Bedienung einer Getränkemischmaschine über Sprachbefehle

Studienarbeit

für die Prüfung zum

Bachelor of Science

des Studienganges Angewandte Informatik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Felix Manuel Gervasi

und

Alena Sutiagina

Abgabedatum 22. Mai 2023

Bearbeitungszeitraum	6 Monate
Matrikelnummer	1052491 (Gervasi)
	6235629 (Sutiagina)
Kurs	TINF20B4
Betreuer der Studienarbeit	Prof. Dr. Jörn Eisenbiegler

Erklärung

Ich versichere hiermit, dass ich meine Studienarbeitmit dem Thema: »Bedienung einer Getränkemischmaschine über Sprachbefehle« selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort Datum

Unterschrift

Zusammenfassung

Ziel der Arbeit ist die Implementierung einer Sprachsteuerung für eine Mischmaschine, die Getränke zu unterschiedlichen Mischungsverhältnissen aus vier verschiedenen Behältern mischen kann. Die Mischmaschine soll dabei nicht nur die Sprache des Benutzers erkennen und daraufhin Aktionen durchführen sondern auch Antworten, sodass eine Art von Dialog zwischen Mischmaschine und Benutzer entsteht. Um das Ziel zu erreichen werden zunächst die theoretischen und technischen Grundlagen erarbeitet, wobei insbesondere auf die Verarbeitung von Sprache mit Hilfe von Künstlicher Intelligenz eingegangen wird. Anschließend werden die Anforderungen an das Projekt konkretisiert. Verschiedene Konzepte werden vorgestellt und evaluiert. Ein Konzept sieht vor zur Implementierung der Sprachsteuerung ausschließlich die bereits in der Mischmaschine verbaute Hardware zu verwenden. Ein weiteres Konzept sieht die Benutzung einer mobilen Anwendung vor, um die vorhandenen Rechenkapazitäten zu erweitern. Das dritte Konzept erklärt, wie Computer-Hardware eingesetzt werden kann, um die Sprachsteuerung zu implementieren. Außerdem werden verschiedene Ansätze für das Dialogsystem diskutiert. Es wird erläutert warum die Wahl für die Implementierung schließlich auf das soeben genannte, dirtte Konzept fällt. Ergebnis der Arbeit ist eine Sprachsteuerung für die Mischmaschine, die mit Hilfe eines Raspberry Pi 4 Modell B in Kombination mit dem, in der Mischmaschine verbauten, Arduino Mega funktioniert. Der Benutzer hat die Möglichkeit über ein Mikrofon, das an den Raspberry Pi angeschlossen ist, mit der Maschine zu kommunizieren. Die über das Mikrofon aufgenommenen Audisognale werden zunächst in Text konvertiert. Der Text wird von einem Sprachmodell interpretiert und klassifiziert. Ergebnis der Interpretation sind ein, für die Mischmaschine verständliches Kommando und die auszugebende Antwort. Das Kommando wird über eine serielle Schnittstelle vom Raspberry Pi an den Ardunino übertragen, der die Mischmaschine steuert. Die Mischmaschine führt daraufhin die gewünschte Aktion aus und der Raspberry Pi gibt die Antwort über Lautsprecher an den Benutzer zurück.

Abstract

The goal of this paper is the implementation of a voice control for a mixing machine that can mix beverages with different mixing ratios from four different containers. The mixing machine should not only recognize the user's speech and perform actions on it, but also answer, so that a kind of dialog between mixing machine and user is created. To achieve this goal, the theoretical and technical basics are worked out, especially the processing of speech with the help of artificial intelligence. Subsequently, the requirements for the project will be specified. Different concepts are presented and evaluated. A concept is to use only the hardware already installed in the mixing machine to implement the voice control. Another concept envisages the use of a mobile application to extend the existing computing capacity. The third concept explains how computer hardware can be used to implement voice control. Different approaches to the dialog system are also discussed. It is explained why the choice for the implementation finally falls on the just mentioned, third concept for the implementation. The result of the work is a voice control for the mixing machine, which is implemented using a Raspberry Pi 4 model B in combination with the Arduino Mega, which is installed in the mixing machine. The user has the possibility to communicate with the mixing machine via a microphone that is connected to the Raspberry Pi, to communicate with the machine. The audio signals are first converted into text. The text is interpreted and classified by a speech model. The result of the interpretation is a command that can be understood by the mixing machine and the response to be output. The command is sent via serial interface from the Raspberry Pi to the Ardunino, which controls the mixing machine. The mixing machine then executes the desired action and the Raspberry Pi returns the response to the user via loudspeaker.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Formelverzeichnis

Abkürzungsverzeichnis

Kapitel 1

Einleitung

Die Informationstechnik versteckt sich heutzutage fast überall - selbst dort, wo sie von den meisten Menschen nicht vermutet werden würde. Beispiele hierfür sind Autos, Kaffeemaschinen, Zahnbürsten, Rasierer, Küchengeräte und vieles mehr. Grund dafür ist die fortschreitende Möglichkeit der Miniaturisierung von Computern, sodass diese nahezu überall verbaut werden können. Beispielsweise können Mikrochips in der Kaffeemaschine dafür sorgen, dass die richtige Menge an Kaffee serviert wird oder der Füllstand der einzelnen Behälter angezeigt werden kann. Solche Systeme, die Informationen mit Hilfe eines Computers verarbeiten und dabei mit ihrer Umgebung derartig „verschmelzen“, nennt man auch *embedded systems* (z. Dt. *eingebettete Systeme*) [Marwedel 2021].

Der aktuelle Trend des Internet der Dinge (IoT) führt zu einem noch größeren Anstieg eingebetteter Systeme im Alltag. Im IoT geht es speziell um eingebettete Systeme, die internetfähig (vernetzt) sind. Nach Schätzungen des Marktforschungsunternehmens *Gartner* gab es im Jahr 2017 8,4 Milliarden solcher vernetzten Geräte weltweit [Jansen 2017]. Das die Menge der vernetzten Geräte als Teilmenge der eingebetteten Systeme betrachtet werden kann ist damit zu rechnen, dass deren Anzahl sogar weit größer ausfällt.

1.1 Aufgabenstellung

Im Rahmen dieser Arbeit geht es um die Sprachsteuerung einer Getränkemischmaschine, die in diesem Fall als eingebettetes System zu verstehen ist und in einer vorangegangenen Arbeit bereits konzipiert und gebaut wurde [Götz, Höckele und Lobert 2022]. Sie verfügt derzeit über ein Touch-Display zur Bedienung durch den Benutzer. Ziel der Arbeit ist es zusätzlich eine natürlichsprachliche Interaktion mit der Maschine zu ermöglichen, die mindestens den Funktionsumfang besitzt, der aktuell über die Bildschirmeingabe möglich ist. Dabei soll die Maschine nicht nur in der Lage sein die natrüliche Sprache des Benutzers in ein geeignetes Format umzuwandeln, sodass die Maschine den korrekten Befehl ausführt. Sie soll auch in der

Lage sein dem Benutzer zu Antworten, sodass die Illusion einer Konversation mit der Maschine entsteht.

1.2 Vorgehen

Zunächst müssen die Sprachverarbeitung und Spracherkennung betrachtet werden. Die Sprachverarbeitung dient der Interpretation des Gesprochenen um eine geeignete Antwort auszugeben und dem Übersetzen in einen Maschinenbefehl. Im Rahmen dieser Arbeit sollen dafür Verfahren und Techniken der Künstlichen Intelligenz (KI) und des Machine Learning (ML) eingesetzt werden. Die Spracherkennung beschäftigt sich mit der Aufnahme des Tonsignals bzw. der Schallwellen (bspw. über ein Mikrofon) und dem Umwandeln dieser Signale in Text, sodass dieser an das KI-Modell weitergereicht werden kann.

Bei der Arbeit mit eingebetteten Systemen muss man sich der vorhandenen Hardwareleistung und den benötigten Hardwareanforderungen bewusst sein, da diese meist sehr begrenzt ist. Deshalb werden im Rahmen dieser Arbeit verschiedene Ansätze diskutiert, wie und wo die einzelnen Schritte und Berechnungen ablaufen sollen (s. Kapitel 4).

Kapitel 2

Theoretische und technische Grundlagen

Dieses Kapitel beschäftigt sich mit der Erarbeitung theoretischer und technischer Grundlagen die im weiteren Verlauf der Arbeit angewendet wurden.

2.1 Getränkemischmaschine

Ziel dieser Arbeit ist die Implementierung einer Sprachsteuerung für eine Getränkemischmaschine. Die Getränkemischmaschine wurde bereits in einem vorangegangenen Projekt erstellt [Götz, Höckele und Lobert 2022]. In diesem Abschnitt wird darauf eingegangen, um was für eine Art von Maschine es sich dabei handelt und es werden ihre Funktionsweise und ihr Aufbau beschrieben.

Das Mischen von Getränken bzw. Flüssigkeiten ist der Anwendungsfall für den die Maschine konzipiert wurde. Dazu besitzt die Mischmaschine fünf Behälter zu je einem Liter. Jedem Behälter ist eine Pumpe zugeordnet, die separat angesteuert werden kann und dafür sorgt, dass die Flüssigkeit aus dem Behälter zur Getränkeausgabe gelangt. Kurz vor dem Ausgang werden die Schläuche der Behälter zusammengeführt, wodurch letztlich die Mischung der verschiedenen Getränke erzielt wird. Sogenannte „Rückschlagventile“ sorgen dafür, dass ein Zurückfließen der Getränkemischung in die Behälter verhindert wird.

Zur Steuerung der Maschine durch einen Benutzer befinden sich an der Vorderseite zwei Knöpfe und ein Touch-Display. Einer der Knöpfe dient dem Anschalten der Maschine und ein weiterer der Ausgabe des Getränks. Über das Touch-Display kann der Benutzer die Mischung seiner Getränke konfigurieren und die Maschine administrieren.

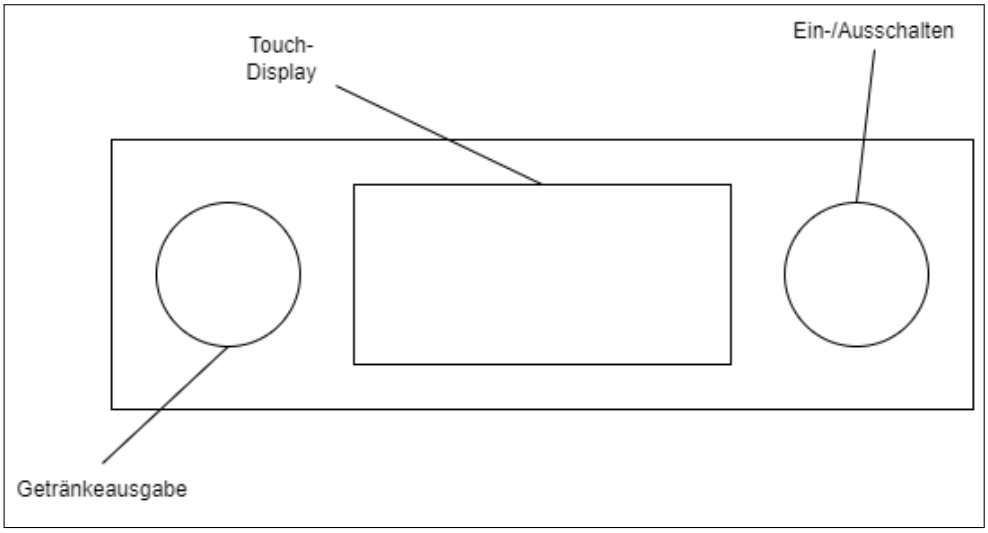

Abbildung 2.1: Schematischer Aufbau der Bedienungsschnittstelle

Abbildung 2.1 stellt den schematischen Aufbau der Bedienungsschnittstelle der Getränkemischmaschine für ein besseres Verständnis dar.

Die technische Umsetzung basiert auf der Kommunikation zwischen dem Touch-Display und einem, in der Getränkemischmaschine verbauten, Arduino, der anhand der Daten von der Benutzereingabe, die Pumpen steuert. Beim Drücken des Startknopfes werden das Display und der Arduino mit Strom versorgt, sodass sowohl das Display als auch der Arduino starten und mit der Ausführung des benutzerdefinierten Quelltextes beginnen. Auf dem Bediendisplay finden sich fünf Schieberegler - ein Schieberegler je Behälter - mit denen der Benutzer die Zusammensetzung seines Mischgetränks aus den fünf Behältern konfigurieren kann. Beim Drücken eines in der User Interface (UI) des Displays dargestellten Knopfes werden die Werte der Schieberegler an den Arduino übertragen. Dieser rechnet die Prozentwerte der Schieberegler in Durchsatzraten für die Pumpen um. Wird anschließend der Knopf für die Getränkeausgabe gedrückt gehalten steuert der Arduino die Pumpen mit ihrer jeweiligen Durchsatzrate an und der Anwender bekommt sein Mischgetränk ausgegeben.

Neben den bereits genannten Komponenten besitzt die Mischmaschine außerdem einen Wasserstandssensor je Behälter, um ein Trockenlaufen der Pumpen zu verhindern. Wie die einzelnen Komponenten zusammenspielen ist in dem Anhang D zu sehen, der den schematischen Aufbau/-Schaltplan der Getränkemischmaschine zeigt [Götz, Höckele und Lobert 2022].

2.2 Hardware

2.2.1 Arduino

Allgemeines

Arduino ist eine Plattform/Organisation, die sich der Herstellung von Mikrocontrollern und der für den Umgang mit der Hardware erforderlichen Software widmet. Darunter fallen in erster Linie die Arduino-IDE und -Sprache zur Programmierung der Arduino-Mikrocontroller. Arduino verfolgt das Ziel, Hardwareprogrammierung oder Informationstechnologie (IT) im Allgemeinen für alle Menschen unabhängig ihres Hintergrunds zu ermöglichen und näher zu bringen - allen voran auch denjenigen, die sich nicht professionell mit IT auseinandersetzen [Arduino o. D.[a]]. Deshalb liegt der Fokus von Arduino auf preisgünstigen Mikrocontrollern und einer einfachen Handhabung. Arduino stellt zu diesem Zweck auch viele Lernmaterialien bereit, die bspw. von Lehrkräften verwendet werden können als auch eine Community-Plattform für den Wissensaustausch zwischen den Benutzern von Arduino Hard- und Software. Pläne für die Arduino-Boards werden unter der *Creative Commons* Lizenz veröffentlicht, sodass sie von anderen nachgebaut oder erweitert werden können. Auch der Quelltext der Arduino-Software ist quelloffen.

Aufgrund des großen Erfolgs von Arduino gibt es inzwischen zahlreiche Nachahmungen wie bspw. ...

Arduino-Programmierung

Die Programmierung eines Arduino-Mikrocontrollers erfolgt über die Arduino-IDE. Die Arduino-IDE kann auf allen gängigen Betriebssystemen (Windows, Mac und Linux) installiert und verwendet werden, was den Umgang mit Arduino sehr flexibel und gut für Einsteiger macht, da sie ihre gewohnte Umgebung verwenden können. Die Arduino-IDE ist dabei angelehnt an andere, gängige Entwicklungsumgebungen, sodass sich erfahrene Programmierer sofort zurecht finden, ist aber auf die Arduino-Programmierung optimiert. Beispielsweise werden an den Computer angeschlossene Boards automatisch erkannt, per Knopfdruck kann der aktuelle Quelltext auf den Mikrocontroller hochgeladen werden und die evtl. zusätzlich benötigten C++-Bibliotheken können verwaltet werden. Neben der Desktop-Version kann die Arduino-Entwicklungsumgebung auch online im Browser verwendet werden. Abbildung 2.2 zeigt einen Ausschnitt der Benutzeroberfläche in der Arduino-Entwicklungsumgebung.

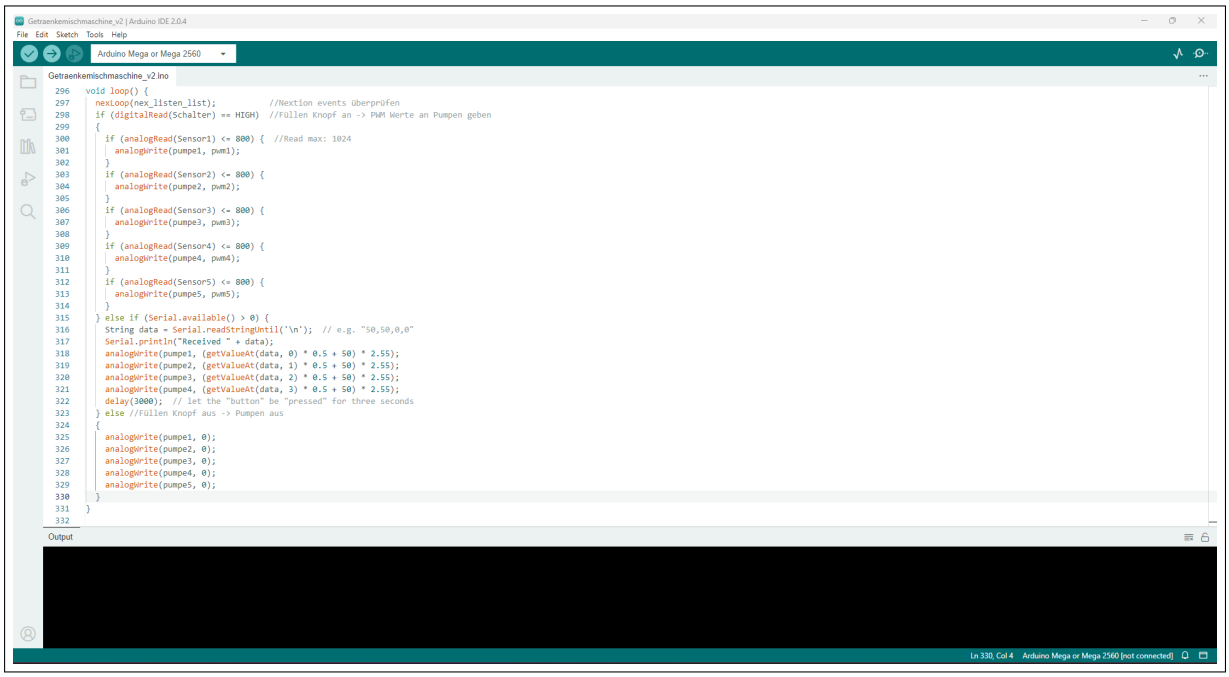

Abbildung 2.2: Arduino-IDE

Ein Arduino-Mikrocontroller wird programmiert, indem das Programm in der Arduino-IDE und -Sprache geschrieben wird. Anschließend kann der Mikrocontroller über die USB-Schnittstelle an den Computer angeschlossen und der Quellcode durch Knopfdruck in der Entwicklungsumgebung an den Mikrocontroller gesendet werden, der sofort mit der Ausführung des Programms beginnt. Die meisten Mikrocontroller unterstützen kein Debugging um den Entwickler bei der Programmierung und Fehlersuche zu unterstützen. Manche Modelle verfügen jedoch über einen in Hardware implementierten Debugger zu diesem Zweck, der ebenfalls über die Entwicklungsumgebung verwendet werden kann.

Die Arduino-Programmiersprache ist angelehnt an *Wiring* [Wiring o. D.] und kann durch eigene C++-Bibliotheken erweitert werden. Die Programmstruktur ist fest vorgegeben. Jedes Arduino-Programm besteht aus einer *setup* und einer *loop* Funktion. Die *setup* Funktion wird exakt einmal zu Beginn des Programms aufgerufen, also wenn der Arduino mit Strom versorgt wird. Die *loop* Funktion stellt die Hauptbefehlsschleife dar und enthält den Code, der den Großteil der Programmlogik enthält. Sie wird in einer Dauerschleife immer wieder neu aufgerufen. Listing 2.1 stellt den Aufbau eines Arduino-Programms dar.

```
1 void setup()
2 {
3     // Setup code
4 }
```

```
6 void loop()
7 {
8     // Programmlogik
9 }
```

Listing 2.1: Aufbau eines Arduino-Programms

Die Arduino-Sprache stellt einige Sprachelemente bereit, die den Umgang mit der zugrundeliegenden Hardware ermöglichen und erleichtern. Mit Sprachelementen sind sowohl innerhalb des Codes zugreifbare, globale Funktionen als auch Objekte gemeint. Abgesehen von *setup* und *loop* sind die wichtigsten Sprachelemente [Arduino o. D.[c]]:

- *HIGH* und *LOW*:
- *INPUT* und *OUTPUT*:
- *digitalRead(pin)* und *digitalWrite(pin, value)*: Wird verwendet um einen der digitalen Ein-/Ausgabe-Pins des Arduinos zu Lesen oder zu Schreiben. Dabei können nur die Werte 0 oder 1 bzw. *LOW* oder *HIGH* gelesen und geschrieben werden. Das Argument *pin* bezieht sich auf die Nummer des Pins, auf den zugegriffen werden soll.
- *pinMode(pin, mode)*: Diese Funktion wird meist in der *setup* Routine eines Arduino-Programms verwendet und registriert einen bestimmten Pin als *INPUT* oder als *OUTPUT* Pin, d.h. als Pin, der entweder lesend oder schreibend zugegriffen wird.
- *analogRead(pin)* und *analogWrite(pin, value)*: Diese beiden Funktionen können, im Gegensatz zu *digitalRead* und *digitalWrite*, verwendet werden um analoge Werte, d.h. Werte die in einem bestimmten Bereich liegen und sich von 0 und 1 unterscheiden können, zu lesen und zu schreiben. *pin* bezeiht sich hierbei auf die Nummer eines der analogen Pins des Arduino-Mikrocontrollers. Die verfügbare Auflösung ist von Modell zu Modell unterschiedlich. Die Auflösung bestimmt die Größe des Wertebereichs (?)...
- *delay(ms)*: Verzögert die Ausführung des Programms um *ms* Millisekunden.
- *Serial*:

Elegoo Mega 2560 R3

Bei dem *Elegoo Mega 2560 R3* handelt es sich um eine Nachahmung des Arduino Mega 2560 R3. Der *Elegoo Mega 2560 R3* ist der, in der Mischmaschine eingebaute, Mikrocontroller und soll im Folgenden kurz vorgestellt werden.

2.2.2 Nextion

2.2.3 Raspberry Pi

2.3 Sprachverarbeitung

Im Zuge des technologischen Fortschritts nutzen die Menschen heutzutage zunehmend Sprachassistenten für verschiedene Aufgaben. Einer der Hauptvorteile der Sprachsteuerung ist die Bequemlichkeit und Geschwindigkeit, mit der Aufgaben erledigt werden können, ohne dass man tippen oder mit der Maus klicken muss. Sprachassistenten nutzen die Verarbeitung natürlicher Sprache, um Befehle zu erkennen und zu verstehen, die der Nutzer laut ausspricht.

Die Verarbeitung natürlicher Sprache - Natural Language Processing (NLP) - ist eine wichtige Technologie, die es Computern ermöglicht, die von Menschen verwendete natürliche Sprache zu verstehen. Diese Technologie ermöglicht es Computern, menschliche Sprache zu erkennen und zu interpretieren und Text und Sprache in natürlicher Sprache zu erzeugen. Im Bereich der Sprachsteuerung spielt NLP eine Schlüsselrolle bei der Erkennung von Sprache und dem Verstehen von Befehlen, die der Benutzer laut ausspricht. Es wird verwendet, um die Sprache des Benutzers in Text umzuwandeln, den ein Computer verstehen und verarbeiten kann. Um dies zu erreichen, verwendet NLP eine Vielzahl von Techniken und Technologien, darunter maschinelles Lernen, Tonanalyse, syntaktische Analyse und mehr. [Jurafsky u. a. 2009]

Im Rahmen dieses Projekts erfordert die Implementierung der Sprachsteuerung einer Getränkemischmaschine die Verarbeitung natürlicher Sprache, damit die Maschine Befehle verstehen kann, die der Benutzer laut ausspricht. Dieses Projekt ähnelt einem Chatbot, bei dem der Benutzer eine Frage stellen oder einen Befehl geben kann und der Chatbot führt die entsprechende Aktion aus. Die Verarbeitung natürlicher Sprache ist für die Entwicklung eines solchen Sprachsteuerungssystems unerlässlich und ermöglicht es der Maschine, Befehle in natürlicher Sprache zu verstehen und auszuführen.

Eine der wichtigsten Komponenten der Verarbeitung natürlicher Sprache ist die Spracherkennung und das Syntaxanalyseverfahren. Bei der Spracherkennung kommen Deep-Learning-Techniken zum Einsatz, die es einem Computer ermöglichen, Sprachlaute zu erkennen und in Text zu übersetzen. Anschließend wird das Syntaxanalyseverfahren verwendet, um die Satzstruktur zu bestimmen und Schlüsselwörter und -sätze hervorzuheben, die zur Bestimmung des Benutzerbefehls verwendet werden können. Auch die Tonwertanalyse ist ein wichtiger Bestandteil der Verarbeitung natürlicher Sprache. Mit Hilfe der Tonalitätsanalyse lässt sich die emotionale Färbung des Textes bestimmen, was für die Ermittlung der Absicht des Nutzers nützlich sein kann. Da die Tonwertanalyse im Rahmen dieser Arbeit nicht relevant ist, wird sie nicht weiter

erörtert.

Darüber hinaus werden grammatik- und regelbasierte Technologien zur Verarbeitung natürlicher Sprache eingesetzt. Diese Technologien werden eingesetzt, um die korrekte Struktur des Benutzerbefehls zu bestimmen und Schlüsselwörter hervorzuheben, die zur Durchführung von Aktionen verwendet werden können. Außerdem werden Techniken des maschinellen Lernens eingesetzt, damit der Computer aus früheren Befehlen und Aktionen „lernen“ kann, was die Genauigkeit der Erkennung von Benutzerbefehlen verbessert.

Daher ist der Einsatz von Technologien zur Verarbeitung natürlicher Sprache für das Projekt der Sprachmischmaschine unerlässlich. Die Verarbeitung natürlicher Sprache wird es der Maschine ermöglichen, die Befehle zu verstehen und zu verarbeiten, die der Benutzer laut ausspricht und die entsprechenden Aktionen durchzuführen.

