SCITOSsi
│   └── SPRECHERLOKALISIERUNG
│       ├── power_locate
│       └── xcorr_locate
└── BA_Sulz.pdf
```

Glossar

BOSS Bonn Open Synthesis System

DFT Diskrete Fourier-Transformation

DTW Dynamic Time Warp (Dynamische Zeitverzerrung)

HAW Hochschule für Angewandte Wissenschaften Hamburg

HMM Hidden Markov Modell

HTK Hidden Markov Toolkit

KIARA Der Name für den SCITOS G5 zusammengesetzt aus: **K**ommunikativer **i**nteraktiver **A**ssistenz-**R**oboter**a**rm

LDS Location Dependent Squelch (Richtungsabhängige Rauschsperre)

LVCSR Large-Vocabulary Continuous Speech Recognition

MFCC Mel Frequency Cepstral Coefficients

TTS Text-to-Speech

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 17. Januar 2011	
Ort, Datum	Unterschrift