2.3.1 Verarbeitung natürlicher Sprache

Die Verarbeitung natürlicher Sprache ist ein Forschungsbereich der Informatik und der KI, der sich mit der Verarbeitung natürlicher Sprache befasst. Bei der Verarbeitung geht es in der Regel darum, natürliche Sprache in Daten zu übersetzen, die ein Computer nutzen kann, um Informationen über die Welt um ihn herum zu erhalten.

Die NLP-Pipeline, die zur Erstellung eines Dialogsystems erforderlich ist, erfordert vier Arten der Verarbeitung sowie eine Datenbank zur Speicherung vergangener Äußerungen und Antworten. Jeder dieser Schritte kann einen oder mehrere Verarbeitungsalgorithmen enthalten, die parallel oder sequentiell arbeiten [Lane, Howard und Hapke 2019]:

- Syntaktische Zergliederung - Extraktion von Merkmalen (strukturierte numerische Daten) aus natürlichem Text.
- Analyse - Erstellen und Kombinieren von Items, um Indikatoren für den Ton, die grammatikalische Korrektheit und die Semantik des Textes zu erhalten.
- Generierung - Generierung möglicher Antworten mit Hilfe von Mustern, Suchwerkzeugen oder Sprachmodellen.
- Ausführung - bereitet Aussagen vor, die auf der Geschichte und dem Zweck des Gesprächs basieren und wählt eine Folgeantwort.

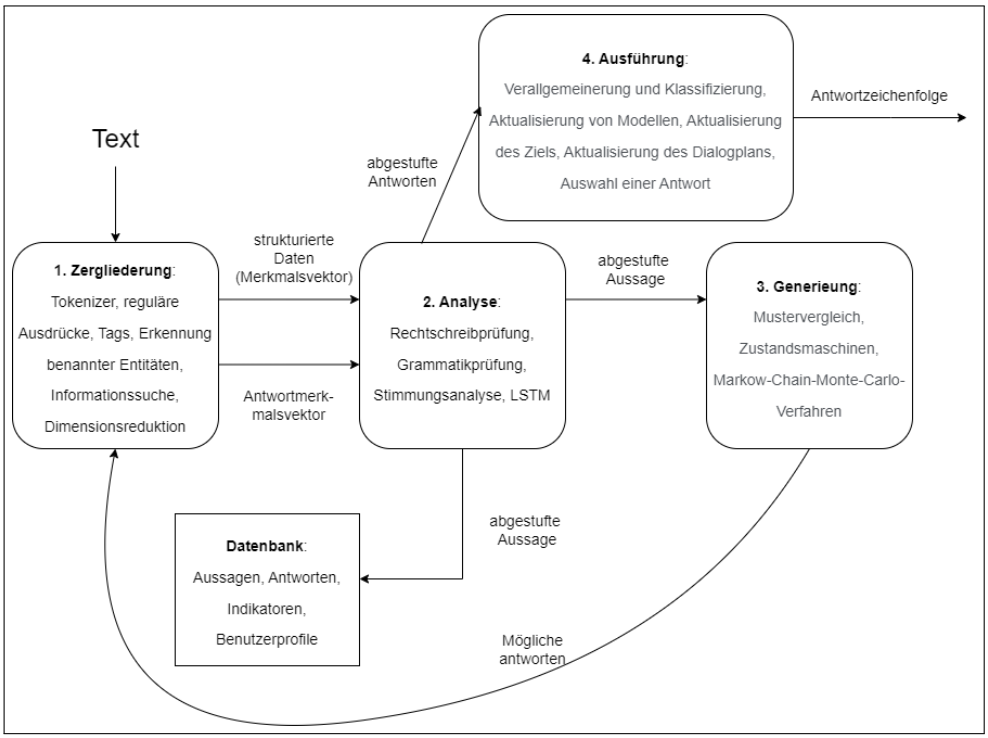

Abbildung 2.3: Rekurrente Chat-Bot-Pipeline

Die meisten Chatbots enthalten Elemente aus allen fünf Teilsystemen (die vier Verarbeitungsstufen sowie die Datenbank). Viele Anwendungen erfordern jedoch nur einfache Algorithmen, um viele dieser Schritte auszuführen. Ein Chatbot oder virtueller Assistent für Verbraucher wie Alexa oder Allo ist in der Regel so konzipiert, dass er äußerst sachkundig und leistungsfähig ist. Die Logik, die zur Beantwortung von Anfragen verwendet wird, ist jedoch oft oberflächlich und besteht aus einer Reihe von Codephrasen, die mit einer einzigen if-then-Entscheidungsverzweigung zur gleichen Antwort führen. Alexa (und die zugrundeliegende Lex-Engine) verhält sich wie ein einschichtiger, flacher Baum von Operatoren (if, elif, elif...). Andererseits stützt sich die Google Translate-Pipeline (oder jedes ähnliche maschinelle Übersetzungssystem) auf eine mehrstufige Hierarchie von Merkmalsextraktoren, Entscheidungsbäumen und Wissensgraphen, die Fragmente von Wissen über die Welt miteinander verbinden.

Alle NLP-Merkmale müssen gut funktionieren, damit das Dialogsystem richtig funktioniert:

- Merkmalsextraktion (normalerweise zur Erstellung eines Vektorraummodells).
- Informationssuche zur Beantwortung von Sachfragen.
- Semantische Suche, um Informationen aus zuvor aufgezeichneten natürlichsprachlichen Texten oder Dialogen zu assimilieren.
- Generierung von natürlicher Sprache, um neue sinnvolle Aussagen zu verfassen.

2.3.2 Tokenisierung von Wörtern

Eine der wichtigsten Aufgaben der Verarbeitung natürlicher Sprache ist die Tokenisierung, d. h. die Zerlegung von Text in einzelne Wörter oder Phrasen, die so genannten Token. Dieser Prozess ist ein wichtiger Bestandteil der Verarbeitung natürlicher Sprache, da er es Programmen und Algorithmen ermöglicht, Texte anhand ihrer Bestandteile zu verstehen und zu analysieren. Die Tokenisierung ist für die Verarbeitung natürlicher Sprache von entscheidender Bedeutung, da sie es Programmen ermöglicht, Schlüsselwörter, Phrasen und semantische Einheiten in einem Text hervorzuheben. Die Tokenisierung kann auch dazu verwendet werden, nicht benötigte Textelemente wie Satzzeichen, Leerzeichen und andere Zeichen, die keine Bedeutung haben, zu entfernen. Dadurch können Programme und Algorithmen effizienter und genauer arbeiten und die Wahrscheinlichkeit von Fehlern und falscher Textverarbeitung verringern.

Im NLP ist die Tokenisierung eine besondere Art der Segmentierung von Dokumenten. Bei der Segmentierung wird der Text in kleinere Abschnitte (Segmente) mit engerem Informationsgehalt unterteilt. Die Segmentierung kann die Aufteilung eines Dokuments in Absätze, in Sätze, in Phrasen und in Token (Wörter) sowie Satzzeichen umfassen. Ein Tokenizer kann mit einem Scanner im Kompilierungsprozess verglichen werden. Token sind in diesem Fall die Endpunkte von kontextfreien Grammatiken - Context-free grammar (CFG) - zur Analyse von Programmiersprachen-Terminalen.

Die Tokenisierung ist der erste Schritt in der NLP-Pipeline und kann daher den Rest der Pipeline stark beeinflussen. Der Tokenizer zerlegt unstrukturierte Daten, d. h. natürlichsprachliche Texte, in Informationseinheiten, die als einzelne Elemente gezählt werden können. Die so gezählte Anzahl von Token-Vorkommen in einem Dokument kann direkt als Vektor verwendet werden, der dieses Dokument repräsentiert. Ein solcher Ansatz ermöglicht es, aus einer unstrukturierten Zeichenfolge (einem Textdokument) unmittelbar eine für das maschinelle Lernen geeignete numerische Datenstruktur zu gewinnen. Diese Werte können den Computer direkt dazu veranlassen, nützliche Aktionen durchzuführen und Reaktionen zu erzeugen.

Die einfachste Art, einen Satz zu tokenisieren, ist die Verwendung von Leerzeichen als Worttrenner in Zeichenketten. Dies ist jedoch nicht optimal, denn wenn ein Satz z. B. ein Satzzeichen enthält, wird es von einem der Tokenisierer erfasst. Optimierte Tokenizer sind in mehreren Python-Bibliotheken implementiert, die jeweils ihre eigenen Vor- und Nachteile haben: spaCy, Stanford CoreNLP und Natural Language Toolkit (NLTK). NLTK und StanfordCoreNLP sind die am längsten bestehenden und am häufigsten verwendeten Bibliotheken zum Vergleich von NLP-Algorithmen in wissenschaftlichen Artikeln. Obwohl die StanfordCoreNLP-Bibliothek eine Python-Application Programming Interface (API) hat, basiert sie auf dem Java 8 CoreNLP-Anwendungsteil, der separat installiert und konfiguriert werden muss. Daher wurde in dieser

Arbeit der NLTK-Tokenizer verwendet.

Ein wichtiges Konzept im Tokenisierungsprozess sind N-Gramme. Ein N-Gramm ist eine Sequenz mit bis zu n Elementen, die aus einer Sequenz dieser Elemente, in der Regel einer Zeichenkette, extrahiert wurden. Im Allgemeinen können die Elemente eines N-Gramms Buchstaben, Silben, Wörter oder sogar Symbole sein. N-Gramme sind notwendig, weil bei der Konvertierung einer Menge von Wörtern in einen Vektor eine Folge von Token einen Großteil der Bedeutung verliert, die in der Reihenfolge dieser Wörter verkapselt ist. Wenn das Token-Konzept auf Mehrwort-Token, N-Gramme, ausgedehnt wird, kann die NLP-Pipeline einen erheblichen Teil der Bedeutung, die in der Wortfolge dieser Äußerungen enthalten ist, beibehalten. So bleibt beispielsweise das Wort „kein“, das die Bedeutung umkehrt, neben den benachbarten Wörtern stehen, wo es hingehört. Ohne N-Gramm-Tokenisierung würde ein solches Wort an verschiedenen Positionen herumhängen und seine Bedeutung würde mit dem gesamten Satz oder Dokument assoziiert werden, anstatt mit benachbarten Wörtern. Das Bigramm „war nicht“ behält viel mehr Bedeutung der einzelnen Wörter „war“ und „nicht“ als die entsprechenden Singlegramme im Multigrammvektor. Durch die Verknüpfung eines Wortes mit seinen Nachbarn in einem Förderband kann ein Teil seines Kontexts erhalten bleiben. N-Gramme sind also ein Instrument zur Speicherung von Kontextinformationen, während die Daten die Pipeline durchlaufen.

Die Größe des Vokabulars spielt eine wichtige Rolle für die Leistung der NLP-Pipeline. Die Größe des Wörterbuchs bestimmt die Größe der Trainingsstichprobe, die benötigt wird, um eine Überanpassung an ein bestimmtes Wort oder eine bestimmte Phrase zu vermeiden und die Größe der Trainingsmenge bestimmt die Kosten der Verarbeitung. Eine Technik zur Verringerung der Größe des Wörterbuchs besteht darin, Token, die ähnliche Dinge bedeuten, in einer einzigen normalisierten Form zusammenzufassen. Eine solche Technik reduziert die Anzahl der gespeicherten Token und verbessert die Verbindungen zwischen den Bedeutungen von Phrasen mit unterschiedlicher „Schreibweise“ der Tokens und verringert die Wahrscheinlichkeit des Überlernens. Eine der Normalisierungsmöglichkeiten ist die Groß- und Kleinschreibung - die Kombination mehrerer Schreibweisen eines Wortes, die sich nur in der Groß- und Kleinschreibung unterscheiden. In diesem Fall wird die Groß-/Kleinschreibung ignoriert und zwei identische Wörter, von denen eines groß und das andere klein geschrieben wird, werden als dasselbe Token behandelt.

Ein weiterer wichtiger Schritt bei der Tokenisierung von Texten ist die Entfernung von Stoppwörtern, um den Umfang des Wörterbuchs zu verringern. Stoppwörter sind in jeder Sprache gebräuchliche Wörter, die sehr häufig vorkommen, aber sehr viel weniger aussagekräftige Informationen über die Bedeutung eines Satzes enthalten (z. B. a, an, the, this, of, on im Englischen oder der, die, das, diese im Deutschen). Stoppwörter können jedoch nützliche Informationen enthalten, so dass man sie nicht immer verwerfen sollte. Das NLTK-Paket für Python enthält

derzeit die umfassendste Liste kanonischer Stoppwörter in verschiedenen Sprachen.

Eine weitere gängige Methode der Normalisierung ist die Beseitigung kleiner semantischer Unterschiede im Zusammenhang mit Pluralendungen und Possessivendungen von Wörtern oder sogar unterschiedlichen Verbformen. Diese Methode der Normalisierung, bei der ein gemeinsamer Wortstamm für verschiedene Wortformen gefunden wird, wird als Stemming bezeichnet. Der gemeinsame Wortstamm von „Gehäuse“ und „Haus“ ist zum Beispiel „Haus“. Beim Stemming werden Suffixe von Wörtern entfernt, um Wörter mit ähnlicher Bedeutung unter einem gemeinsamen Stamm zu gruppieren. Der Wortstamm muss nicht unbedingt ein gültiges Wort sein, sondern kann auch nur ein Token oder eine Bezeichnung sein, das mehrere mögliche Schreibweisen repräsentiert. Einer der Hauptvorteile des Stemming besteht darin, die Anzahl der Wörter zu verringern, deren Bedeutung das Sprachmodell im Auge behalten muss. Durch diese Methode wird die Größe des Wörterbuchs reduziert, wodurch der Verlust nützlicher Informationen und Bedeutungen so weit wie möglich begrenzt wird. Das Stemming spielt eine wichtige Rolle bei der Suche nach Schlüsselwörtern oder Informationen. Zwei der bekanntesten Algorithmen sind Porter's Stemmer und Snowball.

Mit Informationen über die Beziehungen zwischen den Bedeutungen verschiedener Wörter ist es möglich, mehrere Wörter miteinander zu verknüpfen, auch wenn ihre Schreibweise sehr unterschiedlich ist. Eine solche erweiterte Normalisierung eines Wortes auf seine semantische Wurzel - ein Lemma - wird Lemmatisierung genannt. Die Lemmatisierung ist potenziell eine viel genauere Art der Normalisierung als das Stemming oder die Groß- und Kleinschreibung, da sie die Bedeutung des Wortes berücksichtigt. Der Lemmatisierer verwendet eine Wissensbasis von Synonymen und Wortendungen, um nur eng verwandte Wörter zu einem Token zu kombinieren. In Python kann die Lemmatisierung mit dem NLTK-Paket implementiert werden, das den *WordNetLemmatizer* enthält.

Wie bereits gezeigt wurde, ist die Tokenisierung der Prozess der Zerlegung von Text in einzelne Wörter oder Token. Für viele Anwendungen der Verarbeitung natürlicher Sprache ist jedoch nicht nur wichtig, welche Wörter im Text enthalten sind, sondern man muss auch in der Lage sein, diese Wörter als Zahlen darzustellen. Dadurch wird es möglich, maschinelles Lernen und andere Algorithmen, die mit Zahlen arbeiten, zur Textverarbeitung einzusetzen. Bei der Verarbeitung natürlicher Sprache wird dazu die Wortvektorisierung verwendet.

2.3.3 Vektorisierung von Wörtern

Die Vektorisierung hilft dabei, eine numerische Darstellung zu finden, die die Bedeutung oder den Informationsgehalt der dargestellten Wörter widerspiegelt. Es gibt drei Möglichkeiten, Wörter und ihre Bedeutungen darzustellen:

- Multisets von Wörtern - Vektoren von Zahlen oder Häufigkeiten von Wörtern.
- Multisets von N-Grammen - Vektoren von Mengen von Wortpaaren (Bigramme), Worttripeln (Trigramme), usw.
- TF-IDF-Vektoren sind Indikatoren für Wörter, die deren Bedeutung am Besten darstellen.

Jede dieser Methoden kann entweder allein oder als Teil einer NLP-Pipeline angewendet werden. Alle beschriebenen Modelle sind statistisch in dem Sinne, dass sie auf Worthäufigkeiten beruhen.

Multisets von Wörtern

BOW-Vektoren sind Vektoren, die Textdokumente als Wort-Multisets (Bags) darstellen, wobei ein Multiset eine Sammlung von Elementen ist, in der jedes Element mehr als einmal wiederholt werden kann. Ein Wort-Bag ist eine Darstellung von Text, die das Vorkommen von Wörtern in einem Dokument beschreibt. Sie beinhaltet zwei Dinge:

1. Ein Wörterbuch mit bekannten Wörtern.
2. Ein Maß für das Vorhandensein von bekannten Wörtern.

Dies wird als „Word-Bag“ bezeichnet, weil alle Informationen über die Reihenfolge oder Struktur der Wörter im Dokument verworfen werden. Das Modell interessiert sich nur dafür, ob bekannte Wörter im Dokument vorkommen, nicht aber, wo im Dokument. Bei diesem Ansatz wird das Histogramm der Wörter im Text betrachtet, d. h. wir betrachten jedes Wort als ein Merkmal [Goldberg 2017].

Ein Ansatz zur Vektorisierung einer Vielzahl von Wörtern besteht darin, für jedes Wort einen Einheitsvektor zu erstellen und diese Vektoren dann zu einem einzigen Vektor zu kombinieren. Ein unitärer Vektor für ein Wort ist ein Vektor, dessen Länge der Dimension des Raumes entspricht und alle seine Komponenten sind Null, mit Ausnahme einer Komponente, die der Position des Wortes im Wörterbuch entspricht. Unitäre Vektoren sind extrem spärlich. Enthält das Wörterbuch beispielsweise 1000 Wörter, dann hat jeder unitäre Vektor eine Dimension von 1000 und die Komponente, die der Position des Wortes im Wörterbuch entspricht, ist gleich eins. Zum Beispiel sieht der unitäre Vektor für den Satz „Mojito ist ein leckerer Cocktail“ (nach dem Tokenisierungsprozess) so aus:

```
Tokens: 'Cocktail ein ist leckerer Mojito'
unitärer Vektor: [[0, 0, 0, 0, 1]
                  [0, 0, 1, 0, 0]
                  [0, 1, 0, 0, 0]
                  [0, 0, 0, 1, 0]
                  [1, 0, 0, 0, 0]]
```

Abbildung 2.4: Unitärer Vektor

Eine der Eigenschaften der vektoriellen Darstellung von Wörtern und der tabellarischen Darstellung von Dokumenten ist, dass keine ursprünglichen Informationen verloren gehen. Das Originaldokument kann aus dieser einheitlichen Vektortabelle rekonstruiert werden. Daher werden solche unitären Vektoren in neuronalen Netzen, bei der sequence-to-sequence (seq2seq)-Modellierung und bei der Erstellung von Sprachmodellen verwendet. Unitäre Vektoren eignen sich hervorragend für alle NLP-Modelle oder -Pipelines, bei denen die Bedeutung des Ausgangstextes vollständig erhalten bleiben muss. Die Erstellung von unitären Vektoren für jedes Wort kann mit verschiedenen Bibliotheken in Python erfolgen. Zum Beispiel kann die NumPy-Bibliothek verwendet werden, um unitäre Vektoren zu erstellen.

Diese Vektordarstellung hat jedoch ihre Nachteile. Bei Dokumenten, die eine große Anzahl von Wörtern enthalten, wäre die Größe des unitären Vektorwörterbuchs enorm und würde enorme Ressourcen zur Verarbeitung erfordern. Außerdem macht es keinen Sinn, so viele Nullen zu speichern. Um mit dem Wörterbuch arbeiten zu können, ohne viel Zeit für die Verarbeitung zu verschwenden, ist es notwendig, das Wörterbuch zu verkleinern. Dazu ist es notwendig, auf die Möglichkeit der Rekonstuierung des Originaldokuments zu verzichten, d.h. sich die Wortfolge aller Dokumente zu merken. Es ist notwendig, den größten Teil der Informationen zu erfassen, nicht das gesamte Dokument. Dafür sind Multiset-Vektoren hilfreich.

Um einen Vektor für ein Multiset von Wörtern zu erstellen, müssen für jedes Wort der Mehrmenge ein Einheitsvektor erstellt und diese Vektoren dann zu einem einzigen Vektor kombiniert werden. Vektoren können nach verschiedenen Regeln kombiniert werden, z. B. durch Addition, Multiplikation, Verkettung oder andere Operationen. Wenn alle diese unitären Vektoren zusammengezählt werden, wird ein Vektor der Worthäufigkeit erhalten. Er spiegelt die Häufigkeit des Vorkommens von Wörtern wider, nicht die Reihenfolge, in der sie auftreten. Dieser Vektor kann verwendet werden, um ein ganzes Dokument oder einen Satz als einen einzigen Vektor von nicht sehr großer Länge darzustellen. Seine Länge entspricht der Länge des Wörterbuchs (der Anzahl der verfolgten eindeutigen Token). Außerdem kann bei einer einfachen Schlüsselwortsuche ein binärer Multiset-Vektor durch eine logische OR-Operation aus unitären Vektoren gewonnen werden.

Die Anzahl der Vorkommen eines Wortes in einem bestimmten Dokument wird als Termfrequenz (TF) bezeichnet. Je häufiger ein Wort in einem Dokument vorkommt, desto wahrscheinlicher ist es, dass das Dokument von diesem Thema handelt. Anstelle von reinen Wortzählungen können auch normalisierte Termhäufigkeiten zur Beschreibung von Dokumenten aus einem Korpus verwendet werden. Unter normalisierten Dokumenthäufigkeiten versteht man die Anzahl der Wörter im Verhältnis zur Länge des Dokuments. Die Normalisierung ist sehr nützlich, um die Bedeutung eines Dokuments zu ermitteln, da reine Häufigkeiten nicht immer die tatsächliche Situation widerspiegeln. Wenn beispielsweise das Wort „Cocktail“ 15 Mal in Dokument №1 und 400 Mal in Dokument №2 vorkommt, kann man daraus schließen, dass es in Dokument Nummer zwei eher um Cocktails geht. Vielleicht handelt es sich aber bei Dokument №1 um ein Rezept für die Zubereitung von Mai Tais mit 100 Wörtern, während Dokument №2 ein Roman mit 500.000 Wörtern ist, in dem die Figuren gelegentlich Cocktails trinken. Die Häufigkeit des Wortes „Cocktail“ in Dokument №1 beträgt also 0,15 und 0,0008 in Dokument №2, was die tatsächliche Bedeutung des Begriffs „Cocktail“ in diesen Dokumenten viel besser widerspiegelt:

$$TF(Cocktail, Dokument_1) = 15/100 = 0{,}15$$
$$TF(Cocktail, Dokument_2) = 400/500000 = 0{,}0008$$

Abbildung 2.5: Normalisierte TF-Vektoren

Ebenso ist es möglich, diesen Wert für jedes Wort zu berechnen und die relative Bedeutung dieses Begriffs für jedes Dokument zu ermitteln, woraus sich die Bedeutung jedes Dokuments ableiten lässt. Um die Anzahl der Vorkommen von Wörtern zu ermitteln, kann das Objekt *collections.Counter* in Python verwendet und anschließend eine Normalisierung durchgeführt werden:

```
sentence = "Die mit der Getränkmischmaschine zubereiteten Cocktails
können sich von denen an der Bar unterscheiden, da die Cocktails an
der Bar stark mit Wasser verdünnt sind"
tokenizer = TreebankWordTokenizer()
tokens = tokenizer.tokenize(sentence.lower())
token_counts = Counter(tokens)
document_vector = []
doc_length = len(tokens_intro)
for key, value in token_counts.most_common():
    document_vector.append(value/doc_length)
>>> document_vector:

[0.11538461538461539,
 0.07692307692307693,
 0.07692307692307693,
 0.07692307692307693,
 0.07692307692307693,
 0.07692307692307693,
 0.038461538461538464,
 …,
 0.038461538461538464]
```

Abbildung 2.6: Normalisierte TF-Vektoren mit Python

Wenn mehr als ein Dokument bearbeitet wird, wird für jedes Dokument ein eigener Häufigkeitsvektor erstellt, der jedoch absolut alle Begriffe aus allen Dokumenten (alle Wörter aus dem Wörterbuch) enthält, auch wenn einige der Wörter nicht in allen Dokumenten vorkommen. In diesem Fall werden die Vektoren an diesen Stellen Nullwerte enthalten.

Das ultimative Ziel der Textvektorisierung besteht darin, natürliche Sprache in ein Format zu übersetzen, das der Computer verstehen kann. Eine Möglichkeit, dies zu erreichen, ist die Verwendung von Vektorräumen. Vektorräume sind mathematische Objekte, in denen jedes Element durch einen Vektor dargestellt wird und verschiedene Operationen mit Vektoren ermöglichen es uns, nützliche Merkmale zu extrahieren. So wird ein Vektor mit zwei Werten in einem zweidimensionalen Vektorraum verortet, ein Vektor mit drei Werten in einem dreidimensionalen Vektorraum und so weiter. Beispielsweise ist der Vektorraum von Dokumenten, die nur aus den Wörtern „Cocktail“ und „Maschine“ bestehen, zweidimensional und die Vektoren der enthaltenen Dokumente können wie folgt dargestellt werden:

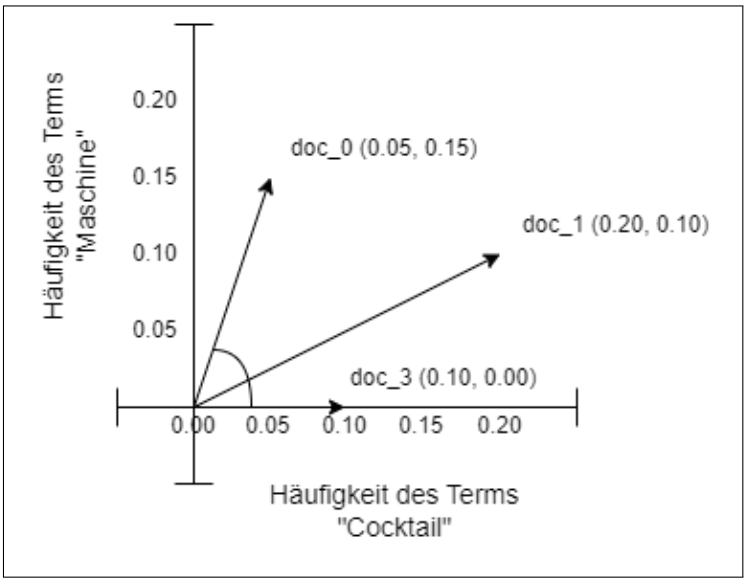

Abbildung 2.7: Zweidimensionale Vektoren von Termhäufigkeiten

Für einen Vektorraum von Dokumenten in einer natürlichen Sprache ist die Dimensionalität des Vektorraums gleich der Anzahl der verschiedenen Wörter im gesamten Korpus. Jedes Dokument in einem K-dimensionalen Vektorraum kann durch einen K-dimensionalen Vektor beschrieben werden. Wenn es Vektorrepräsentationen für alle Dokumente im gleichen Raum gibt, dann ist es möglich, sie zu vergleichen. Der euklidische Abstand zwischen Vektoren kann gemessen werden, indem sie subtrahiert werden und die Länge des Abstands zwischen ihnen berechnet wird, den sogenannten Abstand im Sinne der L2-Norm (L2-Metrik). Dies ist der Abstand auf einer geraden Linie von dem Ort, der durch die Spitze (Scheitelpunkt) des einen Vektors gegeben ist, zu dem Ort, der der Spitze des anderen Vektors entspricht.

$$d(p,q) = ||q-p||_2 = \sqrt{(q_1-p_1)^2 + ... + (q_n-p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2} \tag{2.1}$$

Der Ochiai-Koeffizient (Kosinuskoeffizient) ist ein Maß für die Ähnlichkeit zwischen zwei Multisets. Der Ochiai-Koeffizient kann verwendet werden, um den Grad der Ähnlichkeit zwischen zwei Dokumenten auf der Grundlage ihrer Wort-Multisets zu bestimmen. Er ist einfach der Kosinus des Winkels zwischen zwei Vektoren (Theta).

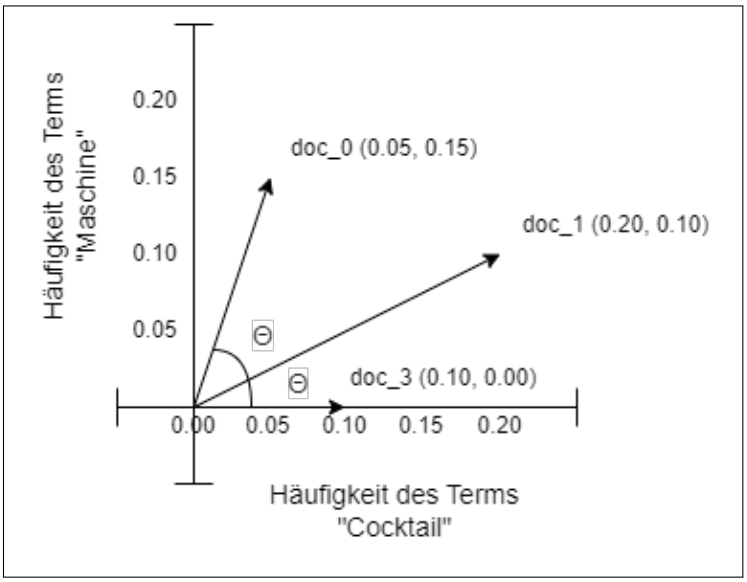

Abbildung 2.8: Zweidimensionale Theta

Er kann mithilfe des euklidischen Skalarprodukts nach folgender Formel berechnet werden:

$$A \cdot B = ||A||\,||B||\,cos\,\theta \tag{2.2}$$

$$cosine\ similarity = S_c(A,B) := cos(\theta) = \frac{A \cdot B}{||A||\,||B||} = \frac{\sum_{i=1}^{n}(A_i\,B_i)}{\sqrt{\sum_{i=1}^{n} A_i^2}\,\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2.3}$$

Das Skalarprodukt zweier Vektoren kann also berechnet werden, indem die Elemente der Vektoren paarweise multipliziert und dann die Ergebnisse addiert werden. Anschließend wird das Ergebnis durch die Norm jedes Vektors dividiert. Die Norm eines Vektors ist gleich dem euklidischen Abstand zwischen seinem Kopf und seinem Ende - der Quadratwurzel aus der Summe der Quadrate seiner Elemente. Dieses normierte Skalarprodukt liegt wie der Kosinuswert zwischen -1 und 1. Es ist auch der Kosinus des Winkels zwischen den beiden Vektoren. Dieser Wert ist gleich dem Anteil des längeren Vektors, der von der senkrecht dazu stehenden Projektion des kürzeren Vektors abgedeckt wird. Daraus ist ersichtlich, wie weit unsere beiden Vektoren in dieselbe Richtung zeigen.

Ein offensichtlicher Vorteil ist, dass der vom Ochiai-Koeffizienten angenommene Wertebereich (von -1 bis +1) für die meisten Aufgaben des maschinellen Lernens geeignet ist. Ein Kosinuskoeffizient von 1 spiegelt identische normalisierte Vektoren wider, die über alle Dimensionen hinweg in eine ähnliche Richtung zeigen. Die Längen dieser Vektoren können unterschiedlich sein, aber sie zeigen in dieselbe Richtung. Je näher der Kosinuskoeffizient bei 1 liegt, desto kleiner ist der Winkel zwischen den Vektoren. Im NLP enthalten Dokumente, deren Kosinuskoeffizient der Vektoren nahe bei 1 liegt, ähnliche Wörter im gleichen Verhältnis. Daher ist es wahrscheinlicher,

dass es sich bei Dokumenten, deren Vektoren nahe beieinander liegen, um dieselbe Sache handelt. Wenn der Kosinuskoeffizient 0 ist, haben die Vektoren keine gemeinsamen Komponenten. Sie stehen in allen Dimensionen in einem 90°-Winkel zueinander. Bei TF-NLP-Vektoren tritt diese Situation nur ein, wenn die beiden Dokumente keine gemeinsamen Wörter haben. Da sie völlig unterschiedliche Wörter verwenden, sprechen sie von völlig unterschiedlichen Dingen. Das bedeutet nicht unbedingt, dass die Bedeutung oder das Thema unterschiedlich ist, sondern nur, dass sie unterschiedliche Wörter verwenden. Bei einem Wert des Kosinuskoeffizient von -1 sind die Vektoren völlig entgegengesetzt. Sie zeigen in entgegengesetzte Richtungen. Bei einfachen Vektoren der Termhäufigkeiten ist dies jedoch nicht möglich, da Vektoren der Termhäufigkeiten nicht negativ sein können. Sie befinden sich also immer im gleichen Quadranten des Vektorraums.

Neben den Multisets von Wörtern gibt es weitere Vektorisierungsmethoden, die ebenfalls in der Verarbeitung natürlicher Sprache verwendet werden - Multisets von N-Grammen und TF-IDF-Vektoren. Bei Multisets von N-Grammen handelt es sich um Mengen von mehreren aufeinanderfolgenden Wörtern in einem Text. N-Gramme können helfen, den Kontext und die Beziehung zwischen Wörtern in einem Text zu erfassen. In diesem Fall wird nicht die Häufigkeit der einzelnen Begriffe im Dokument berücksichtigt, sondern die Häufigkeit des Auftretens von N-Grammen. TF-IDF-Vektoren sind jedoch eine noch effizientere Methode zur Vektorisierung von Text. Im Gegensatz zu Multisets von Wörtern und N-Grammen berücksichtigen TF-IDF-Vektoren nicht nur die Häufigkeit von Wörtern in einem Dokument, sondern auch die Bedeutung jedes Wortes im Kontext des gesamten Textkorpus. So haben Wörter, die in allen Dokumenten häufig vorkommen, eine geringere Bedeutung als seltene Wörter, die dazu beitragen, ein Dokument von anderen zu unterscheiden.

TF-IDF-Vektoren

Das TF-IDF-Vektorkonzept ist eines der grundlegenden Werkzeuge der natürlichen Sprachverarbeitung. Es wird verwendet, um Textdokumente in numerische Vektoren umzuwandeln, die für Textanalyse, Dokumentenklassifikation, Empfehlungssysteme und viele andere Anwendungen verwendet werden können.

Eine der wichtigsten Fragen, die sich bei der Verarbeitung natürlicher Sprache stellen, ist die Bestimmung der Bedeutung von Wörtern in einem Text. Die Bedeutung von Wörtern kann anhand ihrer Häufigkeit in einem Dokument bestimmt werden. Einige Wörter kommen jedoch häufig in allen Dokumenten vor, so dass sie nicht allzu wichtig sein sollten. Andererseits können einige seltene Wörter von großer Bedeutung sein, weil sie möglicherweise nur in einem bestimmten Dokument vorkommen. TF-IDF-Vektoren lösen dieses Problem, indem sie nicht nur die Häufigkeit der Wörter in einem Dokument, sondern auch ihre Bedeutung im Kontext des gesamten Textkorpus berücksichtigen. Für jedes Wort in einem Dokument kann sein TF-IDF-Wert nach

der folgenden Formel berechnet werden:

$$TF(t,d) = \frac{Anzahl(t)}{Anzahl(d)} \tag{2.4}$$

$$IDF(t,D) = \log \frac{Anzahl(D)}{Anzahl(D,\ die\ t\ beinhalten)} \tag{2.5}$$

$$TFIDF(t,d,D) = TF(t,d) \cdot IDF(t,D) \tag{2.6}$$

Dabei ist t ein Wort, d ein Dokument, TF(t, d) der Wert der Worthäufigkeit im Dokument und IDF(t, D) der Wert der inversen Dokumenthäufigkeit. Um einen TF-IDF-Vektor für jedes Dokument zu erhalten, müssen die TF-IDF-Werte für alle Wörter, die in dem Dokument vorkommen, verwendet werden. Dies ergibt einen Vektor, bei dem jede Komponente einem TF-IDF-Wert für ein bestimmtes Wort im Dokument entspricht. Beispielsweise werden die folgenden drei Dokumente betrachtet:
Dokument №1 „Ein Cocktail ist variationsreich und auch zuhause gut zu mixen“
Dokument №2 „Der Mojito ist ein klassischer Cocktail und ein erfrischender Drink der immer passt“
Dokument №3 „Dank unserem Rezept machen Sie diesen Cocktail selber zuhause“
Zunächst müssen die Dokumente tokenisiert und unnötige Stoppwörter herausgefiltert werden, die für das Verständnis der Bedeutung der Dokumente keine große Rolle spielen. Nach diesem Prozess sehen die Dokumente wie folgt aus:
Dokument №1 „Cocktail variationsreich zuhause gut mixen“
Dokument №2 „Mojito klassischer Cocktail erfrischender Drink immer passt“
Dokument №3 „Dank Rezepte machen Cocktail selber zuhause“
Als erstes müssen die normalisierten TF-Vektoren jeden Terms berechnet werden:

TF(t, d)	Dokument №1	Dokument №2	Dokument №3
Cocktail	1/5 = 0.2	1/7 = 0.143	1/6 = 0.167
variationsreich	0.2	0	0
zuhause	0.2	0	0.167
gut	0.2	0	0
mixen	0.2	0	0
Mojito	0	0.143	0
klassischer	0	0.143	0
erfrischender	0	0.143	0
Drink	0	0.143	0
immer	0	0.143	0
passt	0	0.143	0
Dank	0	0	0.167
Rezepte	0	0	0.167
machen	0	0	0.167
selber	0	0	0.167

Tabelle 2.1: TF-Vektoren

Anschließend können die IDF-Vektoren für jeden Term (über den gesamten Dokumentenkorpus) berechnet werden:

IDF(t, D)	Dokumentenkorpus
Cocktail	log(3/3) = 0
variationsreich	0.477
zuhause	0.176
gut	0.477
mixen	0.477
Mojito	0.477
klassischer	0.477
erfrischender	0.477
Drink	0.477
immer	0.477
passt	0.477
Dank	0.477
Rezepte	0.477
machen	0.477
selber	0.477

Tabelle 2.2: IDF-Vektoren

Jetzt kann mit der Berechnung der TF-IDF-Vektoren fortgefahren werden:

TF-IDF(t, d, D)	Dokument №1	Dokument №2	Dokument №3
Cocktail	0.2*0 = 0	0.143*0 = 0	0.167*0 = 0
variationsreich	0.0954	0	0
zuhause	0.0352	0	0.03
gut	0.0954	0	0
mixen	0.0954	0	0
Mojito	0	0.068	0
klassischer	0	0.068	0
erfrischender	0	0.068	0
Drink	0	0.068	0
immer	0	0.068	0
passt	0	0.068	0
Dank	0	0	0.08
Rezepte	0	0	0.08
machen	0	0	0.08
selber	0	0	0.08

Tabelle 2.3: TF-IDF-Vektoren

Wie aus dem Beispiel ersichtlich, haben Wörter, die in allen Dokumenten vorkommen, wie z.B. „Cocktail“, in jedem Dokument den gleichen TF-IDF-Wert, Wörter, die in mehr als einem Dokument vorkommen, wie z.B. „zuhause“, haben einen niedrigeren IDF-Wert als seltene Wörter, wie z.B. „selber“ oder „Rezept“, die in den Dokumenten, in denen sie vorkommen, einen hohen IDF-Wert und damit einen hohen TF-IDF-Wert haben. Diese durch die Berechnung von TF-IDF erhaltene Zahl assoziiert also ein bestimmtes Wort oder Token mit einem bestimmten Dokument in einem bestimmten Korpus und legt dann einen numerischen Wert für die Bedeutung dieses Wortes in diesem Dokument fest, wobei sein Vorkommen im gesamten Korpus berücksichtigt wird. Da für jedes Wort in den Dokumenten nun TF-IDF-Werte vorliegen, können diese zu TF-IDF-Vektoren für jedes Dokument kombiniert werden. Der TF-IDF-Vektor für jedes Dokument hat eine Dimension, die der Gesamtzahl der eindeutigen Wörter im Textkorpus entspricht. Jedes Element im Vektor entspricht einem bestimmten Wort und der Wert des Elements ist gleich seinem TF-IDF-Wert für dieses Dokument. In diesem Beispiel sehen die TF-IDF-Vektoren für jedes Dokument wie folgt aus:
Dokument №1 [0. 0.0954 0.0352 0.0954 0.0954 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Dokument №2 [0. 0. 0. 0. 0. 0.068 0.068 0.068 0.068 0.068 0.068 0. 0. 0. 0.]
Dokument №3 [0. 0. 0.03 0. 0. 0. 0. 0. 0. 0. 0. 0.08 0.08 0.08 0.08]

Aus diesen Vektoren kann die Ähnlichkeit zwischen den Dokumenten berechnet werden, z. B. mit Hilfe des Kosinuskoeffizients. Je näher die Werte des Kosinuskoeffizients bei 1 liegen, desto ähnlicher sind sich die Dokumente. Für die maximale derzeit erreichbare Effizienz bei der semantischen Suche, der Dokumentenklassifikation, bei Dialogsystemen und den meisten anderen Anwendungen ist es daher ausreichend, nur einfache TF-IDF-Vektoren in die Pipeline einzugeben. TF-IDF-Vektoren sind die erste Stufe der Pipeline, die Grundmenge der aus dem Text extrahierten Merkmale. Auf der Grundlage der TF-IDF-Vektoren können Themenvektoren berechnet werden, die für die semantische Analyse von Texten erforderlich sind.

Es gibt viele Bibliotheken, um die TF-IDF-Vektorisierung in Python zu implementieren. Einige der beliebtesten Bibliotheken sind Scikit-Learn, Gensim, NLTK und SpaCy.

2.3.4 Syntaktische Analyse

Mit Hilfe von TF-IDF-Vektoren ist es möglich, die Bedeutung von Wörtern in einem Textfragment zu schätzen. TF-IDF-Vektoren und -Matrizen können verwendet werden, um zu verstehen, wie wichtig jedes Wort für die Gesamtbedeutung eines Textfragments in einer Reihe von Dokumenten ist. Solche TF-IDF-Bedeutungsschätzungen sind nicht nur für Wörter, sondern auch für kurze Wortfolgen, N-Gramme, möglich. Und letztere sind sehr praktisch für die Suche im Text (wenn die gesuchten Wörter oder N-Gramme genau bekannt sind).

Experimente auf dem Gebiet des NLP haben zu einem Algorithmus geführt, mit dem die Bedeutung von Wortkombinationen ermittelt und Vektoren zur Darstellung dieser Bedeutung berechnet werden können. Dieser Algorithmus wird latente semantische Analyse (Latent Semantic Analysis (LSA)) genannt. Mit ihr lassen sich nicht nur Wörter, sondern ganze Dokumente als Vektoren darstellen [Rygl u. a. 2017]. Diese Vektoren werden semantische Vektoren oder Themenvektoren genannt. Diese Themenvektoren können verwendet werden, um nach Dokumenten anhand ihrer semantischen Bedeutung zu suchen - eine semantische Suche. In den meisten Fällen sind die Ergebnisse dieser Suche wesentlich besser als die einer Schlagwortsuche (TF-IDF-Suche). Manchmal liefert eine semantische Suche genau die Dokumente, nach denen der Benutzer gesucht hat, auch wenn er nicht die richtigen Wörter für seine Anfrage gefunden hat. Darüber hinaus können diese semantischen Vektoren verwendet werden, um die Wörter und N-Gramme zu finden, die das Thema einer Aussage, eines Dokuments oder eines Korpus (Menge von Dokumenten) am besten repräsentieren. Mit Hilfe der Wortvektoren und ihrer relativen Bedeutung ist es möglich, die repräsentativsten Wörter für ein Dokument zu finden, wie z. B. eine Reihe von Schlüsselwörtern, die seine Bedeutung widerspiegeln. Es ist nun auch möglich, zwei beliebige Aussagen/Dokumente zu vergleichen und festzustellen, wie nahe sie sich in ihrer Bedeutung sind.

Themenvektoren sind eine Möglichkeit, ein Dokument als Vektor darzustellen, wobei jedes

Element einem Thema entspricht. Jedes Thema ist eine Wahrscheinlichkeitsverteilung auf die Wörter im Korpus und für jedes Dokument kann geschätzt werden, wie sehr es mit jedem Thema zu tun hat. Zur Berechnung der Themenvektoren wird der LSA-Algorithmus verwendet, der auf einer SWZ(eng. Singular Value Decomposition (SVD)) der Term-Dokumenten-Matrix beruht.

Bei der SWZ wird die Matrix in drei Komponenten zerlegt: die Matrix der linken singulären Vektoren, die Matrix der rechten singulären Vektoren und die Diagonalmatrix der singulären Werte. Diese drei einfacheren Matrizen aus der SWZ offenbaren jedoch Eigenschaften der ursprünglichen TF-IDF-Matrix. Sie eignen sich zur Vereinfachung der Matrix. Man kann die Matrizen abschneiden (einige Zeilen und Spalten weglassen), bevor man sie erneut multipliziert, wodurch die Anzahl der Dimensionen, mit denen man sich im Vektorraummodell befassen muss, reduziert wird. Das Ergebnis der Multiplikation dieser abgeschnittenen Matrizen ist nicht dieselbe TF-IDF-Matrix, sondern eine verbesserte: die neue Dokumentendarstellung enthält das Wesentliche, die latente Semantik dieser Dokumente. Aus diesem Grund wird SWZ auch in anderen Anwendungen wie der Komprimierung eingesetzt (z. B. zur Komprimierung von JPEG-Bildern).

Die latente semantische Analyse (LSA) ist eine mathematische Methode, um den besten Weg zur linearen Transformation (Rotation und Streckung) eines beliebigen Satzes von NLP-Vektoren, wie TF-IDF-Vektoren oder Multiset-Vektoren, zu finden. Für viele Anwendungen ist es am besten, die Achsen (Dimensionen) in den neuen Vektoren an der größten Streuung (Varianz) der Worthäufigkeiten auszurichten [Jurafsky u. a. 2009]. Es können dann diejenigen Dimensionen aus dem neuen Vektorraum ausgeschlossen werden, die nur geringe Auswirkungen auf die Vektoränderungen von Dokument zu Dokument haben. Diese Anwendung der SWZ wird als trunkierte SWZ (trunkated SVD) bezeichnet.

LSA verwendet SWZ, um nach Wortkombinationen zu suchen, die gemeinsam für die größte Varianz in den Datenwerten verantwortlich sind. Die TF-IDF-Vektoren können so gedreht werden, dass ihre neuen Dimensionen (Basisvektoren) mit diesen Richtungen der maximalen Varianz übereinstimmen. Die Basisvektoren stellen die Koordinatenachsen des neuen Vektorraums dar. Jede der Dimensionen (Koordinatenachsen) wird zu einer Kombination von Worthäufigkeiten und nicht zu einer einzelnen Worthäufigkeit. Sie können also als gewichtete Kombinationen der Wörter betrachtet werden, die die verschiedenen im Korpus verwendeten Themen bilden. Die LSA-Methode liefert, wie der IDF-Teil von TF-IDF, Informationen darüber, welche Vektordimensionen für die Semantik (Bedeutung) von Dokumenten eine wichtige Rolle spielen. Die Dimensionen (Topics) mit der geringsten Variation in den Vektoren von Dokument zu Dokument können verworfen werden. Solche Themen mit geringer Varianz stellen in der Regel Rauschen dar und sind eine Ablenkung für jeden Algorithmus des maschinellen Lernens. Wenn ein Thema

in allen Dokumenten etwa gleich häufig vorkommt und nicht zur Unterscheidung zwischen den Dokumenten beiträgt, kann es verworfen werden, so dass die Vektordarstellung verallgemeinert werden kann, so dass sie sich besser für Dokumente eignet, auf die die Pipeline noch nicht gestoßen ist, selbst wenn sie aus einem anderen Kontext stammen.

SWZ ist ein Algorithmus zur Zerlegung einer beliebigen Matrix in drei Faktoren, d. h. drei Matrizen, die multipliziert werden, um die ursprüngliche Matrix zu bilden. Dieser Algorithmus wird in der linearen Algebra verwendet, um eine Matrix umzukehren und die drei mit SWZ berechneten Matrixmultiplikatoren haben mehrere nützliche mathematische Eigenschaften, die für die Dimensionalitätsreduktion und die LSA-Methode geeignet sind [Eckart und Young 1936].

Jedes Mal, wenn SWZ eine Term-Dokumenten-Matrix BOW oder eine Term-Dokumenten-Matrix TF-IDF durchläuft, findet SWZ übereinstimmende Wortkombinationen. Um diese übereinstimmenden Wörter zu finden, berechnet SWZ die Korrelation zwischen den Spalten (Termen) der Term-Dokumenten-Matrix. SWZ ermittelt gleichzeitig die Korrelation der Verwendung von Begriffen in Dokumenten und die Korrelation von Dokumenten untereinander. Auf der Grundlage dieser Informationen berechnet SWZ auch lineare Kombinationen von Begriffen mit maximaler Varianz innerhalb eines Korpus. Solche Kombinationen von Termhäufigkeiten werden zu Themen. Und nur die Themen, die die maximale Information und Varianz innerhalb eines Korpus beherbergen, werden als relevant markiert und beibehalten, die anderen werden abgeschnitten. Darüber hinaus besteht die Möglichkeit der linearen Transformation (Drehung) der Term-Dokument-Vektoren, um sie in kürzere Themenvektoren für jedes der Dokumente umzuwandeln. [Wong, Ziarko und Wong 1985]

SWZ gruppiert die Begriffe mit hoher Korrelation (weil sie häufig in denselben Dokumenten vorkommen) und auch gleichzeitig in einer Reihe von Dokumenten variieren [Isbell und Viola 1998]. Diese linearen Wortkombinationen werden als Themen betrachtet, die BOW-Vektoren (oder TF-IDF-Vektoren) in Themenvektoren verwandeln, die angeben, auf welche Themen sich das Dokument bezieht. Ein Themenvektor ist eine Zusammenfassung (Synthese) des Inhalts des Dokuments. Eine SWZ sieht wie folgt aus [Deerwester u. a. 1990]:

$$A_{m\times n} = U_{m\times p} S_{p\times p} V^T_{p\times n} \tag{2.7}$$

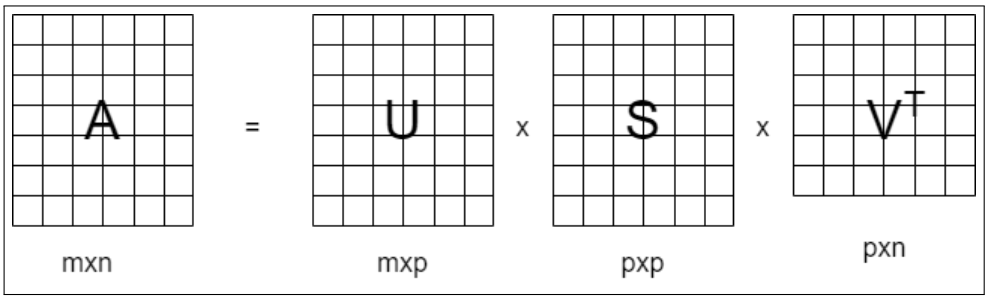

Abbildung 2.9: SWZ-Matrize

Wobei m die Anzahl der Begriffe im Wörterbuch, n die Anzahl der Dokumente im Korpus und p die Summe der Themen im Korpus ist, die der Anzahl der Wörter entspricht.

Nach dem Theorem von Eckart-Young reduziert die SWZ das Rauschen und die Spärlichkeit der ursprünglichen Matrix, indem sie durch eine Matrix derselben Dimensionalität, aber niedrigeren Rangs ersetzt wird, in der nur die wichtigsten Informationen erhalten bleiben. [Eckart und Young 1936]

Die U-Matrix enthält die Term-Themen-Matrix. Sie wird auch als linke singuläre Vektormatrix bezeichnet, da sie die Zeilenvektoren enthält, die von links mit der Spaltenvektormatrix zu multiplizieren sind. Die U-Matrix spiegelt die gegenseitige Korrelation von Wörtern und Themen auf der Grundlage des gemeinsamen Auftretens von Wörtern im selben Dokument wider. Vor der Trunkierung (Entfernung der Spalten) ist sie quadratisch. Die Anzahl der Spalten und Zeilen in ihr entspricht der Anzahl der Wörter im Wörterbuch (m). Vor der Trunkierung ist die Anzahl der Themen (p) gleich der Anzahl der Wörter. Die U-Matrix enthält als Spalten Vektoren der Themen für alle Wörter des Korpus. Das bedeutet, dass sie verwendet werden kann, um einen Wort-Dokument-Vektor (TF-IDF-Vektor oder BOW-Vektor) in einen Themen-Dokument-Vektor umzuwandeln. Um einen neuen „topic-document“-Vektor zu erhalten, multipliziert man einfach die U-Matrix „topic-word“ mit einem beliebigen „word-document“-Vektor. Der Punkt ist, dass die Gewichte (Scores) in den Zellen der U-Matrix die Bedeutung der Wörter für die Themen widerspiegeln.

Die quadratische diagonale S-Matrix (Sigma-Matrix) enthält die Singulärwerte (Eigenwerte) des Themas. Diese singulären Werte geben an, wie viel Information in jeder Dimension des neuen semantischen (Themen-)Vektorraums enthalten ist. In der diagonalen Matrix befinden sich Werte, die nicht Null sind, nur auf der Diagonalen von der linken oberen Ecke bis zur rechten unteren Ecke. Alle anderen Positionen in der S-Matrix enthalten Nullen. Und die Dimensionen (Themen) sind so angeordnet, dass die erste Dimension die meisten Informationen (erklärbare Varianz) über den Fall enthält. Wenn es also notwendig ist, das Themenmodell abzuschnei-

den, kann man mit den Messungen von unten rechts beginnen und sich nach links bewegen. Und dann aufhören, diese singulären Werte auf Null zu setzen, wenn der Fehler des thematischen Modells eine signifikante Auswirkung auf den Gesamtfehler der NLP-Pipeline haben wird.

Die V^T-Matrix „Dokument-Dokument“ enthält als Spalten die rechten singulären Vektoren und gibt Aufschluss über die gemeinsamen Bedeutungen von Dokumenten, da sie die Häufigkeit der Verwendung der gleichen Themen in Dokumenten in dem neuen semantischen Modell von Dokumenten misst. Es enthält so viele Zeilen (p) und Spalten, wie es Dokumente im Korpus gibt.

Nachdem die singuläre Zerlegung abgeschlossen ist, müssen die Themen so gekürzt werden, dass sie die Bedeutung der Dokumente wiedergeben. Dabei müssen die Elemente der Matrix S in absteigender Reihenfolge sortiert werden. In diesem Fall kann die Formel in einer anderen Form geschrieben werden:

$$A_k = U_k S_k V_k^T \tag{2.8}$$

Dabei sind U_k und V_k Matrizen, die durch Extraktion der ersten k Spalten aus den Matrizen U bzw. V gewonnen werden.

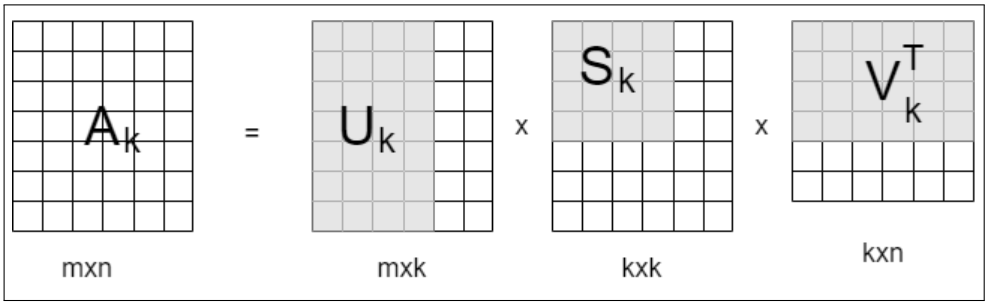

Abbildung 2.10: Trunkierte SWZ-Matrize

Im latent-semantischen Modell kann also jeder Text und jeder Begriff durch Vektoren in einem reduzierten Raum der Dimension k - dem Themenraum - dargestellt werden. Diese SWZ wird als parsimonious bezeichnet, weil sie in dem Fall, in dem k viel kleiner als m, n und p ist, eine erhebliche Kompression der ursprünglichen Information ermöglicht. Komprimierung ist in dem Sinne zu verstehen, dass ein Teil der von der ursprünglichen Matrix übermittelten Informationen verloren geht und nur die wichtigsten (dominanten) Informationen erhalten bleiben. Der Informationsverlust entsteht durch die Vernachlässigung kleiner Singulärzahlen. Je mehr Singulärzahlen verworfen werden, d. h. je kleiner k, desto größer ist dieser Verlust. Mit anderen Worten: Der Informationsverlust ist umso größer, je niedriger der Rang der approximierenden Matrix ist. In diesem Fall gehen zunächst die nicht signifikanten Werte verloren, wodurch die dominanten Werte der Matrix erhöht werden.

Der SWZ-Algorithmus, das Herzstück von LSA, erkennt, ob Wörter immer nebeneinander vorkommen und fasst sie im Betreff zusammen. Auf diese Weise kann er die Anzahl der Messungen ohne zusätzlichen Aufwand reduzieren. LSA (SWZ) ist eine großartige Methode, um Wort-Dokument-Matrizen zu komprimieren und potenzielle zusammengesetzte Wörter oder N-Gramme für die Pipeline zu erkennen. LSA reduziert die Anzahl der Messungen und hilft so, Übertraining zu vermeiden. Sie führt eine Verallgemeinerung auf der Grundlage eines kleinen Datensatzes durch, indem sie eine lineare Beziehung zwischen den Worthäufigkeiten annimmt.

LSAs können auch für die semantische Suche verwendet werden. Eine semantische Suche (semantic search) ist eine Art der Volltextsuche, die die Bedeutung der Wörter in der Anfrage und der gesuchten Dokumente berücksichtigt. Durch die Verwendung von Themenvektoren ist es möglich, die Bedeutung von Wörtern, Dokumenten, Aussagen und Korpora zu vergleichen. Es ist möglich, Cluster von ähnlichen Dokumenten und Aussagen zu finden. Mit Themenvektoren wird der Abstand zwischen Dokumenten nicht nur auf der Grundlage der in ihnen verwendeten Wörter betrachtet, sondern auf der Grundlage von Wörtern, die der Bedeutung entsprechen. Semantische Suchen mit Themenvektoren beschränken sich nicht nur auf die Suche nach Schlüsselwörtern und Relevanzrankings, die allein auf der Auswahl von Wörtern oder Vokabeln basieren. Sie können verwendet werden, um Dokumente zu finden, die für die Anfrage relevant sind, und nicht nur solche, die gut zu den Wortstatistiken passen. Dazu muss die Suchanfrage des Nutzers zunächst mit denselben Attributen vektorisiert werden wie die Dokumentenvektorisierung. Dann kann SWZ verwendet werden, um die Dimensionalität des Abfragevektors auf die gleichen Dimensionen wie die der Dokumentvektoren zu reduzieren. Anschließend kann der Kosinuskoeffizient zwischen dem Abfragevektor und den Dokumentvektoren berechnet werden, um festzustellen, welche Dokumente der Abfrage am nächsten liegen. LSA und SWZ können also zum Aufspüren versteckter Themen in Textdokumenten sowie zur semantischen Suche in großen Dokumentensammlungen verwendet werden.

Es gibt viele Python-Bibliotheken und -Werkzeuge für die Implementierung von LSA und SWZ. Eine der beliebtesten Bibliotheken für Matrix- und Singulärzerlegung ist NumPy. NumPy erleichtert die Durchführung von Matrixoperationen und verfügt über eine Funktion zur Singulärzerlegung: *numpy.linalg.svd()*. Zusammen mit anderen Bibliotheken wie *Pandas* und *Scikit-learn* kann LSA effizient durchgeführt werden. Um LSA zu starten, müssen die Textdaten mit Hilfe des TF-IDF-Modells vektorisiert werden, wie zuvor beschrieben. Dann muss eine singuläre Zerlegung der Term-Dokument-Matrix mit *numpy.linalg.svd()* durchgeführt werden. Danach kann eine bestimmte Anzahl von Hauptkomponenten (Topics) ausgewählt werden, die zur Darstellung der Dokumente verwendet werden. Dies kann durch die Auswahl der Komponenten mit den höchsten Singulärwerten geschehen. Anhand dieser Komponenten kann dann die Nähe zwischen

den Dokumenten bestimmt werden.

Scikit-learn bietet auch eine *TruncatedSVD*-Klasse, mit der eine Truncated-SVD durchgeführt werden kann, um nur die ausgewählten Hauptkomponenten zu erhalten. Diese Klasse erleichtert die Durchführung von LSA auf großen Dokumentenkorpora und verfügt über zusätzliche Funktionen wie Datenverarbeitung und Normalisierung sowie automatische Auswahl der Anzahl der Komponenten auf der Grundlage des Prozentsatzes der erklärten Varianz.

2.3.5 Ansätze für die Erstellung eines Chatbots

Derzeit gibt es vier Hauptansätze für die Erstellung eines Chatbots [Lane, Howard und Hapke 2019]:

- Musterabgleich: Musterabgleich und Antwortvorlagen (vorgefertigte Antworten)
- Grounding: logische Wissensgraphen und das Ziehen von Schlussfolgerungen aus diesen basierend auf diesen Graphen
- Suche: Abrufen von Text
- Generierungsmethoden: Statistik und maschinelles Lernen

Die vier grundlegenden Ansätze zur Erstellung von Chatbots lassen sich kombinieren, was zu benutzerfreundlicheren Chatbots führt. Eine Vielzahl von Anwendungen nutzen alle vier grundlegenden Methoden. Hybride Chatbots unterscheiden sich hauptsächlich darin, wie genau sie diese Ansätze kombinieren und wie viel Gewicht auf jeden einzelnen Ansatz gelegt wird.

Musterabgleich

Bei den ersten Chatbots basierte die Antwort auf die Nachricht eines Benutzers auf einem Mustervergleich. Diese Chatbots suchen nach Mustern im eingehenden Text und geben eine feste (gemusterte) Antwort, wenn eine Übereinstimmung gefunden wird [Woudenberg 2014].

Solche rudimentären Dialogsysteme sind vor allem in automatisierten Benutzerunterstützungssystemen mit interaktiven Sprachmenüs nützlich, wo es möglich ist, das Gespräch an einen Menschen weiterzuleiten, wenn der Chatbot keine Antwortmuster mehr hat.

Da es viele NLP-Dienstprogramme in Python-Paketen gibt, ist es möglich, komplexere Chatbots auf der Grundlage von Mustervergleichen zu erstellen, indem man die Bot-Logik nach und nach direkt in Python mit regulären Ausdrücken und Suchmustern aufbaut.

1995 machte sich Richard Wallace daran, einen allgemeinen Rahmen für die Erstellung von

Chatbots auf der Grundlage des Pattern-Matching-Ansatzes zu schaffen. Zwischen 1995 und 2002 schuf seine Entwicklergemeinschaft die AIML zur Beschreibung von Mustern und Chatbot-Antworten.

AIML ist eine deklarative Sprache, die auf dem eXtensible Markup Language (XML)-Standard basiert, der die Sprachkonstrukte und Datenstrukturen einschränkt, die im Bot verwendet werden dürfen. [*AIML Foundation* o. D.] Ein Chatbot, der auf AIML basiert, sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8"?><aiml version="2.0">
<category>
  <pattern>Hi, Bot</pattern>
  <template>Hallo</template>
</category>
<category>
  <pattern>Wie geht es dir, bot?</pattern>
  <template>Gut</template>
</category>
</aiml>
```

Abbildung 2.11: AIML Chatbot

Eine der Einschränkungen von AIML ist die Art der Muster, die abgeglichen werden können und auf die reagiert wird. Der AIML-Kern (Pattern Matching Engine) reagiert nur auf Eingabetext, der einem vom Entwickler manuell vorgegebenen Muster entspricht. Unscharfe Suchanfragen, Smileys, Satzzeichen, Tippfehler oder falsch geschriebene Wörter sind nicht erlaubt, es findet kein automatischer Abgleich statt. In AIML müssen alle Synonyme manuell einzeln beschrieben werden.

Grounding

Die Grounding-Methode ist ein Ansatz zur Erstellung eines Chatbots auf der Grundlage logischer Wissensgraphen und der Durchführung von Schlussfolgerungen auf der Grundlage dieser Graphen. [Diana und Ismael 2011] Sie wird verwendet, um natürliche Sprache zu verarbeiten und sie dem Verständnis des Bots zuzuordnen. Das Wesentliche an der Grounding-Methode ist, dass der Chatbot nicht nur die Textnachrichten, sondern auch den Kontext und die Umgebung verarbeitet, um Anfragen besser zu verstehen und zu beantworten. Durch die Extraktion von Informationen wird ein Netz von Verbindungen oder Fakten geschaffen. Dieses Netz logischer Verbindungen

zwischen Entitäten - ein Graph oder eine Wissensbasis - kann die Grundlage für die Antworten des Chatbots bilden.

Ein Beispiel für eine Grounding-Methode ist die Verwendung eines Wissensgraphen zur Beschreibung der Umgebung. Ein Wissensgraph enthält Informationen über die Objekte, mit denen der Bot interagieren kann und die Beziehungen zwischen ihnen. Ein Wissensgraph könnte zum Beispiel Informationen über ein Glas auf einem Tisch und das darin befindliche Wasser enthalten. Wenn ein Benutzer eine Frage stellt, verwendet der Chatbot den Wissensgraphen, um den Kontext der Anfrage zu verstehen und die am besten geeignete Antwort abzuleiten. Wenn ein Benutzer zum Beispiel fragt: „Wie hoch ist die Temperatur des Wassers in dem Glas auf dem Tisch?“, kann der Chatbot Informationen aus dem Wissensgraphen verwenden, um die Frage zu beantworten.

Ein solcher Wissensgraph kann abgeleitet werden, um Fragen über die in dieser Wissensbasis enthaltene Welt zu beantworten und anschließend können auf der Grundlage der logischen Antworten die Werte der in den Antworten enthaltenen Template-Variablen ausgefüllt werden, um natürlichsprachliche Antworten zu erstellen. Ursprünglich wurden auf diese Weise Systeme zur Beantwortung von Fragen eingerichtet, wie z. B. der Watson-Bot von IBM (heutzutage wird für ähnliche Systeme jedoch die Informationssuchemethode verwendet). Der Wissensgraph stellt eine Art „Erdung“ des Chatbots in der realen Welt dar.

Die Erstellung von Chatbots auf der Grundlage von „Grounding“ eignet sich hervorragend für Chatbots, die Fragen generieren, bei denen das zur Beantwortung einer Frage erforderliche Wissen in einer umfangreichen Wissensbasis enthalten ist, die aus einer offenen Datenbank (z. B. Wikidata, Open Mind Common Sense oder DBpedia) bezogen werden kann.

Einer der Hauptvorteile der Grounding-Methode besteht darin, dass sie sich an ein sich veränderndes Umfeld anpassen kann. Wenn der Benutzer zum Beispiel ein Glas Wasser von einem Tisch auf einen anderen stellt, wird der Wissensgraph automatisch aktualisiert, um diese Änderung widerzuspiegeln.

Die Grounding-Methode hat jedoch auch ihre Grenzen. So kann es vorkommen, dass bei der Verarbeitung großer Informationsmengen Zusammenhänge nicht berücksichtigt werden und dem Bot möglicherweise verborgen bleiben.

Insgesamt ist die Grounding-Methode ein effektiver Ansatz zur Erstellung wissensbasierter Chatbots. Sie ermöglicht es dem Bot, Benutzeranfragen besser zu verstehen und eine genauere Antwort zu geben.

Suche

Die Informationssuchemethode ist eine der Methoden zum Aufbau von Chatbots, die auf der Extraktion von Informationen aus einer großen Menge von Textinformationen basiert. Die Hauptidee der Informationssuchemethode ist die Analyse des Eingabetextes (Benutzeranfrage), die Auswahl von Schlüsselwörtern und Phrasen daraus und die anschließende Suche nach den relevantesten Informationen in der Wissensdatenbank oder in offenen Quellen. [Diana und Ismael 2011]

Die Wissensbasis kann auch eine Art „Gesprächsprotokoll“ sein, in Form von Aussage-Antwort-Paaren. Dabei sucht der Bot nach früheren Aussagen in den Protokollen früherer Unterhaltungen. Der Bot kann nicht nur in den Protokollen seiner eigenen Gespräche suchen, sondern auch in beliebigen Transkripten von Gesprächen zwischen Menschen, Gesprächen zwischen Menschen und Bots oder sogar Gesprächen zwischen Bots. Aber wie immer gilt: je besser die Eingabedaten, desto besser das Ergebnis. Daher ist es notwendig, die Datenbank früherer Gespräche sorgfältig zu säubern und zu organisieren, damit der Bot nach einem qualitativ hochwertigen Gespräch sucht und es dann imitiert.

Für die Umsetzung der Informationssuchemethode werden verschiedene Algorithmen und Techniken verwendet, z. B. Indizierung und Schlagwortsuche, Kontextsuche, Textanalyse mit Hilfe von maschinellen Lernverfahren usw. Die Informationssuchemethode kann in Python mit verschiedenen Bibliotheken und Tools wie NLTK, Scikit-learn und Gensim implementiert werden.

Einer der ersten Schritte bei der Implementierung einer Informationssuchemethode in Python ist die Vorbereitung der Daten. Dies erfordert Tokenisierung, Lemmatisierung und die Entfernung von Stopp-Wörtern. Als nächstes muss ein Index auf der Grundlage von Schlüsselwörtern erstellt werden. Der Index kann auf der Grundlage von Bag-of-Words oder TF und IDF (TF-IDF-Modelle) erstellt werden. Sobald der Index erstellt ist, kann eine Stichwortsuche durchgeführt werden. Dazu muss die Benutzeranfrage in einen Vektor umgewandelt und mit den Dokumentvektoren im Index verglichen werden. Dies kann mit Hilfe der Scikit-learn-Bibliothek erfolgen [scikit-learn o. D.] Sobald die relevantesten Dokumente gefunden wurden, können sie in eine Rangfolge gebracht und als Antwort auf die Benutzeranfrage angezeigt werden.

Der Vorteil der Informationssuchemethode besteht darin, dass sie ein schnelles und genaues Auffinden der gewünschten Informationen ermöglicht, insbesondere wenn die Wissensbasis gut strukturiert ist und genügend Informationen enthält. Ein Nachteil dieser Methode ist jedoch, dass sie den Kontext der Anfrage nicht berücksichtigt und nicht immer eine vollständige und genaue Antwort auf die Frage des Nutzers liefert. Wenn die Aussage semantisch mit der vom Bot zu beantwortenden übereinstimmt, ist es möglich, die Antwort wortwörtlich und ohne Änderungen wiederzuverwenden. Aber selbst wenn die Datenbank alle möglichen Benutzeräußerungen enthält,

wird der Bot die Persönlichkeiten der Personen widerspiegeln, die diese Äußerungen machen. Wenn die Antworten konsistent sind und von einer Vielzahl von Personen stammen, ist das gut. Problematisch wird es jedoch, wenn die Äußerung, auf die der Bot reagieren soll, vom Gesamtkontext des jeweiligen Gesprächs oder von den Umständen in der Umgebung abhängt, die sich seit der Erstellung des Dialogkorpus geändert haben können.

Beispielsweise sollte der Bot auf die Frage „Wie spät ist es?“ nicht die von der Person gegebene Antwort, sondern die am besten geeignete Aussage aus der Datenbank verwenden. Diese Antwort funktioniert nur, wenn die Zeit, zu der die Frage gestellt wurde, mit der Zeit übereinstimmt, zu der die passende Äußerung aus der Datenbank aufgezeichnet wurde. Neben dem natürlichsprachlichen Text der Äußerung müssen auch ähnliche Informationen über die Zeit - der Kontext (Zustand) - erfasst und verglichen werden. Sie spielt vor allem dann eine wichtige Rolle, wenn die Semantik der Äußerung auf eine aktive Veränderung des im Kontext (Wissensbasis des Chatbots) erfassten Zustands hinweist.

Um den Zustand (Kontext) in einem Chatbot auf der Grundlage der Informationssuche zu berücksichtigen, kann etwas Ähnliches für einen Chatbot mit Musterabgleich durchgeführt werden, da die Auflistung einer Liste von Benutzeraussagen nur eine andere Art ist, ein Muster zu beschreiben. Dies auch ist der Ansatz von Amazon Lex [AMAZON o. D.] und Google Dialogflow [CHAWLA o. D.] Anstatt ein starres Muster zu beschreiben, um den Befehl des Benutzers zu erfassen, können der Dialogflow-Engine einfach ein paar Beispiele geliefert werden. So wie jedes Muster im Chatbot auf der Grundlage der Musterzuordnung einem Zustand zugeordnet wurde, muss auch hier nur die Aussage-Antwort-Beispielpaare mit dem genannten Zustand verknüpft werden.

Der suchbasierte Chatbot indiziert also den Korpus der Dialoge, so dass er leicht frühere Aussagen finden kann, die derjenigen ähnlich sind, auf die er antworten muss und antwortet dann mit einer der passenden Aussagen aus dem Korpus, die er sich „gemerkt“ und für eine schnelle Suche indiziert hat. Im Allgemeinen ist die Methode der Informationssuche eine der gängigsten und beliebtesten Methoden zum Aufbau von Chatbots, die in verschiedenen Bereichen wie Wirtschaft, Medizin, Tourismus und vielen anderen eingesetzt werden.

Generierungsmethoden

Generierungsmethoden sind einer der wichtigsten Ansätze bei der Entwicklung von Chatbots auf der Grundlage künstlicher Intelligenz. Sie ermöglichen es Chatbots, Textantworten auf der Grundlage der Analyse der eingehenden Nachricht und des Kontextes des Dialogs zu generieren. Die folgenden Generierungsmodelle sind nützlich, um einen kreativen Chatbot zu erstellen, der Dinge sagen kann, die noch niemand zuvor gesagt hat:

- Sequenz-zu-Sequenz-Konvertierungsmodelle: Modelle, die darauf trainiert sind, Antworten auf der Grundlage von Eingabesequenzen zu generieren;
- Restricted Boltzmann Machines (RBM): Markov-Ketten, die so trainiert werden, dass sie die „Energie“-Funktion minimieren [Nupur Sharma o. D.];
- Generative Adversarial Network (GAN): statistische Modelle, die darauf trainiert sind, einen Experten, der die Qualität eines Gesprächs bewertet, zu täuschen. [Li u. a. 2017]

Die Vorteile des Einsatzes der Generierungsmethoden:

- Flexibilität: Generative Methoden können für eine breite Palette von Aufgaben eingesetzt werden, einschließlich Texterstellung, Sprachübersetzung, Verarbeitung natürlicher Sprache und mehr.
- Automatisierung: Generative Methoden können auf großen Datensätzen trainiert werden, wodurch die Erstellung von Inhalten automatisiert werden kann.
- Qualität: Generative Methoden zeigen eine hohe Qualität bei der Textgenerierung, Sprachübersetzung und anderen Aufgaben der natürlichen Sprachverarbeitung, wenn sie auf einem ausreichend großen Datensatz trainiert werden.
- Schnelligkeit: Generative Methoden können schneller arbeiten als Menschen, was die Erstellung von Inhalten mit großer Geschwindigkeit ermöglicht.

Die Nachteile der generativen Methoden:

- Große Datenmengen für das Training: Generative Methoden benötigen große Datenmengen für das Training, was bei einigen Aufgaben schwierig sein kann, insbesondere wenn nur ein kleiner Datensatz zur Verfügung steht.
- Sicherheitsrisiken: Generative Methoden können Inhalte erzeugen, die möglicherweise falsch, unvollständig oder irreführend sind. Dies kann zu Sicherheitsrisiken führen, wenn der generierte Inhalt für wichtige Entscheidungen verwendet wird.
- Unterstützungsbedarf: Generative Methoden können erhebliche Unterstützung benötigen, um effektiv zu sein. Dies kann die Modellabstimmung, die Auswahl optimaler Parameter und die Optimierung der Modellleistung auf einer bestimmten Hardwarekonfiguration umfassen.
- Modellbeschränkungen: Generative Methoden können Beschränkungen hinsichtlich der Arten von Inhalten haben, die sie erzeugen können, insbesondere wenn sie nur auf bestimmte Datentypen trainiert wurden.

Eine der beliebtesten Methoden zur Texterstellung ist die sequence-to-sequence-Methode (seq2seq). Die seq2seq-Methode basiert auf Recurrent Neural Networks (RNN), die die Simulation von Datenfolgen ermöglichen. Sie besteht aus zwei Hauptteilen: einem Encoder und einem Decoder. Ein Encoder empfängt eine Wortfolge und baut daraus einen Kontextvektor auf, der Informationen über die Eingabedaten enthält. Der Decoder erhält diesen Vektor als Eingabe und beginnt mit der Generierung einer Folge von Antwortnachrichten, wobei er schrittweise den Kontext und die zuvor generierten Wörter berücksichtigt. [Alammar o. D.]

Einer der Hauptvorteile der seq2seq-Methode ist ihre Fähigkeit, qualitative und grammatikalisch korrekte Textantworten zu generieren, einschließlich Antworten, die nicht in den Trainingsdaten enthalten waren. Sie kann auch mit langen Sequenzen umgehen, was sie ideal für die Generierung von Antworten in Dialogsystemen macht. Darüber hinaus kann die seq2seq-Methode in einer Vielzahl von Anwendungen eingesetzt werden, z. B. in der maschinellen Übersetzung, der Spracherkennung und anderen.

Die seq2seq-Methode hat jedoch ihre Nachteile. Sie erfordert große Datenmengen zum Trainieren und Verarbeiten sowie erhebliche Rechenressourcen. Dies kann die Anwendung der Methode bei einigen Anwendungen einschränken. Wenn der Trainingsdatensatz nicht eine ausreichend große Bandbreite möglicher Antworten repräsentiert, kann das Modell außerdem dazu neigen, vorhersehbare oder falsche Antworten zu erzeugen.

Die Implementierung der seq2seq-Methode in Python kann mit der TensorFlow-Bibliothek erfolgen, die eine Reihe von Werkzeugen für den Aufbau und das Training neuronaler Netze bietet. In TensorFlow kann man die vortrainierten seq2seq-Modelle verwenden oder ein eigenes Modell erstellen, indem die Architektur und die Trainingsparameter des Netzwerks konfiguriert wird. [TensorFlow o. D.[b]]

2.3.6 Mehrschichtiges Perzeptron für ein Mehrklassen-Klassifizierungsproblem

Ein Mehrschichtiges Perzeptron (MLP) ist eine neuronale Netzwerkarchitektur, die aus mehreren Schichten von Neuronen besteht und für Klassifizierungs- und Regressionsaufgaben verwendet werden kann. Mehrschichtige Perzeptronen sind Arten von neuronalen Netzen, die bidirektional sind, da sie die Eingabedaten vorverteilen und die Gewichte zurückverteilen. Ein mehrschichtiges Perzeptron hat eine Eingabeschicht und eine Ausgabeschicht mit einer oder mehreren versteckten Schichten. Bei MLP sind alle Neuronen einer Schicht mit allen Neuronen der nächsten Schicht verbunden. Die Eingabeschicht empfängt Eingangssignale und die gewünschte Aufgabe wird von der Ausgabeschicht ausgeführt. Die versteckten Schichten sind für alle Berechnungen zuständig. Die Architektur des mehrschichtigen Perzeptrons sieht folgendermaßen aus:

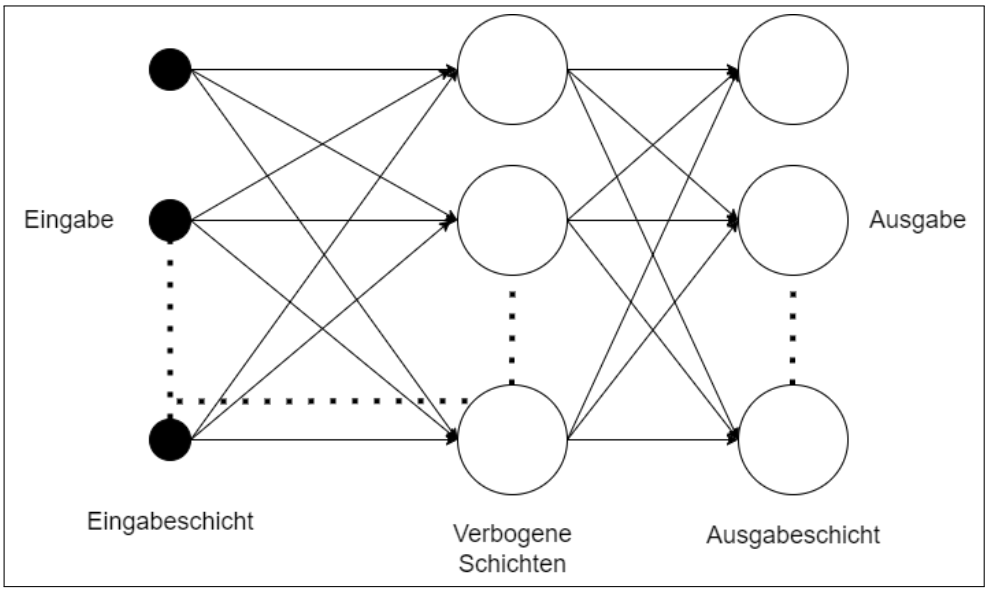

Abbildung 2.12: Mehrschichtiges Perzeptron

Für ein Mehrklassen-Klassifizierungsproblem kann ein MLP mit einer Verlustfunktion wie der kategorialen Kreuzentropie (Categorical Crossentropy) trainiert werden. Zunächst muss ein MLP über eine Eingabeschicht verfügen, die die Merkmale eines Objekts aufnimmt, und eine Ausgabeschicht, die die Wahrscheinlichkeiten für die Zugehörigkeit des Objekts zu den einzelnen Klassen ausgibt. Die Dimensionalität der Ausgabeschicht muss der Anzahl der zu klassifizierenden Klassen entsprechen. Außerdem können zwischen der Eingabe- und der Ausgabeschicht mehrere verborgene Schichten liegen, die nichtlineare Transformationen der Eingabedaten durchführen, so dass das MLP-Modell komplexe Beziehungen zwischen Objektmerkmalen und Klassen erfassen kann. Jedes Neuron im MLP nimmt als Eingabe die gewichtete Summe der Neuronenausgaben der vorherigen Schicht, fügt eine Vorspannung hinzu und durchläuft dann eine Aktivierungsfunktion. Die Aktivierungsfunktion wird verwendet, um dem MLP Nichtlinearität hinzuzufügen und kann z. B. sigmoid-Funktion oder ReLU sein. Beim MLP-Training mit Gradientenabstieg werden die Gewichte und Vorspannungen der Neuronen bei jeder Trainingsiteration aktualisiert, wobei der Wert der Verlustfunktion minimiert wird.

Die kategoriale Kreuzentropie ist eine Verlustfunktion, die für die Klassifizierung mehrerer Klassen verwendet wird und die Diskrepanz zwischen den wahren Klassenbezeichnungen und den vorhergesagten Klassenwahrscheinlichkeiten misst. Daher kann MLP nach dem Training zur Klassifizierung neuer Objekte verwendet werden, indem die Objektmerkmale durch MLP geleitet werden und die Ausgabewahrscheinlichkeiten für jede Klasse ermittelt werden und dann die Klasse mit der höchsten Wahrscheinlichkeit als vorhergesagte Klassenbezeichnung ausgewählt wird.

Die Theorie hinter der kategorialen Kreuzentropie ist die Informationstheorie und die Messung der Unsicherheit. Angenommen, es gibt ein System, das mit einem probabilistischen Modell beschreiben werden muss. Dieses System habe N mögliche Ergebnisse, die jeweils mit den Wahrscheinlichkeiten p_1, p_2, ..., p_N auftreten können, wobei $\sum_i p_i = 1$ und $p_i \geq 0$ für i = 1, 2, ..., N. Dann kann die Entropie H des Systems wie folgt definiert werden:

$$H = -\sum(p_i * \log_2 p_i) \tag{2.9}$$

Dabei wird $\sum(p_i * \log_2 p_i)$ über i von 1 bis N genommen [Koech 2022]. Diese Formel zeigt, dass die Entropie des Systems ein Maß für die Ungewissheit oder Unsicherheit bei der Wahl eines der möglichen Ergebnisse ist.

Wenn die kategoriale Kreuzentropie als Verlustfunktion für ein Mehrklassen-Klassifizierungsproblem verwendet wird, wird erwartet, dass das trainierte Modell einen Ausgabewahrscheinlichkeitsvektor hat, bei dem jedes Element des Vektors einer der Klassen entspricht. Je unsicherer der Modellausgabevektor für ein bestimmtes Beispiel ist, desto höher ist der Wert der kategorialen Kreuzentropie. Umgekehrt ist der Wert der kategorialen Kreuzentropie gleich Null, wenn das Modell eine Klasse für ein bestimmtes Beispiel genau vorhersagt. Beim Training von Klassifizierungsmodellen für Aufgaben des maschinellen Lernens mit mehreren Klassen wird daher versucht, den Wert der kategorialen Kreuzentropie zwischen dem Ausgabevektor des Modells und der richtigen Klasse für jedes Eingabebeispiel zu minimieren. Zu diesem Zweck wird ein Optimierungsverfahren verwendet, z. B. den Gradientenabstieg, um die Gewichte und Verzerrungen des Modells bei jeder Trainingsiteration zu aktualisieren.

In Keras kann die kategoriale Kreuzentropie als Verlustfunktion für verschiedene Modelltypen wie mehrschichtige Perzeptronen, faltungsneuronale Netze und rekurrente neuronale Netze verwendet werden. Es handelt sich um eine Standardverlustfunktion für Mehrklassen-Klassifizierungsprobleme. Bei der Verwendung der kategorialen Kreuzentropie in Keras können auch verschiedene Parameter wie Klassengewichte und Modellbewertungsmetriken angepasst werden, um genauere Klassifizierungsergebnisse zu erhalten. Somit ist die kategoriale Kreuzentropie ein wichtiges Werkzeug beim maschinellen Lernen zur Lösung von Klassifizierungsproblemen mit mehreren Klassen. Ihre Verwendung ermöglicht es, Modelle zu trainieren, die Eingabebeispiele genau klassifizieren und eine hohe Genauigkeit bei Klassifizierungsaufgaben erreichen können.

Kapitel 3

Anforderungen

Im Folgenden sollen die Anforderungen an das Ergebnis der Arbeit konkretisiert werden.

3.1 Antwortzeit

Diese Eigenschaft beschreibt die Zeitdauer vom registrieren eines Sprachbefehls bis zur Ausführung des Befehls durch die Mischmaschine und das Zurückgeben einer Antwort an den Benutzer. Die Antwortzeit spielt eine große Rolle bei der Bedienbarkeit eines interaktiven Systems, um das es sich bei der Mischmaschine handelt. Zu lange Antwortzeiten können dazu führen, dass der Benutzer seine ursprünglichen Ziele vergisst oder in Stress gerät, da in den aller meisten Fällen der Grund für eine lange Antwortzeit vor dem Benutzer verborgen bleibt. Umgekehrt können zu kurze Antwortzeiten ebenfalls zu Stress und Fehlbedienung seitens des Benutzers führen. Dies liegt unter anderem daran, dass kurze Antwortzeiten den Benutzer dazu veranlassen weniger über seine Aktionen und deren Folgen nachzudenken. Als eine für viele Anwendungen geeignete Antwortzeit werden zwei bis vier Sekunden genannt [Herczeg 2018]. Als maximale tolerierbare Antwortzeit werden für dieses Projekt sechs Sekunden festgelegt. Diese vergleichsweise lange Zeitdauer lässt sich zum Einen mit den langwierigen aber notwendigen Berechnungen begründen, die für die Spracherkennung und -verarbeitung benötigt werden. Zum Anderen werden die Auswirkungen einer langen Antwortzeit als gering eingeschätzt, da der Benutzer für diese Anwendung keine Teilarbeitsschritte o. ä. im Gedächtnis behalten muss. Das Ziel des Benutzers sich ein Getränk zubereiten zu lassen ist nach dem Eingang des Sprachbefehls bereits erfüllt.

3.2 Offline-Funktionalität

Die Mischmaschine sollte für die Sprachsteuerung keine Verbindung zum Internet benötigen, da dies die möglichen Einsatzorte der Maschine deutlich einschränken würde. Diese Anforderung schränkt die möglichen, einzusetzenden Technologien zur Spracherkennung und -verarbeitung stark ein, da keine Cloud-Services wie bspw. *Google Cloud Speech* eingesetzt werden können

[Google o. D.] Eine weitere Herausforderung die dadurch entsteht ist, dass Berechnungen die unter Umständen sehr aufwendig sein können nicht ausgelagert sondern auf der Hardware innerhalb der Mischmaschine ausgeführt werden müssen.

3.3 Lautstärke

Die Lautstärke der, von der Mischmaschine zurückgegebenen Antwort, muss laut genug sein, sodass sie vom Benutzer gut verstanden werden kann. Diese Eigenschaft schränkt die Art und Weise wie die Hardware (Computer und Mikrofon) in die Mischmaschine eingebaut werden kann ein und welche Art von Hardware überhaupt verwendet werden kann.

3.4 Entfernung

Mit dieser Eigenschaft ist die Entfernung des Anwenders zu der Mischmaschine gemeint. Es muss dem Anwender ermöglicht werden aus einer moderaten Entfernung mit der Mischmaschine über die Sprachsteuerung zu interagieren. Sowohl die Eingabe eines Befehls über die Sprachsteuerung als auch die zu hörende Antwort sollte mindestens aus einer Entfernung von einem Meter möglich sein. Dafür müssen die Lautsprecher eine bestimmte Lautstärke erreichen können und das Mikrofon eine moderate Empfindlichkeit aufweisen.

3.5 Antworten

Die Antworten, die durch die Mischmaschine an den Benutzer zurückgegeben werden, sollen mit Hilfe eines eigens erstellten Sprachmodells auf Basis von künstlicher Intelligenz und ML erfolgen. Die Antworten der Mischmaschine sollen außerdem bissiger bzw. sarkastischer Natur sein was, je nach verwandter Technik, bei der Auswahl der Trainingsdaten eine große Rolle spielt.

Des Weiteren bestehen die Anforderungen, dass der Benutzer auf Deutsch mit der Mischmaschine kommunizieren können muss und die Antworten der Mischmaschine kontrollierbar sein müssen. Mit der Kontrollierbarkeit ist gemeint, dass Vorhersagen darüber gemacht werden können, was die Mischmaschine in Etwa auf eine bestimmte Frage oder sonstige Benutzereingabe antworten wird. Dies soll verhindern, dass der Benutzer von unerwarteten Reaktionen seitens der Maschine überrascht wird und die Antwroten der Maschine den Benutzer nicht beleidigen (aufgrund der sarkastischen bzw. humorvollen Art und Weise, wie die Maschine antworten soll).

3.6 Kosten

Die Materialkosten sollten einen gewissen Maximalbetrag nicht überschreiten. Zu den benötigten Materialien zählen ein Mikrocomputer zur Durchführung der Spracherkennung und -verarbeitung,

ein Mikrofon zur Aufnahme der Sprache und Lautsprecher zur Tonausgabe. Das Ziel besteht darin einen Betrag von 200€ nicht zu überschreiten.

3.7 Verbrauch von Arbeits- und Festplattenspeicher

Diese Anforderung korreliert mit der Anforderung nach Offline-Funktionalität. Diese bedingt, dass aufwendige, rechen- oder speicherintensive Operationen nicht auf entfernte Rechner ausgelagert werden können sondern, alles auf „kleiner“ Hardware innerhalb der Mischmaschine ablaufen muss. Diese Einschränkung soll anhand eines konkreten Beispiels verdeutlicht werden. Die vierte Version der bekannten Mikrocomputerreihe *Raspberry Pi* umfasst im Modell B maximal acht Gigabyte Read Only Memory (RAM) und einen Micro-SD-Karten-Steckplatz [Ltd, Raspberry Pi o. D.] Zwar sind Micro-SD-Karten von bis zu mehreren hundert Gigabyte erhältlich, jedoch muss dabei der Kostenfaktor mit beachtet werden. Dadurch sind sowohl Primär- als auch Sekundärspeicher stark beschränkt.

Auch Prozessorgeschwindigkeit und Grafikkartenleistung können bei der Spracherkennung- und verarbeitung eine Rolle spielen. Diese ist bei Mikrocomputern ebenfalls eingeschränkt und korreliert negativ mit der Anforderung an Geschwindigkeit. Einen ebenso negativen Einfluss auf die Geschwindigkeit des Gesamtsystems haben die virtuelle Vergrößerung des RAM durch Swapping oder die SD-Karte selbst, welche im Vergleich mit anderen Speichertechnologien in Sachen Geschwindigkeit deutlich das Nachsehen hat.

In anbetracht der geschilderten Herausfoderungen und Kostenbetrachtung wird der maximal zu verbrauchende Arbeits- und Festplattenspeicher für dieses Projekt auf acht Gigabyte RAM und 20 Gigabyte Festplattenspeicher festgelegt.

3.8 Anpassungsfähigkeit

Diese Anforderung beschreibt den Grad der Einfachheit bei Änderung der Anforderungen oder Umwelt die Sprachsteuerung der Mischmaschine an diese neuen Begebenheiten anzupassen. Ein einfaches Beispiel für die Anpassungsfähigkeit des Systems wäre das Hinzukommen eines Behälters innerhalb der Mischmaschine. Tritt dieser Fall ein sollte es leicht möglich sein die Sprachsteuerung so anzupassen, dass der Benutzer auch die Möglichkeit bekommt aus dem neuen, fünften Behälter Getränke zu Mischen und zu Bestellen.

3.9 Bewertung der Ansätze zur Erstellung eines Dialogsystems

Unter Berücksichtigung der genannten Anforderungen und der im Kapitel 2.3.5 besprochenen Ansätze zur Erstellung eines Dyalogsystems, können die folgenden Schlussfolgerungen gezogen werden, ob die Ansätze den Anforderungen entsprechen oder nicht:

Ansatz/ Anforderung	Musterabgleich	Grounding	Suche	Generierungs-methoden
Antwortzeit	entspricht	entspricht nicht	entspricht	entspricht nicht
Offline-Funktionalität	entspricht	entspricht	entspricht	entspricht
Sarkastische Antworten	entspricht	entspricht	entspricht	entspricht nicht
Antworten auf deutsch	entspricht	entspricht nicht	entspricht nicht	entspricht nicht
kontrollierbare Antworten	entspricht	entspricht	entspricht	entspricht nicht
Verbrauch von Arbeits- und Festplattenspeicher	entspricht	entspricht nicht	entspricht nicht	entspricht nicht

Tabelle 3.1: Bewertung der Ansätze zur Erstellung eines Dialodsystems

Kapitel 4

Konzept

In diesem Kapitel wird zunächst erläutert, wie die Steuerung der Getränkemischmaschine durch Sprachbefehle im Allgemeinen ablaufen wird. Anschließend werden mehrere Konzepte vorgestellt, die das allgemeine Konzept konkretisieren. Diese werden anhand der, in Kapitel 4.2 erläuterten Kriterien, bewertet. Zuletzt wird die Wahl des finalen Konzepts begründet.

4.1 Allgemein

Der Benutzer soll über Spracheingaben mit der Mischmaschine interagieren können. Dafür muss das Gesprochene zunächst durch ein Mikrofon aufgenommen werden. Anschließend können die Audiosignale weiterverarbeitet werden. Der Benutzer soll hierbei nicht auf fest vorgegebene Sprachbefehle beschränkt sein, sondern für nahezu jede Eingabe eine sinnvolle Antwort zurückerhalten. Um dies zu gewährleisten wird die Spracheingabe durch ein Sprachmodell, welches mittels maschinellen Lernverfahren trainiert wurde, verarbeitet. Ergebnisse dieser Verarbeitung sind die Antwort, die an den Benutzer zurückgegeben wird, und ein konkreter Befehl für die Mischmaschine. Ein Beispiel für einen solchen Befehl könnte etwa die Zubereitung eines bestimmten Getränks sein. Für die Ausgabe einer Antwort ist ein Lautsprecher notwendig. Denkbar wäre auch eine textbasierte Ausgabe, allerdings ginge damit der Eindruck des Benutzers verloren eine echte Konversation mit der Mischmaschine zu führen. Das Sprachmodell mit der Getränkemischmaschine zu verknüpfen stellt eine Herausforderung dieser Arbeit dar.

4.2 Bewertungskriterien

Im Folgenden sind die Bewertungskriterien für die einzelnen Konzepte aufgelistet:

- Freiheitsgrade in der Spracheingabe des Benutzers: Erhält der Benutzer passende Antworten zurück egal was er sagt oder ist er auf einige wenige Befehle beschränkt?

- Hardwarekosten: Wie kostspielig ist das Konzept bezüglich der zusätzlich benötigten Hardware?
- Verfügbare Rechenleistung: Wie hoch ist die Verfügbare Rechenleistung im Vergleich zu den anderen Konzepten? Reicht diese aus um das Sprachmodell auszuführen?
- Performanz: Als wie performant wird die Lösung eingeschätzt? Ist mit Latenzen zwischen der Einagbe des Benutzers, der Ausgabe einer Antwort und Ausführung der Aktion zu rechnen?
- Overhead: Wie hoch ist im Allgemeinen der Mehraufwand einzuschätzen?

4.3 Konzept A: Spracherkennung und -verarbeitung mittels Arduino

Ein erstes Konzept sieht vor, dass das Audiosignal direkt von einem der Arduinos in der Getränkemischmaschine aufgenommen wird. Das Audiosignal wird vom Arduino interpretiert und eine passende Antwort wird ausgegeben. Außerdem sendet der Arduino die entsprechenden Signale, um die vom Benutzer gewünschte Aktion von der Getränkemischmaschine ausführen zu lassen. Ein Problem ist hierbei die Interpretation des Audiosignals durch den Arduino, da dessen Leistung nicht für das Ausführen eines Sprachmodells ausreicht. Folglich muss dieser Prozess ausgelagert werden. Das Konzept wird deshalb um ein cloudbasiertes Sprachverarbeitungssystem ergänzt, welches den Sprachbefehl des Benutzers vom Arduino entgegennimmt und einen passenden Befehl und eine passende Antwort zurückgibt (s. Abb. 4.1). Die Kommunikation zwischen Arduino und Cloudsystem kann über das Hypertext Transfer Protocol (HTTP) erfolgen.

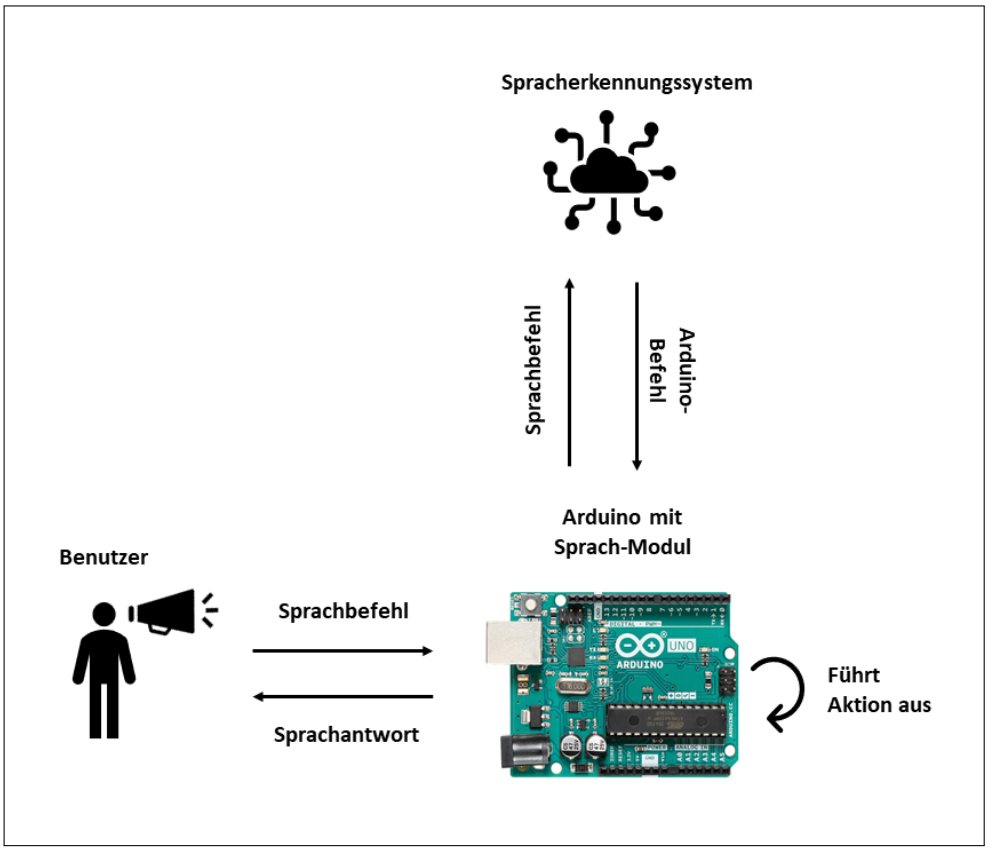

Abbildung 4.1: Spracherkennung und -verarbeitung mittels Arduino

Es muss ein geeignetes Format zum Versenden des Sprachbefehls über HTTP gefunden werden. Eine Möglichkeit besteht darin, das eingehende Audiosignal im Arduino in textform umzuwandeln und diesen String zu versenden. Die auf dem Markt verfügbaren Arduino-Sprach-Module sind jedoch nicht in der Lage beliebige Spracheingaben in Text umzuwandeln, sondern bieten diese Funktionalität nur für vordefinierte Werte an. Dies würde das Ziel dieser Arbeit verfehlen, dem Benutzer eine Konversation mit der Mischmaschine zu ermöglichen. Ein weiteres Problem dieser Lösung besteht darin, dass beispielsweise bei wechselnder Getränkeauswahl die zur Verfügung stehenden Sprachbefehle wie „Ich hätte gerne Getränk xy“ jedes Mal aufs neue manuell angepasst werden müssten. Dies hat zur Folge, dass auch die reine Spracherkennung aus der Mischmaschine ausgelagert werden muss. Ein denkbares Format sind die rohen Audiosignale, die vom Arduino aufgenommen werden.

4.4 Konzept B: Spracherkennung und -verarbeitung mittels mobiler Anwendung

Die Audiosignale über ein Mikrofon in der Mischmaschine aufzunehmen und eine Antwort über einen Lautsprecher auszugeben, so wie es in Konzept A der Fall ist, kann ein Problem darstellen. Zum Einen wird dadurch zusätzliche Hardware benötigt und zum Anderen muss diese korrekt verbaut werden. Das Tonsignal muss vom Mikrofon in einer guten Qualität aufgenommen werden können und die Antwort aus dem Lautsprecher für den Benutzer verständlich sein. Konzept B umgeht dieses Problem durch den Einsatz einer mobilen Anwendung, die durch den Benutzer installiert wird. Über diese Anwendung können anschließend die Aufnahme der Audiosignale, die Spracherkennung und die Kommunikation mit dem Sprachverarbeitungsservice und der Mischmaschine abgewickelt werden, wie in Abbildung 4.2 zu sehen ist.

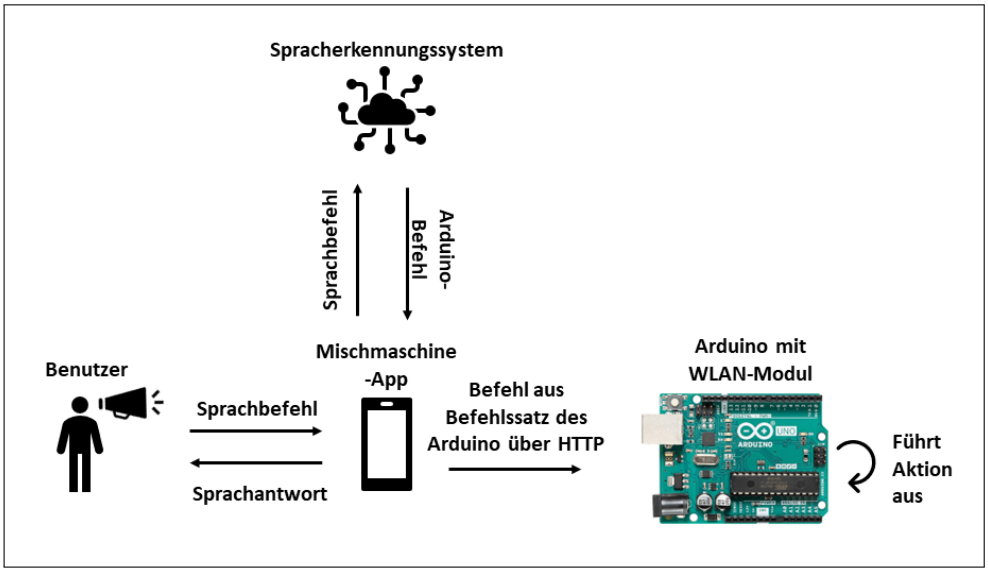

Abbildung 4.2: Spracherkennung und -verarbeitung mittels mobiler Anwendung

Ein Problem dieser Lösung ist der offensichtliche Mehraufwand durch die Entwicklung einer eigenen Anwendung für Mobiltelefone. Auch der Anwender hat zusätzlichen Aufwand durch die Installation. Außerdem ist die Spracheingabe und -ausgabe über das Mobiltelefon nicht intuitiv, da der Anwender eigentlich mit der Maschine kommunizieren sollte. Dieser Effekt kann dadurch abgeschwächt werden, dass wenigstens die Antwort durch einen Lautsprecher in der Mischmaschine an den Benutzer zurückgegeben wird.

4.5 Konzept C: Spracherkennung und -verarbeitung auf Computer-Hardware

Ein weiteres Konzept stützt sich auf die Verwendung eines Computers in der Mischmaschine anstelle eines Mikrocontrollers wie dem Arduino. Motivation ist hierbei der Leistungsgewinn gegenüber eines Mikrocontrollers, um die Spracherkennung und -verarbeitung mittels Sprachmodell zu gewährleisten. Ein Beispiel für einen solchen Miniaturcomputer ist der Raspberry-Pi. Dieser bietet genügend Schnittstellen, wie etwa USB-Hubs, zum verbinden von Mikrofon als auch Lautsprecher. Nimmt der Computer das Audiosignal auf verarbeitet er dieses und generiert daraus die Antwort, die durch den Lautsprecher ausgegeben wird, zusammen mit der Aktion für die Getränkemischmaschine. Diese muss an den Arduino, welcher die Mischmaschine steuert, übermittelt werden. Um dies zu ermöglichen können der Computer und der Arduino über eine serielle Schnittstelle, wie etwa einem USB-Kabel, miteinander verbunden werden. Abbildung 4.3 stellt den konzeptionellen Aufbau graphisch dar.

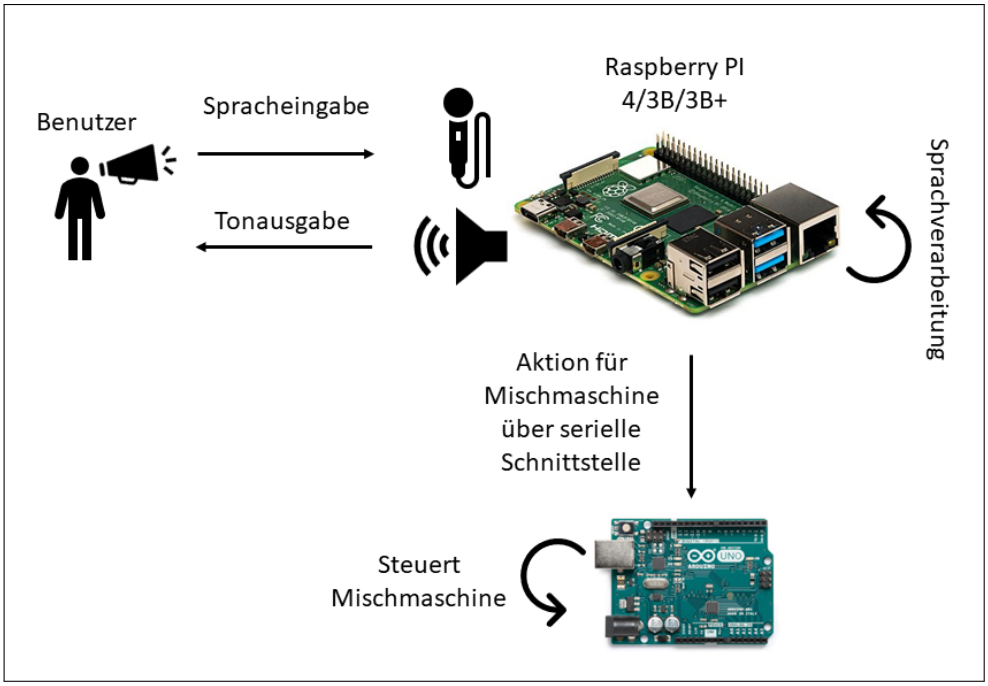

Abbildung 4.3: Spracherkennung und -verarbeitung auf Computer-Hardware

Obwohl ein Computer wie der Raspberry-Pi im Allgemeinen eine höhere Leistung als ein Mikrocontroller hat ist damit nicht sichergestellt, dass diese zur Ausführung des Sprachmodells ausreicht. Beispielsweise ist das vierte Modell der Raspberry-Pi-Serie mit nur maximal acht Gigabyte Arbeitsspeicher erhältlich. Das Sprachmodell könnte allerdings noch weitaus mehr Daten im Arbeitsspeicher benötigen. Des Weiteren ist zu beachten, dass die Miniaturcomputer von Raspberry-Pi im Speziellen zum Zeitpunkt dieser Arbeit kaum zu vertretbaren Preisen

verfügbar sind.

4.6 Finales Hardware-Konzept

Die folgende Tabelle zeigt eine Übersicht bezüglich der Bewertung der einzelnen Konzepte.

Bewertungsmatrix	Konzept A (nur Arduino)	Konzept A	Konzept B	Konzept C
Freiheitsgrade in der Spracheingabe des Benutzers	niedrig	sehr hoch	sehr hoch	hoch
Hardwarekosten	niedrig	hoch	niedrig	sehr hoch
Verfügbare Rechenleistung	niedrig	sehr hoch	sehr hoch	hoch
Performanz	sehr hoch	mittel	mittel	sehr hoch
Overhead	niedrig	hoch	sehr hoch	niedrig

Tabelle 4.1: Bewertung der Konzepte

Ein erster Ansatz wurde im Kapitel 4.3 zu Konzept A erläutert. Hierbei war die Idee, die Spracherkennung und -verarbeitung nur auf dem Arduino auszuführen. Aufgrund der mangelnden Leistung eines Arduinos können mit Hilfe von Sprachmodulen jedoch nur vordefinierte Sätze oder Wörter erkannt werden. Damit ist der Freiheitsgrad in der Spracheingabe des Benutzers äußerst eingeschränkt. Dafür sind die aufzuwendenden Hardwarekosten minimal. Lediglich das Sprachmodul sowie ein Lautsprecher müssten besorgt werden. Die Performanz wird als sehr hoch eingeschätzt, da die Spracherkennung direkt auf dem Ardunino erfolgen kann, der auch die Mischmaschine steuert. Niedrig ist hingegen der benötigte Mehraufwand, da kaum zusätzliche Awendungen, Services oder Hardwarekomponenten benötigt werden.

Das Konzept wurde schließlich durch einen Sprachverarbeitungsservice in der Cloud ergänzt (Konzept A). Der Benutzer hat bei diesem Ansatz große Freiheiten in seinen Formulierungen, da es kein Problem darstellt ein großes Sprachmodell in der Cloud auszuführen. Die Hardwarekosten könnten allerdings hoch sein, je nach dem, ob der Server selbst bereitgestellt oder von einem externen Anbieter bezogen wird. Auch die letzten Endes tatsächlich benötigte Rechenleistung spielt dabei eine Rolle. Die verfügbare Rechenleistung ist theoretisch unbegrenzt, wobei die Performanz des Gesamtsystems nur als mittelmäßig eingestuft werden kann. Nach der Aufnahme und eventuell einer Vorverarbeitung des Aduiosignals durch den Arduino müssen HTTP-Nachrichten gesendet und Empfangen werden. Je nach Last auf dem Netzwerk kann es dadurch zu Latenzen oder sogar Verbindungsabbrüchen kommen. Außerdem ist der Overhead durch den Einsatz einer Cloud recht hoch.

Konzept B unterscheidet sich in Sachen Freiheitsgrad, Rechenleistung und Performanz nicht von Konzept A, da auch hier eine Cloud zum Einsatz kommt. Die Hardwarekosten sind jedoch niedriger, da immerhin kein Sprachmodul, Mikrofon und Lautsprecher benötigt werden. Die Aufgaben dieser Komponenten kann das Mobiltelefon des Anwenders übernehmen. Der Overhead ist deutlich größer, da das Konzept die Entwicklung einer eigenen Mobilanwendung voraussetzt.

Der Freiheitsgrad wird bei Konzept C als hoch, jedoch nicht als sehr hoch, bewertet. Grund hierfür ist die, im Vergleich zur Cloud, etwas beschränkte Leistung, welche die Leistung/Größe des Sprachmodells beeinträchtigen könnte. Hardwarekosten können jedoch sehr hoch werden. Bei dem Einsatz einer Cloud kann ein günstiger Anbieter gefunden werden, sodass die Anschaffung eigener Hardware entfällt. Dies ist hier nicht der Fall. Die Performanz des Gesamtsystems kann, wie bei Konzept A (nur mittels Arduino), sehr hoch eingeschätzt werden, da Spracherkennung und -verarbeitung direkt in der Mischmaschine von der Hardware übernommen wird. Der Mehraufwand ist gering, da weder ein Cloudservice noch eine externe Anwendung entwickelt werden müssen.

Letzten Endes wurde das Konzept C als finales Hardware-Konzept gewählt. Wie ein Blick in die Bewertungsmatrix zeigt lässt sich dies mit den sehr guten Werten, die sich aus der Betrachtung der beschriebenen Bewertungskriterien ergaben, begründen.

4.7 Konzept für die Sprachsteuerung

4.7.1 Ansatz für das Dialogsystem

Aus dem Vergleich der wichtigsten Ansätze zur Erstellung von Chatbots, die in Kapitel 2.3.5 besprochen wurden, lassen sich die folgenden Vor- und Nachteile der einzelnen Methoden ableiten:

Ansatz	Vorteile	Nachteile
Musterabgleich	• Einfacher Einstieg • Leicht wiederverwendbar • Modularität • Leicht zu kontrollieren/ einzuschränken	• Themenbereich begrenzt • Die Möglichkeiten sind durch die Arbeitsbelastung des Entwicklers begrenzt • Komplexität der Fehlersuche • Strenge und „spröde“ Regeln
Grounding	• Gut im Beantworten logischer Fragen • Leicht zu kontrollieren/ einzuschränken	• Künstlicher, mechanischer Ton • Probleme mit Zweideutigkeiten • Probleme mit dem Allgemeinwissen • Begrenzt auf strukturierte Daten • Erfordert die Extraktion von Informationen in großem Umfang • Erfordert menschliche Aufsicht
Suche	• Einfachheit • Leicht zu lehren • Simulation von menschlicher Konversation	• Unzureichende Skalierung • Die simulierte Persönlichkeit des Bots ist inkonsistent • Kennt den Kontext nicht • Keine sachlichen Fragen
Generierungs-methoden	• Neue, kreative Dialoge • Weniger Arbeit für den Entwickler • Kontextsensitiv	• Schwierig zu lehren • Erfordert mehr Daten (Dialoge) • Schwierig, in die richtige Richtung zu lenken • Erfordert mehr Rechenleistung

Tabelle 4.2: Bewertung der Ansätze für die Erstellung eines Dialogsystems

Bei der Analyse des Problems, ein Dialogsystem für eine Getränkmischmaschine zu entwickeln, kann man zu dem Schluss kommen, dass die beste Option eine Mischung aus dem Ansatz der Informationssuchemethode und dem Musterabgleich ist.

Das erste Argument, das für diesen Ansatz spricht, ist die Möglichkeit, die Sprachsteuerung in deutscher Sprache zu verwenden, was die Anwendung der Generierungsmethoden erschwert, da der Zugang zu einer geeigneten Datenbank in dieser Sprache, die den Anforderungen des Projekts, nämlich eine ausreichende Anzahl von Beleidigungen zu enthalten, nicht möglich ist. Gleichzeitig können bei der Verwendung von Musterabgleich Antwortvorlagen und Muster für

entsprechende Anfragen in deutscher Sprache im Voraus erstellt werden, was die Erstellung und das Training des Dialogsystems erleichtert.

Das zweite Argument ist, dass in diesem Projekt ein Raspberry Pi verwendet wird, was die Verwendung generativer Methoden aufgrund der begrenzten Hardware-Ressourcen einschränken kann. Andererseits kann das Modell für die Informationssuche und den Musterabgleich auf Geräten mit geringem Stromverbrauch implementiert werden, was diesen Ansatz für dieses Projekt vorteilhaft macht.

Generierungsmethoden könnten auch für dieses Problem zu mächtig sein. Beim Mixen von Cocktails können die Antworten einfach und formelhaft sein, wie z. B. „Das hört sich eklig an, bist du sicher, dass du es willst?“ oder „Ich hoffe, ich sehe dich nie wieder“. In diesem Fall kann der Einsatz von Generierungsmethoden wie seq2seq-Modellen überflüssig und ineffizient sein. Für diese Aufgabe ist im Gegensatz zu komplexen natürlichsprachlichen Abfragen keine detaillierte semantische Verarbeitung erforderlich, so dass der Musterabgleich einen einfacheren und effizienteren Ansatz darstellt.

Bei der Verwendung des Musterabgleiches kann man den Ton und den Humor des Geräts leicht steuern und so die gewünschte Atmosphäre erzeugen. Die Antworten des seq2seq-Modells sind wiederum sehr schwer zu steuern. Generierungsmethoden können zu unerwünschtem Maschinenverhalten führen, wenn das Modell auf ungeeigneten Daten trainiert wird oder Fehler in der Betriebslogik enthält. Die Verwendung einer Datenbank, die genügend Beleidigungen enthält (z. B. die 4chan-Datenbank), kann dazu führen, dass die Antworten der Maschine über das Ziel hinausschießen und statt lustig zu sein, den Benutzer beleidigen.

Mit diesem Ansatz wird auch die Effizienz der Maschinensteuerung verbessert. Beim Musterabgleich kann eine Kategorie „Getränkebestellung“ zugewiesen werden, die, wenn sie erkannt wird, die Maschine automatisch zur Bearbeitung der Befehle veranlasst. Im Falle von seq2seq-Modell ist keine Kategorie vorgesehen, so dass eine zusätzliche Prüfung jeder Eingabeanweisung eingeführt werden müsste, um festzustellen, wann der Benutzer die Bestellung aufgegeben hat.

Ein letztes Argument, das für den Ansatz spricht, ist die Möglichkeit, die Maschine bei Bedarf schnell an neue Anfragen und Anforderungen anzupassen, indem neue Muster und Regeln in das System eingeführt werden. Daher ist eine Mischung aus Informationssuchemethode und Musterabgleich für das vorliegende Problem am besten geeignet.

4.7.2 Befehle

Folgendes Übergangsdiagramm beschreibt die Abfolge von Aktionen und Übergängen zwischen Zuständen bei der Analyse der Befehle:

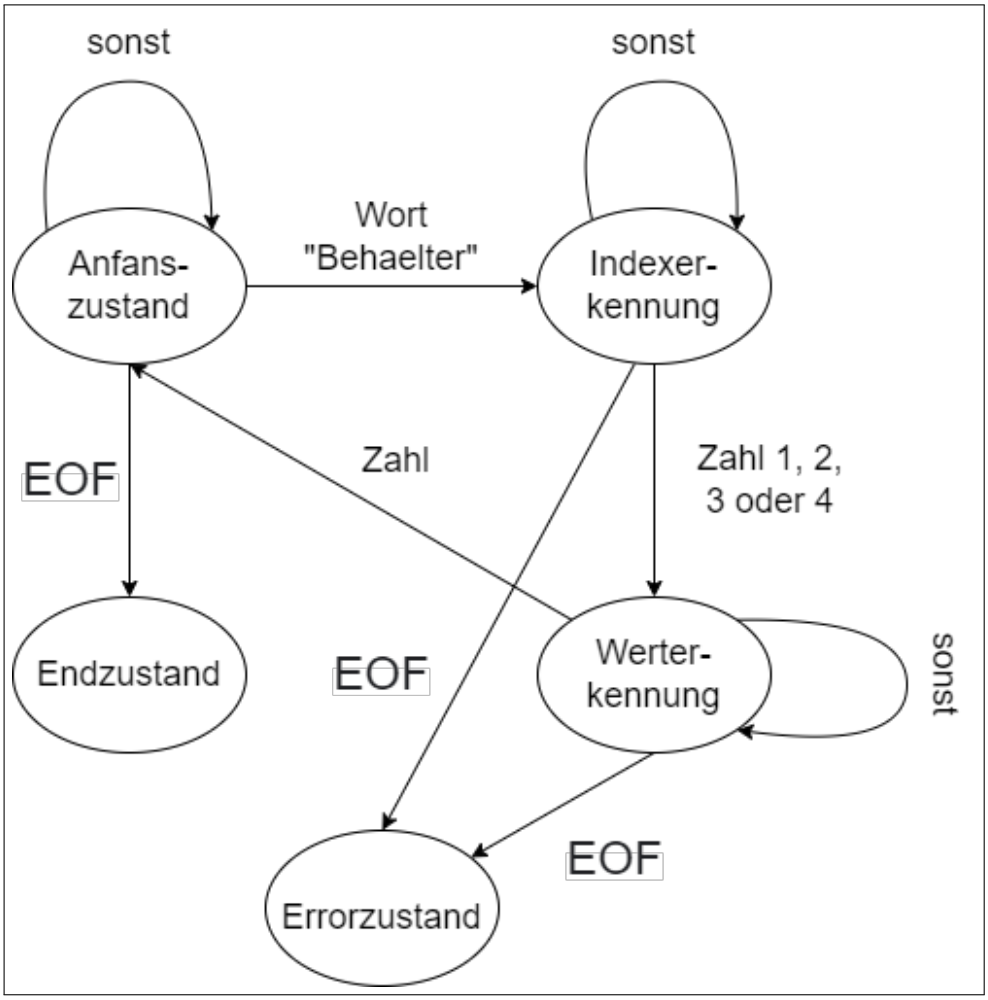

Abbildung 4.4: Übergangsdiagramm beim Analysieren der Befehle

In diesem Fall erhält die Methode eine Zeichenkette mit Informationen über den prozentualen Anteil von Flüssigkeiten aus verschiedenen Behältern und muss den prozentualen Anteil jeder Flüssigkeit zurückgeben.

Der erste Zustand „Anfangszustand“ bedeutet, dass die Methode gestartet wird. Dann verarbeitet die Methode nacheinander jedes Wort der Eingabezeichenkette. Jedes Eingabewort, außer Wort „Behälter“ wird als irrelevant wahrgenommen und erstmal ignoriert.

Sobald die Maschine das Wort „Behälter“ hört, geht die Methode zum nächsten Zustand

über - der Indexerkennung. Nach den Regeln kommt der Index nach dem Wort „Behälter“, aber zwischen dem Wort „Behälter“ und der Indexzahl können andere Wörter stehen, die für diese Methode nicht relevant sind, z. B. „Nummer“ oder „mit Nummer“. Daher werden diese Wörter ignoriert, und die Maschine bleibt in demselben Zustand, bis sie eine Zahl zwischen eins und vier hört.

Sobald der Index ermittelt ist, geht die Methode zum nächsten Zustand über - der Wertermittlung. In diesem Fall werden die Wörter, angefangen mit dem Wort „Behälter“, in umgekehrter Reihenfolge betrachtet, da der Wert nach den Regeln der Grammatik vor dem Wort „Behälter“ stehen wird. Alle Wörter bis zur ersten gefundenen Zahl werden ignoriert. Sobald die Maschine eine beliebige Zahl hört, kehrt die Methode in den Ausgangszustand zurück.

Die Analyse der Befehle kann dann zum Endzustand zurückkehren, wenn sie sich aktuell im Anfangszustand befindet und alle Wörter gelesen wurden. Sobald die Zeichenkette in einem anderen Zustand beendet wird, wird die Methode zum Errorzustand zurückkehren.

Kapitel 5

Implementierung

Dieses Kapitel beschäftigt sich detailliert mit den verwendeten Techniken und der Vorgehensweise zur Implementierung und Umsetzung der Sprachsteuerung für die Getränkemischmaschine.

5.1 Implementierung des Sprachverarbeitungssystems

Um die Möglichkeit des Dialogs mit der Getränkmischmaschine zu realisieren, war es notwendig, ein Modell für das Dialogsystem zu implementieren. Das Projekt des Dialogsystems besteht aus den folgenden Komponenten:

- train_dialog_model.py - der Code zum Einlesen der natürlichsprachlichen Daten in einen Trainingssatz und zur Verwendung eines sequenziellen neuronalen Netzes von Keras zur Erstellung eines Modells für die Klassifizierung.
- model_classes.pkl - eine Liste verschiedener Klassen von Antworten.
- words_for_pattern.pkl - eine Liste verschiedener Wörter, die für die Mustererkennung verwendet werden können.
- patterns.json - ein Stapel von JavaScript-Objekten, die verschiedene Tags auflisten, die verschiedenen Arten von Wortmustern entsprechen.
- dialog_model.h5 - das von train_dialog_model.py erstellte Modell.

Diese Komponenten wurden sowohl zur Erstellung eines Deep Learning Modells für Klassifizierung und zum Trainieren dieses Modells als auch zur Implementierung des Dialogsystems und Befehlverarbeitungssystems verwendet.

5.1.1 Implementierung des Klassifizierungsmodells

Für die Implementierung des Modells wurden folgenden Python-Bibliotheken und Klassen verwendet:

- ‘nltk’: eine Python-Bibliothek für die Arbeit mit natürlichen Sprachdaten. Es bietet Werkzeuge für Aufgaben wie Tokenisierung, Stemming, Lemmatisierung, Part-of-Speech-Tagging und mehr.
- ‘nltk(‘punkt’)’: die NLTK-Daten, die für die Tokenisierung benötigt werden, bei der ein Text in einzelne Wörter zerlegt wird.
- ‘nltk’(‘wordnet’): die NLTK-Daten, die für die Lemmatisierung benötigt werden, d. h. die Reduzierung eines Wortes auf seine Grund- oder Wurzelform.
- ‘json’: Python-Bibliothek für die Arbeit mit JavaScript Object Notation (JSON)-Daten, einem leichtgewichtigen Format für den Datenaustausch. Sie bietet Methoden zur Kodierung von Python-Objekten in das JSON-Format und zur Dekodierung von JSON-Daten in Python-Objekte.
- ‘pickle’: eine Python-Bibliothek zur Serialisierung und Deserialisierung von Python-Objekten. Sie kann verwendet werden, um ein Python-Objekt in einen Byte-Stream zu konvertieren, der in einer Datei gespeichert oder über ein Netzwerk übertragen und später wieder in ein Python-Objekt deserialisiert werden kann.
- ‘numpy’: eine Python-Bibliothek für numerische Berechnungen. Sie bietet Unterstützung für große, mehrdimensionale Arrays und Matrizen sowie eine große Sammlung mathematischer Funktionen, die mit diesen Arrays arbeiten.
- ‘random’: eine Python-Bibliothek zur Erzeugung von Zufallszahlen und zur Durchführung von Zufallsoperationen. Sie bietet Methoden zur Erzeugung zufälliger Ganzzahlen, zur Auswahl zufälliger Elemente aus einer Liste, zum zufälligen Mischen einer Liste und mehr.
- ‘keras.models.Sequential’: eine Klasse, die von der Keras-Bibliothek bereitgestellt wird, einer in Python geschriebenen API für neuronale Netze auf hoher Ebene. Die Klasse *Sequential* wird verwendet, um einen linearen Stapel von Schichten für ein neuronales Netzwerkmodell zu erstellen.
- ‘keras.layers.Dense’: eine Klasse, die von der Keras-Bibliothek bereitgestellt wird, um eine vollständig verbundene neuronale Netzwerkschicht zu erstellen.
- ‘keras.layers.Activation’: eine von der Keras-Bibliothek bereitgestellte Klasse zur Anwendung von Aktivierungsfunktionen auf die Ausgabe einer Schicht eines neuronalen Netzes.
- ‘keras.layers.Dropout’: eine von der Keras-Bibliothek bereitgestellte Klasse zur Anwendung der Dropout-Regularisierung auf die Ausgabe einer neuronalen Netzwerkschicht.

- 'keras.optimizers.SGD': eine von der Keras-Bibliothek bereitgestellte Klasse zur Definition des stochastischen Gradientenabstiegs-Optimierers für das Training eines neuronalen Netzes.

Als erste wurden alle Listen initialisiert, in denen die Daten der natürlichen Sprache gespeichert werden. Dafür gibt es die „patterns" json-Datei, die die „Vorsätze" enthält. Die Datei sieht folgendemaßen aus:

```
{
    "intents": [
      {
        "tag": "greeting",
        "patterns": ["Hallo", "Hallo Maschine"],
        "responses": [
          "Hey du Looser", "Seit wann kann Luft reden"],
        "context": [""]
      },
      {
        "tag": "goodbye",
        "patterns": ["Tschüss", "Auf Wiedersehen"],
        "responses": ["Endlich", "Geh einfach"],
        "context": [""]
      }
    ]
}
```

Abbildung 5.1: Patterns-Datei

Das json-Modul wird verwendet, um die Datei zu laden und als Variable zu speichern.

```
1 for intent in intents['intents']:
2     for pattern in intent['patterns']:
3         w = nltk.word_tokenize(pattern)
4         words_for_pattern.extend(w)
5         documents.append((w, intent['tag']))
6         if intent['tag'] not in model_classes:
7             model_classes.append(intent['tag'])
```

```
words_for_pattern = [lemmatizer.lemmatize(w.lower()) for w in words_for_pattern if w not in ignore_words]
words_for_pattern = sorted(list(set(words_for_pattern)))
model_classes = sorted(list(set(model_classes)))
```

Listing 5.1: Speichern der Patterns

Listing 5.1 zeigt, dass eine verschachtelte for-Schleife alle bereits tokenisierte Wörter in „patterns“ extrahiert und sie der „words_for_pattern“ Variable hinzufügt. Anschließend werden jedes Paar von Mustern mit dem entsprechenden Tag einer Dokumentenliste hinzugefügt. Auch die Tags werden in eine Klassenliste („model_classes“) aufgenommen. Als Nächstes wird die Wortliste genommen und alle Wörter darin lemmatisiert und kleingeschrieben. Nach der Sortierung der Liste sind die Daten für das Training eines Deep Learning Modells vorbereitet.

```
training = []
output_empty = [0] * len(model_classes)
for doc in documents:
    bag = []
    pattern_words = doc[0]
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]
    for w in words_for_pattern:
        bag.append(1) if w in pattern_words else bag.append(0)

    output_row = list(output_empty)
    output_row[model_classes.index(doc[1])] = 1
    training.append([bag, output_row])

random.shuffle(training)
training = numpy.array(training)
train_x = list(training[:,0])
train_y = list(training[:,1])
```

Listing 5.2: Initialisierung der Test- und Trainingsdaten

Wie in Listing 5.2 gezeigt wird, werden zunächst die Trainingsdaten mit einer Variable „training“ initialisiert. Es wird eine verschachtelte Liste erstellt, die BOW für jedes einzelne Dokument enthält. Das Merkmal *output_row* dient einfach als Schlüssel für die Liste. Dann werden die Trainingsmengen gemischt und in Training und Test aufgeteilt, wobei die Patterns die Variable X und die Intents die Variable Y sind. Da die Trainings- und Testdaten fertig sind, wird jetzt ein Deep-Learning-Modell von Keras namens *Sequential* verwendet.

```
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

hist = model.fit(numpy.array(train_x), numpy.array(train_y), epochs=500, batch_size=5, verbose=1)
model.save('dialog_model.h5', hist)
```

Listing 5.3: Training des Modells

Listing 5.3 zeigt, dass in diesem Fall ein MLP-Modell verwendet wird, um das Problem der Mehrklassenklassifizierung zu lösen. MLP ist eine der häufigsten Arten von neuronalen Netzen,

die aus mehreren Schichten bestehen, die jeweils mehrere miteinander verbundene Neuronen enthalten. In der Regel werden mehrere versteckte Schichten zwischen der Eingabe- und der Ausgabeschicht verwendet, damit das Modell komplexere Merkmale aus den Eingabedaten extrahieren und die Qualität der Klassifizierung verbessern kann. In diesem Fall besteht das Modell aus drei Schichten: Die erste Schicht enthält 128 Neuronen, die zweite Schicht 64 Neuronen und die dritte Schicht (Ausgabeschicht) enthält die Anzahl der Neuronen, die der Anzahl der Klassen (Absichten) entspricht. Für die Aktivierung der Neuronen in den ersten beiden versteckten Schichten wird die Aktivierungsfunktion Rectified Linear Unit (ReLU) verwendet, für die Aktivierung der Neuronen in der Ausgabeschicht wird die Funktion Softmax verwendet, um die Zugehörigkeitswahrscheinlichkeiten der Eingabebeispiele zu jeder Klasse zu erhalten.

Als Verlustfunktion wird kategoriale Kreuzentropie (categorical_crossentropy) verwendet, die Standardverlustfunktion für Mehrklassen-Klassifizierungsaufgaben in Keras. Der Optimierer, der zum Trainieren des Modells verwendet wird, ist der stochastische Gradientenabstieg mit beschleunigtem Nesterov-Gradienten (SGD mit beschleunigtem Nesterov-Gradienten), der ein schnelleres Lernen und eine höhere Genauigkeit bei Klassifizierungsaufgaben ermöglicht. Der gesamte Quellcode für das Klassifierzierungsmodell lässt sich in Anhang E finden.

Nachdem das Modell trainiert wurde, wird das Ganze in ein Numpy-Array umgewandelt und als *dialog_model.h5* gespeichert und kann für den Dialog mit der Getränkemischmaschine verwendet werden.

5.1.2 Implementierung des Dialogsystems

Für die Implementierung des Dialogsystems müssen zunächst die notwendigen Bibliotheken und Werkzeuge importiert werden: ‘nltk’, ‘nltk(‘punkt’)’, ‘nltk(‘wordnet’)’, ‘pickle’, ‘json’, ‘numpy’, ‘random’ und ‘re’ (das Modul für reguläre Ausdrücke), sowie die Methode ‘load_model’ von ‘keras.models’. Dann werden das trainierte Modell, die Patternsdatei und das Vokabular für das Modell von der Festplatte geladen. Die Funktion ‘load_model’ von Keras wird verwendet, um ein trainiertes neuronales Netzwerkmodell aus einer .h5-Datei zu laden. Die Datei ‘patterns.json’ enthält vordefinierte Intents und die entsprechenden Antworten, während die Dateien ‘words_for_pattern.pkl’ und ‘model_classes.pkl’ Vokabular- und Klasseninformationen für das Modell enthalten. Wie aus dem Quellcode in Anhang E hervorgeht, besteht die Antwortgenerierung der Getränkemischmaschine aus den folgenden Methoden:

- ‘get_response(msg)’: Zurückgabe der Antwort und der extrahierten Behälterwerte.
- ‘get_dialog_response(ints, intents_json)’: Antwortgenerierung.
- ‘predict_class(sentence, model)’: Voraussage der Klasse des Satzes.

- 'bow(sentence, words)': Erstellung einer BOW-Darstellung des Satzes.
- 'clean_up_sentence(sentence)': Tokenisierung und Lemmatisierung der Eingabezeichenkette.

```
def clean_up_sentence(sentence):
    sentence_words = nltk.word_tokenize(sentence)
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
    return sentence_words
```

Listing 5.4: Preprocessing des Satzes

Listing 5.4 zeigt, dass die 'clean_up_sentence' Methode einen Satz als Eingabe nimmt und folgende Operationen durchführt: Sie tokenisiert den Satz mit der Methode 'nltk.word_tokenize', lemmatisiert jedes Wort (wandelt es in seine Grundform um) mit dem *WordNetLemmatizer* von NLTK, wandelt alle Wörter in Kleinbuchstaben um und gibt eine Liste der bereinigten Wörter zurück.

```
def bow(sentence, words):
    sentence_words = clean_up_sentence(sentence)
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                bag[i] = 1
    return(numpy.array(bag))
```

Listing 5.5: Erstellung der BOW-Darstellung

Die 'bow' Methode aus 5.5 nimmt einen bereinigten Satz und eine Liste von Wörtern als Eingabe. Sie erstellt eine BOW-Darstellung des Satzes, wobei jedes Element im Bag angibt, ob ein bestimmtes Wort aus der Wortliste im Satz vorkommt oder nicht. Im Einzelnen führt sie folgende Operationen durch: Sie initialisiert ein Array bag mit Nullen, dessen Länge der Länge der Wortliste entspricht; für jedes Wort im bereinigten Satz durchläuft sie die Wortliste und setzt das entsprechende Element in bag auf 1, wenn das Wort aus der Liste mit diesem Wort übereinstimmt, und gibt am Ende das resultierende Array bag zurück.

```
def predict_class(sentence, model):
    p = bow(sentence, words)
    res = model.predict(numpy.array([p]))[0]
    ERROR_THRESHOLD = 0.75
    results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
    return return_list
```

Listing 5.6: Voraussage der Klasse

Wie Listing 5.6 zeigt, nimmt die 'predict_class' Methode einen Satz und ein zuvor trainiertes Modell als Eingabe entgegen. Sie verwendet das Modell, um die Klasse des Satzes (d. h. den Intent dahinter) auf der Grundlage der BOW-Darstellung vorherzusagen. Es gibt eine Liste von

Wörterbüchern zurück, wobei jedes Wörterbuch die vorhergesagte Absicht und die entsprechende Wahrscheinlichkeit enthält. Es verwendet einen 'ERROR_THRESHOLD'-Wert von 0,75, um Vorhersagen mit niedrigen Wahrscheinlichkeiten herauszufiltern.

```
def get_dialog_response(ints, intents_json):
    tag = ints[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if(i['tag']== tag):
            result = random.choice(i['responses'])
            break
    return result
```

Listing 5.7: Antwortgenerierung

Die 'get_dialog_response' Methode (5.7) nimmt die Liste der vorhergesagten Intents und das *intents_json*-Verzeichnis als Eingabe. Sie verwendet den vorhergesagten Intent, um eine Liste möglicher Antworten aus dem *intents_json*-Verzeichnis abzurufen. Anschließend wählt sie eine zufällige Antwort aus der Liste aus und gibt sie zurück.

```
def get_response(msg):
    ints = predict_class(msg, model)
    if not ints:
        ints.append({"intent": "noanswer", "probability": ""})
    nums = [0] * 4
    tag = ints[0]['intent']
    if(tag == "order"):
       nums = parse_line(msg)
    res = get_dialog_response(ints, intents)
    result = ','.join(str(num) for num in nums) + '\n'
    return res, result
```

Listing 5.8: Antwortrückgabe

Wie Listing 5.8 zeigt, nimmt die 'get_response' Methode schließlich eine Nachricht als Eingabe. Sie verwendet die 'predict_class' Methode, um die Absicht der Nachricht vorherzusagen, und verwendet dann die 'get_dialog_response' Methode, um eine Antwort für diese Absicht zu erhalten. Wenn die vorhergesagte Absicht „Bestellung" ist, wird auch die Befehlverarbeiungsmethode aufgerufen, um numerische Werte aus der Nachricht zu extrahieren. Anschließend wird die Antwort zusammen mit den numerischen Werten (falls vorhanden) zurückgegeben.

5.1.3 Implementierung der Befehlverarbeitung

Wie Listing 5.8 zeigt, wird, wenn die Absicht „Bestellung" vorhergesagt wurde, auch die Befehlverarbeiungsmethode 'parse_line' aufgerufen, um einen Prozentwert für jeden Behälter zurückzugeben.

```
def parse_line(line):
    reservoirs = [0] * 4
    line = re.sub(r'[,.!?]', '', line)
    words = line.split()
    for i in range(len(words)):
        if words[i] == 'Behaelter' or words[i] == 'Reservoir':
```

```
7             for x in range(i+1, len(words)):
8                 if words[x].isdigit():
9                     index = int(words[x]) - 1
10                    break
11            for x in range(i-1, 0, -1):
12                if words[x].isdigit():
13                    reservoirs[index] = int(words[x])
14                    break
15    return tuple(reservoirs)
```

Listing 5.9: Befehlverarbeitung

Listing 5.8 zeigt wie die 'parse_line' Methode aufgebaut ist. Die Methode nimmt eine String-Zeile als Eingabe und extrahiert daraus die mit jedem Behälter verbundenen Prozentsätze. Die Methode geht davon aus, dass die Eingabezeichenkette ein bestimmtes Format hat, in dem die Prozentsätze für jeden Behälter in einer bestimmten Reihenfolge stehen (siehe Übergangsdiagramm 4.4 im Konzept-Kapitel 4.7.2).

Die Methode verwendet die in der Python-Bibliothek integrierten Funktionen 'split()' und 'isdigit()', um die erforderlichen Informationen aus der Eingabezeichenfolge zu extrahieren. Die Methode 'split()' wird verwendet, um die Eingabezeichenfolge in einzelne Wörter zu zerlegen, und die Methode 'isdigit()' wird verwendet, um zu prüfen, ob ein Wort eine Ziffer ist. Mithilfe von der 're' Bibliothek werden auch alle Sonderzeichen entfernt, die potenziell vorkommen können: die Methode 're.sub()' aus dem 're' Modul wird verwendet, um alle Vorkommen eines Kommas, eines Ausrufezeichens, eines Punktes oder eines Fragezeichens durch eine leere Zeichenkette '' zu ersetzen. Der reguläre Ausdruck '[,.!?]' innerhalb der 'sub()'-Funktion passt auf jedes einzelne Zeichen, das entweder ein Komma, ein Ausrufezeichen, ein Fragezeichen oder ein Punkt ist.

Im Einzelnen führt sie folgende Operationen durch: Sie initialisiert ein Array mit vier Elementen, wobei jedes Element auf 0 initialisiert ist. Danach wird über die Wortliste mit einer for-Schleife iteriert und geprüft, ob das Wort am aktuellen Index gleich der Zeichenkette „Behaelter“ oder „Reservoir“ ist. Wenn das Wort „Behaelter“ oder „Reservoir“ ist, iteriert die Methode über die Wörter nach dem aktuellen Index mit einer weiteren for-Schleife und prüft mit der Methode isdigit(), ob das Wort am aktuellen Index x eine Ziffer ist. Wenn das der Fall ist, wird diese Zahl als Behälter-Index betrachtet. Danach wird ab dem Wort Behälter mit einer weiteren for-Schleife zurückiteriert, bis noch eine Zahl getroffen wird, die die Prozentzahl eines Getränks für den Behälter mit aktuellem Index darstellt. Diese Zahl wird in der Reservoirs-Liste bei dem Index des aktuellen Behälter gespeichert. Die Methode endet, wenn über alle Wörter der Liste iteriert wurde.

Die Methode gibt ein Tupel von vier ganzen Zahlen zurück, wobei jede ganze Zahl den mit jedem Behälter verbundenen Prozentsatz darstellt. Wenn ein Behälter in der Eingabezeichenkette nicht erwähnt wird, gibt die Methode 0 als Prozentsatz für diesen Behälter zurück.

5.2 Implementierung der Sprachsteuerung

Im Folgenden wird erläutert, wie die Sprachsteuerung für die Getränkemischmaschine implementiert wurde und welche Technologien dafür zum Einsatz kamen. Dabei wird zunächst auf die Spracherkennung d.h., die Umwandlung der Audiosignale (Sprachbefehl des Benutzers) in eine Form, die innerhalb des Quelltextes weiterverarbeitet werden kann, eingegangen (s. Abschnitt 5.2.1). Danach wird die Anbindung des Sprachverarbeitungssystems beschrieben, dessen Implementierung in Abschnitt 5.1 erklärt wird. Abschließend wird die Kommunikation mit der Mischmaschine über den Arduino illustriert (s. Abschnitt 5.2.2).

5.2.1 Spracherkennung

Die Spracherkennung ist der erste Schritt bei der Implementierung einer Sprachsteuerung für die Getränkemischmaschine. Mit Spracherkennung ist die Aufnahme eines Tonsignals über ein Audio-Eingabegerät (Mikrofon) und die Umwandlung der Audiodaten in Text gemeint. Der Quelltext zur Implementierung der Sprachsteuerung erfolgt mit der Programmiersprache *Python*, da hier sehr viele, leicht zu bedienende Bibliotheken zur Spracherkennung, -verarbeitung und KI zur Verfügung stehen.

Für dieses Projekt viel die Wahl auf das Paket *SpeechRecognition*, das die Verwendung verschiedener Spracherkennungsdienste über eine einheitliche Schnittstelle ermöglicht und zu diesem Zweck auch zur Aufnahme und Verarbeitung der Audiosignale verwendet werden kann [Zhang (Uberi) o. D.] Ein großer Vorteil davon ist, dass dadurch ein schneller Wechsel der eingesetzten API erfolgen kann, sollte dies erforderlich sein. Die Verwendung des *SpeechRecognition*-Pakets findet fast ausschließlich über die *Recognizer*-Klasse statt. Um Audiosignale über eine physische Audioquelle (bspw. ein Mikrofon am Computer) aufzunehmen kann die *Microphone*-Klasse verwendet werden, die ebenfalls im Paket enthalten ist. Mit Hilfe eines Objekts vom Typ *Microphone* und der Methode *listen* der *Recognizer*-Klasse können anschließend Audiodaten aufgenommen werden, die in einem Objekt vom Typ *AudioData* gespeichert sind. Die Verwendung von *Recognizer* und *Microphone* sind in Listing 5.10 zu sehen.

```
1 import speech_recognition as sr

3 r = sr.Recognizer()
4 m = sr.Microphone()

6 with m as source:
7     audio = r.listen(source)
```

Listing 5.10: Audioaufnahme mit *SpeechRecognition*

Das *AudioData*-Objekt kann nun verwendet werden um die darin gespeicherten Audiodaten zu erkennen und in Text umzuwandeln. Das *SpeechRecognition*-Paket stellt dafür verschiedene

Möglichkeiten zur Verfügung, wie eingangs erwähnt wurde. Diese sollen im Folgenden kurz beschrieben werden:

- Whisper: Whisper ist ein neuronales Netz das von der Firma *OpenAI* trainiert und als Open-Source-Projekt zur Verfügung gestellt wird [OpenAI o. D.[a],[b]]. Neben der Fähigkeit Sprache in Text zu konvertieren kann es auch eingesetzt werden um Transkripte zu generieren, die gesprochene Sprache automatisch zu erkennen oder in die englische Sprache zu übersetzen. Das Paket *SpeechRecognition* lässt sowohl die Verwendung der von *OpenAI* zur Verfügung gestellten Online-API zu, als auch die lokale Ausführung des Sprachmodells. Aufgrund der Anforderung nach Offline-Funktionalität entfällt die erste Möglichkeit (s. Kapitel 3).

- Sphinx: Das *CMUSphinx* Projekt wird von der Carnegie Mellon University (CMU) unterhalten und stellt eine Reihe von Werkzeugen und Bibliotheken zur Spracherkennung zur Verfügung [Shmyrev o. D.] Darunter fallen bspw. *PocketSphinx*, *SphinxTrain* und *sphinx4*. Bei *PocketSphinx* handelt es sich um eine C-Bibliothek zur Spracherkennung, die auch innerhalb von Python verwendet werden kann. *SphinxTrain* hingegen stellt Ressourcen zum Trainieren eigener Modelle bereit. *sphinx4* ist das Java-Equivalent zu *PocketSphinx*. Somit ist für dieses Projekt nur *PocketSphinx* interessant. Leider unterstützt enthält das Paket standardmäßig nur ein Modell für die englische Sprache. Anderssprachige Modelle und weitere dazugehörige Ressourcen müssen mühsam aus externen Quellen bezogen und eingebunden werden.

- Snowboy: *Snowboy Hotword Detection* ist ein quelloffenes Projekt welches Bibliotheken für verschiedene Programmiersprachen bereitstellt, die dem Erkennen sog. „hot words“ dienen [Kitt-AI 2023]. Ein *hot word* sind ein oder mehrere Wörter die der Aktivierung eines Sprachsteuerungssystems dienen. Bekannte Beispiele hierfür sind „Alexa“ oder „OK Google“. Das System „hört“ im Hintergrund und wartet darauf, dass das *hot word* genannt wird. Anschließend nimmt das System weitere Sätze, die vom Benutzer gesprochen werden, wahr und verarbeitet diese. Ein *hot word* ist notwendig damit das System nicht dauerhaft versucht Sätze zu verarbeiten, die nicht für das System gedacht sind und damit wiederholt Fehler bekommt oder unerwünschtes Verhalten an den Tag legt. Da die Ansprüche des vorliegenden Projekts an die Detektion von *hot words* nicht besonders hoch sind und um die von Snowboy benötigte Rechenzeit einzusparen wird *Snowboy* in diesem Projekt keine Relevanz haben. Auf die konkrete Umsetzung des *hot word* Mechanismus wird später noch eingegangen. Ein weiterer Grund dafür, dass Snowboy nicht zum Einsatz kommen soll ist, dass das Projekt bereits im Jahr 2020 offiziell eingestellt wurde und nur noch von der Community weiter betrieben wird.

- Vosk: Das *Vosk Speech Recognition Toolkit* ist neben *Whisper* und *Sphinx* die dritte und letzte verfügbare offline API für die *SpeechRecognition* eine Schnittstelle bereitstellt [Alpha

Cephei o. D., 2023]. Es wird von der *Alpha Cephei Incorporation* vertrieben. Ähnlich wie bei Whisper sind vortrainierte Modelle in vielen verschiedenen Sprachen bereits vorhanden, müssen allerdings manuell heruntergeladen und dem *SpeechRecognition* Paket zur Verfügung gestellt werden.

- Google: *SpeechRecognition* bietet zwei Möglichkeiten Google-Dienste zur Konvertierung von Sprache zu Text zu verwenden. Die erste Möglichkeit besteht in der Verwendung der *Google Speech API* in der *Google Cloud* [Google o. D.] Die zweite Möglichkeit besteht in der Verwendung der *Google Speech API* ohne die *Google Cloud*. Die Verwendung der Google-Schnittstelle entfällt jedoch für dieses Projekt, da eine aktive Internetverbindung Voraussetzung dafür ist. Außerdem sind die Google-Dienste weder kostenfrei noch open-source.

- Microsoft: Auch hier gibt es zwei verschiedene Möglichkeiten Microsoft-Dienste zur Umwandlung von Sprache zu Text über das *SpeechRecognition*-Paket zu nutzen. Eine davon ist die Verwendung der *Speech to text* Funktion in der *Microsoft Azure Cloud* [Microsoft o. D.] Hierfür sind ein aktiver *Azure*-Account sowie ein gültiger API-Schlüssel notwendig. Die zweite Möglichkeit besteht in der *Microsoft Bing Voice Recognition*. Diese ist jedoch veraltet und wird nicht mehr unterstützt. Auch diese Schnittstelle enfällt jedoch wegen der notwendigen Internetverbindung und Bezahlung.

- IBM: Auch die Firma IBM bietet ihre eigene Schnittstelle zur Spracherkennung mit Hilfe der von IBM entwickelten KI *Watson* an [IBM 2023]. Auch hier gilt: zur Verwendung der API ist sowohl eine aktive Internetverbindung als auch Bezahlung vorgesehen, weshalb diese Schnittstelle für das Projekt nicht in Frage kommt.

- Weitere Schnittstellen sind *Wit.ai*, *Houndify* und *Tensorflow* [SoundHound o. D.; TensorFlow o. D.[a]; Wit.ai o. D.] Da diese Schnittstellen allderdings ebenso eine aktive Internetverbindung voraussetzen und damit ausscheiden soll nicht weiter auf sie eingegangen werden.

Die einzelnen APIs lassen sich jeweils über einen Methodenaufruf der Form *Recognizer.recognize_x* verwenden, wobei das „x“ für den Namen der jeweiligen API steht. Für dieses Projekt viel die Wahl auf die Verwendung der *Whisper*-Bibliothek von *OpenAI*. Dies ist damit zu begründen, dass sie von den offline verwendbaren APIs die mit Abstand am einfachsten zu verwendende ist und gleichzeitig sehr gute Ergebnisse beim Testen damit erzielt wurden. Beispielsweise werden die Modelle für viele verschiedene Sprachen bereits von OpenAI zur Verfügung gestellt und für jedes Modell stehen weitere Ausführungen zur Verfügung die nach den eigenen Ansprüchen und vorhandenen Ressourcen ausgewählt werden können. Die verschiedenen Ausführungen sind nach der „Größe“ des Modells unterteilt in „tiny“, „base“, „small“, „medium“ und „large“ [OpenAI

2023].

```python
import speech_recognition as sr

r = sr.Recognizer()
m = sr.Microphone()

with m as source:
    print("Start listening...")
    audio = r.listen(source)
    text = ""
    try:
        recognized_text = r.recognize_whisper(audio, language="german", model="tiny")
        text = recognized_text
        print("Recognized text: " + text)
    except sr.UnknownValueError:
        print("Whisper could not understand audio.")
    except sr.RequestError as e:
        print("Could not request results from Whisper; {0}".format(e))
```

Listing 5.11: Sprache zu Text mit *OpenAI Whisper*

Listing 5.11 zeigt, wie mit Hilfe der *Whisper*-API die aufgenommenen Audiodaten zu Text verarbeitet und ausgegeben werden können. Die Sprache und die Modellgröße werden bei dem Methodenaufruf *Recognizer.recognize_whisper* angegeben. Sollte das entsprechende Modell noch nicht lokal vorliegen wird dieses bei der ersten Ausführung automatisch installiert, was einen hohen Grad an Benutzerfreundlichkeit seitens der API bedeutet. Natürlich ist es unzureichend einmal die Sprache des Benutzers aufzunehmen und in Text zu konvertieren. Insgesamt muss das Programm während des Betriebs der Getränkemischmaschine in einer Dauerschleife ausgeführt werden und der Benutzer muss die Möglichkeit haben mehrere Befehle nacheinander über die Sprachsteuerung auszuführen. Listing 5.12 zeigt wie dies konkret erreicht wurde.

```python
import speech_recognition as sr

r = sr.Recognizer()
m = sr.Microphone()
with m as source:
    r.adjust_for_ambient_noise(source)

while True:
    print("Start listening...")
    with m as source:
        audio = r.listen(source)
    text = ""
    try:
        recognized_text = r.recognize_whisper(audio, language="german", model="tiny")
        text = recognized_text
        print("Recognized text: " + text)
    except sr.UnknownValueError:
        print("Whisper could not understand audio.")
    except sr.RequestError as e:
        print("Could not request results from Whisper; {0}".format(e))

    if text.lower().strip().startswith("mischmaschine"):
        print("Recognized hot word. Will output answer and command.")
```

Listing 5.12: Hot-Word-Erkennung

Der Code der für die Konvertierung in Sprache zu Text zuständig wird nun innerhalb einer While-True-Schleife ausgeführt. Um zu verhindern, dass die Mischmaschine Befehle ver-

arbeitet, die keine sind, soll das *hot word* „Mischmaschine“ zur Aktivierung genutzt werden. Deshalb wird in Zeile 22 nun geprüft, ob der erkannte Text mit diesem Wort beginnt und nur dann werden weitere Schritte zur Befehlsverarbeitung eingeleitet (mehr dazu in Abschnitt 5.2.2). Da davon auszugehen ist, dass die Mischmaschine an Orten eingesetzt wird, die vielen Umgebungsgeräuschen ausgesetzt sind wird in Zeile sechs außerdem die Methode *speech_recognition.Recognizer().adjust_for_ambient_noise(source)* aufgerufen, die dafür sorgt, dass die Empfindlichkeit des Mikrofons automatisch an die Lautstärke der Umgebungsgeräusche angepasst wird.

Herausforderungen bei der Implementierung der Spracherkennung

Die Verwendung von *Whisper* setzt die Installation von *PyTorch* voraus - einem quelloffenen ML-Framework [PyTorch o. D.] Die Installation von *PyTorch* kann auf zwei unterschiedlichen Wegen erfolgen:

1. Unter der Benutzung vorkompilierter Distributionen mit Hilfe eines Paketverwaltungssystems wie etwa *pip* [PPA o. D.]

2. Durch das eigenständige Bauen des Quelltextes auf dem lokalen Rechner.

Da es sich bei *PyTorch* um ein überaus großes und komplexes Projekt handelt ist es empfehlenswert die erste Methode anzuwenden, da der Build-Vorgang bei Methode zwei sehr viele Ressourcen benötigt.

Nach der Verwendung von *pip* zur Installation von *PyTorch* auf dem für das Projekt verwendeten *Raspberry Pi 4B* kam es jedoch zu einem unerwarteten Fehler. Das Importieren der *PyTorch*-Bibliothek ließ das Programm sofort beenden und die Nachricht „Illegal Instruction“ auf dem Bildschirm ausgeben. Die Fehlermeldung lässt darauf schließen, dass die Bibliothek einen für den Prozessortyp unbekannten Befehl ausüben will. Drei Lösungsansätze wurden daraufhin konzipiert:

1. Das Programm sollte mit einer anderen Python-Version ausgeführt werden.

2. Das vorkompilierte Paket, das automatisch von *pip* ausgewählt wurde scheint nicht auf die vorhandene Prozessorarchitektur zu passen und führt unbekannte Maschinenbefehle aus. Deshalb sollte die *PyTorch*-Bibliothek selbstständig, lokal kompiliert werden.

3. Ein anderes vorkompiliertes Paket herunterladen und installieren, das auf die vorhandene Prozessorarchitektur angepasst und optimiert ist.

4. Sollten alle zuvor genannten Versuche scheitern muss *Whisper* durch eine andere Bibliothek ersetzt werden, die keine Abhängigkeit zu *PyTorch* aufweist.

Lösungsansatz eins erwies sich als nicht effektiv. Das Programm verursachte immernoch den selben Fehler. Lösungsansatz zwei scheiterte, da der Build-Prozess mehr als den zur Verfügung stehenden RAM des Raspberry Pi aufbrauchte. Sogar mit vier Gigabyte (GB) zusätzlich konfiguriertem Swap-Speicher wurde der Build-Prozess nach einiger Zeit vom Betriebssystem beendet aufgrund des hohen RAM-Verbrauchs. Lösungsansatz drei funktionierte schließlich. Mit Hilfe des vorkompilierten Pakets aus [Q-ENGINEERING o. D.] konnte *PyTorch* installiert und erfolgreich importiert und verwendet werden.

Ein weiteres Problem trat bei der Verwendung des Mikrofons über die *SpeechRecognition*-Bibliothek auf. Bei den Aufrufen *speech_recognition.Microphone()* und *speech_recognition.Recognizer().listen(source)* kommt es auf dem Raspberry Pi zu unerwarteten Fehlermeldungen, die in Abbildung 5.2 zu sehen sind.

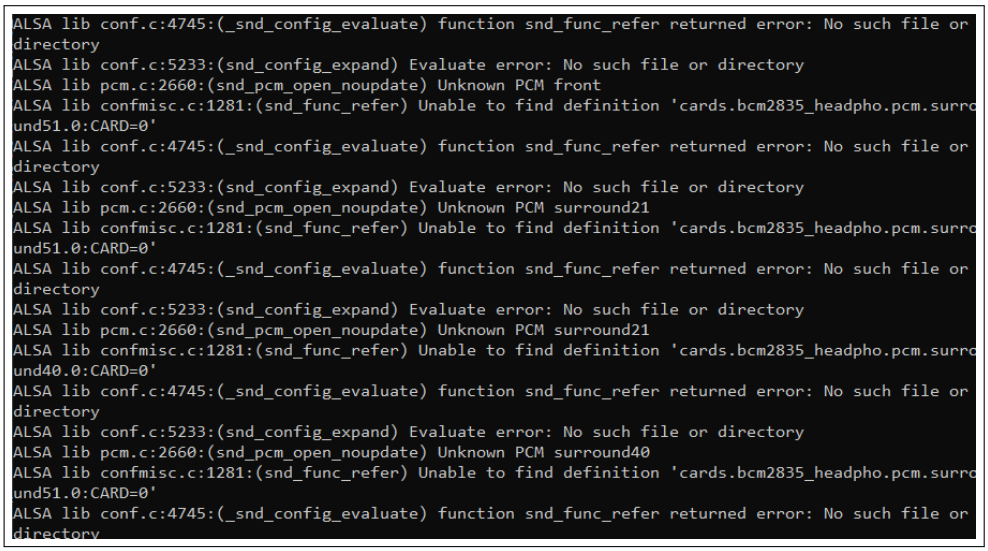

Abbildung 5.2: Fehlermeldungen beim Zugriff auf das Mikrofon

Zwar kann das Programm trotz der Meldungen fehlerfrei ausgeführt werden, jedoch stören die Meldungen bspw. beim Debuggen und irritieren evtl. Programmierer, die zu einem späteren Zeitpunkt einen Blick auf das Projekt werfen. Um die Fehlermeldungen zu umgehen wurde deshalb im finalen Quelltext die Fehlerbehandlungsfunktion der *libasound*-Bibliothek ausgetauscht wie in Listing 5.13 zu sehen ist.

```
1 from ctypes import *
2 import speech_recognition as sr

4 ERROR_HANDLER_FUNC = CFUNCTYPE(None, c_char_p, c_int, c_char_p, c_int, c_char_p)
5 def py_error_handler(filename, line, function, err, fmt):
6     pass
7 c_error_handler = ERROR_HANDLER_FUNC(py_error_handler)

9 asound = cdll.LoadLibrary("libasound.so")
```

```
10 asound.snd_lib_error_set_handler(c_error_handler)

12 r = sr.Recognizer()
13 m = sr.Microphone()
14 with m as source:
15     r.adjust_for_ambient_noise(source)

17 while True:
18     asound.snd_lib_error_set_handler(c_error_handler)
19     print("Start listening...")
20     with m as source:
21         audio = r.listen(source)
22     asound.snd_lib_error_set_handler(None)
23     text = ""
24     try:
25         recognized_text = r.recognize_whisper(audio, language="german", model="tiny")
26         text = recognized_text
27         print("Recognized text: " + text)
28     except sr.UnknownValueError:
29         print("Whisper could not understand audio.")
30     except sr.RequestError as e:
31         print("Could not request results from Whisper; {0}".format(e))

33     if text.lower().rstrip().startswith("mischmaschine"):
34         print("Recognized hot word. Will output answer and command.")
```

Listing 5.13: Fehlerbehandlung

Für einen Vorher-/Nachher-Vergleich sei auf Listing 5.11 verwiesen. Die Zeilen 4 bis 10 dienen der Definition einer C-Funktion zur Fehlerbehandlung, die bei ihrer Ausführung die auftretende Meldung ignoriert. In Zeile 18 wird die Standard Fehlerfunktion durch diese ersetzt und in Zeile 22 wieder zurückgeändert.

5.2.2 Befehlsverarbeitung in der Mischmaschine

Nachdem die Sprache des Benutzers erkannt, in Text konvertiert und durch das Sprachmodell interpretiert wurde muss der zurückerhaltene Befehl vom Raspberry Pi and den Arduino in der Getränkemischmaschine gesendet werden. Zur Umsetzung der Kommunikation zwischen Raspberry Pi und Arduino kommt die *PySerial*-Bibliothek zum Einsatz [PySerial o. D.] Listing 5.14 zeigt, wie die Bibliothek zum Versenden von Nachrichten an ein Gerät über die serielle Schnittstelle erfolgen kann.

```
1 import serial
2 import time

4 ser = serial.Serial("/dev/ttyACM0", 9600, timeout=5)
5 ser.reset_input_buffer()
6 time.sleep(5)
7 count = 0
8 while True:
9     ser.write("Message {0}\n".format(count).encode("utf-8"))
10    line = ser.readline().decode("utf-8").rstrip()
11    print(line)
12    count += 1
13    time.sleep(1)
```

Listing 5.14: Serielle Kommunikation in Python

Das Programm öffnet in Zeile 4 die serielle Schnittstelle zu dem Gerät mit dem Namen „ttyACM0“, mit einer Baud-Rate von 9600 und einem Timeout von fünf Sekunden. Der Gerätename ist je nach

Gerät und Betriebssystem unterschiedlich und muss zuvor mit den verfügbaren Betriebssystemmitteln bestimmt werden. Die Baud-Rate gibt an, mit welcher Geschwindigkeit Symbole über die Schnittstelle gesendet werden. Die Baud-Rate muss sowohl beim Sender als auch beim Empfänger gleich sein, da es sonst zu Kommunikationsproblemen kommt. Der Timeout steuert wie lange bei einer Leseoperation gewartet werden soll, bis alle zu lesenden Bytes im Puffer vorhanden sind.

In der darauffolgenden Zeile wird der Eingabepuffer geleert, um zu verhindern, dass unvollständige Daten an den Empfänger gesendet werden, die beim letzten Programmdurchlauf im Puffer verblieben sein könnten. Da der Verbindungsaufbau mit dem Gerät einige Zeit dauern kann wird die Programmausührung in Zeile 6 zunächst um fünf Sekunden verzögert. Das nachfolgende Programm sendet in einer While-True-Schleife mehrere Nachrichten über die serielle Schnittstelle, die über einen Zähler voneinander zu unterscheiden sind. Anschließend wird in Zeile zehn eine Nachricht zurück erwartet die anschließend auf der Kommandozeile ausgegeben wird.

In Listing 5.15 ist der Quelltext auf Empfängerseite zu sehen, der dafür zuständig ist die in Listing 5.14 generierten Nachrichten zu empfangen und Daten zurückzusenden.

```
1  void setup() {
2    Serial.begin(9600);
3  }

5  void loop() {
6    if (Serial.available() > 0) {
7      String data = Serial.readStringUntil('\n');
8      Serial.println(data);
9      delay(1000);
10   }
```

Listing 5.15: Serielle Kommunikation in Python

Das Programm ist in der Arduino-Programmiersprache verfasst und besteht, wie vorgegeben, aus einer *setup* und einer *loop* Methode. Die *setup* Methode wird einmalig beim Start des Programms, wenn der Arduino mit Strom versorgt wird, ausgeführt wohingegen die *loop* Methode dauerhaft während der Arduino mit Strom versorgt ist wiederholt ausgeführt wird. In der *setup*-Methode wird die serielle Schnittstelle mit einer Baud-Rate von 9600 geöffnet. Innerhalb der *loop*-Methode wird geprüft, ob Bytes in den Eingabepuffer geschrieben wurden. Ist dies der Fall wird der If-Block betreten und die in den Puffer geschriebenen Bytes werden bis zu einem vorgegebenen Terminalsymbol (in diesem Fall ein Zeilenumbruch) eingelesen und in ein String-Objekt konvertiert. Mit dem Aufruf *Serial.println(data)* wird der empfangene String über die serielle Schnittstelle zurück an den ursprünglichen Sender übertragen. Dabei handelt es sich um den String, der in dem Programm aus Listing 5.14 in Zeile 10 erwartet wird.

Die eben an einem einfachen Beispiel beschriebene Vorgehensweise/Implementierung kann eins zu eins auf die Problemstellung des vorliegenden Projekts übertragen werden. Statt der

generischen Nachricht wird das Kommando, das von dem Sprachmodell zurückgegeben wird übertragen. Auf Seiten des Arduinos muss das Kommando interpretiert und die entsprechenden Aktionen innerhalb der Mischmaschine ausgelöst werden. Listing 5.16 zeigt wie auf seiten des Raspberry Pi die eben beschriebenen Techniken in das Hauptprogramm aus Listing 5.13 integriert werden können. Die Definition und das Setzen der C-Fehlerbehandlungsfunktion wurde aus Gründen der Übersichtlichkeit weggelassen.

```
1  import speech_recognition as sr
2  import serial
3  import time

5  def language_model_stub(s):
6      return s, "50,50,0,0\n"

8  r = sr.Recognizer()
9  m = sr.Microphone()
10 with m as source:
11     r.adjust_for_ambient_noise(source)

13 ser = serial.Serial("/dev/ttyACM0", 9600, timeout=5)
14 ser.reset_input_buffer()
15 time.sleep(5)

17 while True:
18     print("Start listening...")
19     with m as source:
20         audio = r.listen(source)
21     text = ""
22     try:
23         recognized_text = r.recognize_whisper(audio, language="german", model="tiny")
24         text = recognized_text
25         print("Recognized text: " + text)
26     except sr.UnknownValueError:
27         print("Whisper could not understand audio.")
28     except sr.RequestError as e:
29         print("Could not request results from Whisper; {0}".format(e))

31     if text.lower().rstrip().startswith("mischmaschine"):
32         print("Recognized hot word. Will output answer and command.")
33         answer, cmd = language_model_stub(text)
34         ser.write(cmd.encode("utf-8"))
35         serial_answer = ser.readline().decode("utf-8").rstrip()
36         print(serial_answer)
```

Listing 5.16: Serielle Kommunikation für die Mischmaschine

In den Zeilen 5 und 6 wird eine Methode definiert, die das Sprachmodell, welches die Sprache des Benutzers interpretiert und einen Maschinenbefehl und eine Antwort zurückgibt, nachahmen soll. Das Kommando besteht aus vier natürlichen Zahlen, die die Prozentwerte der jeweiligen Behälter in der Mischmaschine darstellen. Der String „50,50,0,0“ ist also so zu interpretieren, dass das Mischungsverhältnis zu 50 Prozent aus Behälter eins und zu 50 Prozent aus Behälter zwei bestehen soll. In Zeile 34 wird dieses Kommando an den Arduino, welcher die Mischmaschine steuert, übertragen. Listing 5.17 zeigt einen Ausschnitt des abgeänderten Code auf Seiten des Arduinos. Der gesamte Quelltext ist in Anhang A zu finden.

```
1 int getValueAt(String s, int vpos) {
2   int pos = 0;
3   int valueCount = 0;
```

```
4   for (int i = 0; i < s.length(); i++) {
5     if (s.charAt(i) == ',' || i == s.length() - 1) {
6       String value = "";
7       int bound = (i == s.length() - 1 ? i + 1 : i);
8       for (int j = pos; j < bound; j++) {
9         value.concat(s.charAt(j));
10      }
11      if (valueCount == vpos) {
12        return value.toInt();
13      }
14      pos = i + 1;
15      valueCount++;
16    }
17  }
18  return -1;
19 }

21 void setup() {
22   Serial.begin(9600);
23 }

25 void loop() {
26     if (Serial.available() > 0) {
27         String data = Serial.readStringUntil('\n');  // e.g. "50,50,0,0"
28         Serial.println("Received " + data);
29         analogWrite(pumpe1, (getValueAt(data, 0) * 0.5 + 50) * 2.55);
30         analogWrite(pumpe2, (getValueAt(data, 1) * 0.5 + 50) * 2.55);
31         analogWrite(pumpe3, (getValueAt(data, 2) * 0.5 + 50) * 2.55);
32         analogWrite(pumpe4, (getValueAt(data, 3) * 0.5 + 50) * 2.55);
33         delay(3000);  // let the "button" be "pressed" for three seconds
34     } else {
35         analogWrite(pumpe1, 0);
36         analogWrite(pumpe2, 0);
37         analogWrite(pumpe3, 0);
38         analogWrite(pumpe4, 0);
39         analogWrite(pumpe5, 0);
40     }
41 }
```

Listing 5.17: Interpretation der Kommandos

Mit der Methode *getValueAt(String, int): int* wird die Prozentangabe an der gewünschten Stelle im Kommandostring ausgelesen. In der Hauptprogrammschleife werden die Pumpen eins bis vier mit ihren entsprechenden Werten angesprochen. Das Delay in der letzten Zeile des If-Blocks sorgt dafür, dass die Pumpen nicht sofort wieder in der nächsten Schleifeniteration ausgeschaltet werden.

Herausforderungen bei der Implementierung der Befehlsverarbeitung

Eine Herasuforderung bestand in der Steuerung der seriellen Kommunikation von Raspberry Pi und Touch-Display mit dem Arduino. Zunächst wurde versehentlich die selbe serielle Schnittstelle sowohl für das *Nextion*-Display als auch den Raspberry Pi verwendet. Dies führte dazu, dass der Empfänger (in diesem Fall der Arduino, welcher die Mischmaschine steuert) korrumpierte Daten erhielt. Die korrumpierten Daten waren für den Arduino nicht mehr lesbar, sodass die Mischmaschine auf keine Benutezreingaben mehr reagieren konnte. Die anfängliche Verwirrung entstand dadurch, dass das *Serial*-Objekt sich auf die Standardschnittstelle bezieht, welche sowohl den USB-Hub, als auch die TX0/RX0-Pins mit einschließt, an die das Display ursprünglich angeschlossen war. Die verfügbaren Pins sind in Abbildung 5.3 zu sehen [Arduino o. D.[b]].

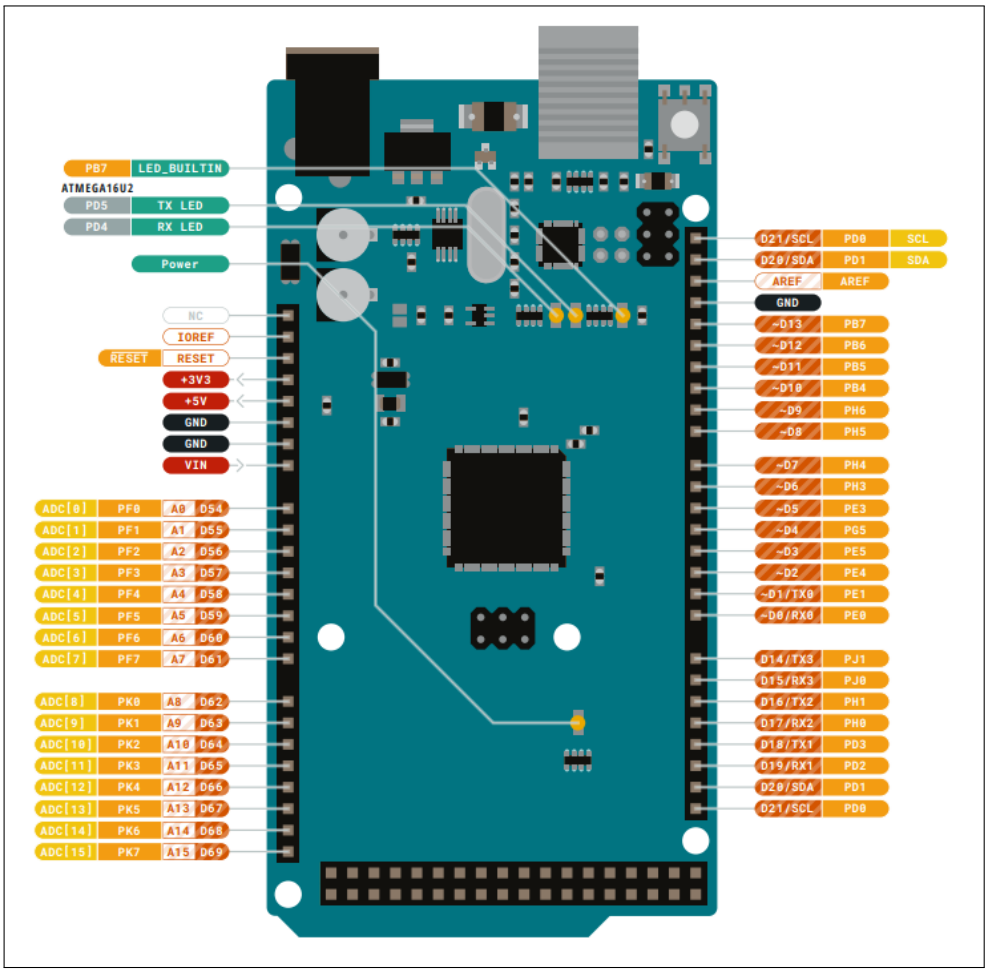

Abbildung 5.3: Pinout für Arduino Mega

Nachdem die Ursache des Problems gefunden wurde konnte nach einer Lösung gesucht werden. Die Lösung bestand darin, das *Nextion*-Display über eine andere serielle Schnittstelle mit dem Ardunino kommunizieren zu lassen. Zu diesem Zweck wurde das *Serial2*-Objekt gewählt welches sich auf die beiden Pins 16 und 17 bezieht. Anhang A, Zeile 264 zeigt, wie über die *Serial2*-Schnittstelle die Kommunikation mit dem *Nextion*-Display initialisiert wird, wohingegen der Raspberry Pi weiterhin den USB-Hub und damit die Standardschnittstelle benutzt. So konnte die gleichzeitige Kommunikation zweier Geräte mit dem Arduino gewährleistet werden.

Eine weitere Herausforderung bestand in einem Programmierfehler, der dazu führte, dass die Mischmaschine nicht wie gewünscht über die Sprachsteuerung bedient werden konnte. Der Fehler war auf eine falsche Verwendung der *PySerial*-Bibliothek zurückzuführen. In einer früheren Version des Programms aus Listing 5.16 wurde der Funktionsaufruf *serial.Serial(name, rate, timeout).reset_input_buffer()*, der in Zeile 13 zu sehen ist, in jeder Schleifeniteration der While-True-Schleife ausgeführt. Die Idee hierbei war, dass vor jeder neuen Übertragung der

Eingabepuffer geleert werden muss, da evtl. korrupte Daten darin vorliegen. Dabei handelt es sich allerdings um einen Irrtum, da in jeder Schleifeniteration alle Bytes zum Arduino gesendet und alle Bytes vom Arduino zurückerhalten werden. Der einzige Fall in dem es möglich ist, dass korrupte Daten in dem Eingabepuffer zurückgeblieben sind ist, wenn das Programm unerwartet stoppt, bspw. wenn der Benutzer einen Stopp veranlasst oder die Stromversorgung abbricht. Deshalb darf der Eingabepuffer nur am Anfang des Programms einmal geleert werden und nicht in jeder Schleifeniteration, da sonst die Nutzdaten, die tatsächlich übertragen werden sollen, aus dem Puffer entfernt werden.

5.2.3 Sprachausgabe

Die Sprachausgabe beschäftigt sich damit, die Antwort, die von dem Sprachmodell in textform ausgegeben wird, durch Lautsprecher in der Mischmaschine auszugeben und sie somit Teilnehmer einer Konversation mit dem Benutzer werden zu lassen. Um dies zu bewerkstelligen kommt die *pyttsx3*-Bibliothek zum Einsatz [PYTTSX3 o. D.] Listing 5.18 zeigt die Benutzung der Bibliothek anhand eines einfachen Beispiels.

```
1 import pyttsx3 as tts

3 engine = tts.init()
4 voices = engine.getProperty("voices")
5 engine.setProperty("voice", voices[1].id)

7 engine.say("Hallo Welt!")
8 engine.runAndWait()
9 engine.stop()
```

Listing 5.18: Text-to-Speech mit *pyttsx3*

Zunächst wird ein Objekt vom Typ *pyttsx3.Engine* durch den Aufruf *pyttsx3.init()* erzeugt. Dabei werden automatisch die vorhandenen Treiber auf dem Gerät gesucht, die für die Sprachausgabe verwendet werden können. Die am häufigsten vorkommenden Treiber sind *SAPI5* für Windows, *NSSpeechSynthesizer* für Mac OS X und *eSpeak* für die sonstigen Betriebssysteme wie etwa Raspberry Pi OS. Die Treiber stellen verschiedene „Stimmen“ zur Verfügung, die in der Property *voices* gespeichert werden und wie in Zeile 4 zu sehen ist mit einem Aufruf von *Engine.getProperty(„voices“)* abgerufen werden können. In diesem Beispiel wird die von *pyttsx3* zu verwendende Stimme auf die Stimme an Index eins gesetzt. Wenn eine bestimmte Stimme gewünscht ist, dann müssen die von *pyttsx3* gefundenen Stimmen auf ihre Attribute/Eigenschaften hin untersucht werden. Ein Objekt vom Typ *pyttsx3.voice.Voice* speichert das Alter, Geschlecht, die unterstützten Sprachen und den Namen als auch eine eindeutige Kennung der Stimme. Somit kann entschieden werden, welche Stimme am besten für den eigenen Anwendungsfall geeignet ist. Alternativ kann ein eigener Treiber/Synthesizer implementiert werden. Für die schlussendliche Implementierung der Sprachausgabe innerhalb des Hauptprogramms sei auf Anhang B verwiesen.

5.2.4 Anbindung des Sprachmodells an die Mischmaschine

Die Anbindung des Sprachmodells and die Mischmaschine war schlussendlich ohne großen Aufwand möglich. Es musste lediglich der Platzhalter aus Listing 5.16 in den Zeilen 5 und 6 durch einen Methodenaufruf der tatsächlichen Implementierung in Zeile 48 ersetzt werden, wie im Anhang B zu sehen ist. Für den vollständigen Code des Sprachmodells sei auf C verwiesen.

Kapitel 6

Fazit und Ausblick

TODO

Literatur

AIML Foundation [o. D.] URL: http://www.aiml.foundation/doc.html [besucht am 12. 03. 2023] [siehe S. 31].

Alammar, Jay [o. D.] *Sequence-to-Sequence Models for Chatbots* [siehe S. 36].

Alpha Cephei [o. D.] *VOSK Offline Speech Recognition API*. en. URL: https://alphacephei.com/vosk/ [besucht am 03. 04. 2023] [siehe S. 63].

— [Apr. 2023]. *Vosk Speech Recognition Toolkit*. original-date: 2019-09-03T17:48:42Z. URL: https://github.com/alphacep/vosk-api [besucht am 03. 04. 2023] [siehe S. 64].

amazon [o. D.] *Amazon Lex: How It Works - Amazon Lex*. URL: https://docs.aws.amazon.com/lex/latest/dg/how-it-works.html [besucht am 12. 03. 2023] [siehe S. 34].

Arduino [o. D.[a]]. *Arduino Education*. en. URL: https://www.arduino.cc/education [besucht am 10. 04. 2023] [siehe S. 5].

— [o. D.[b]]. *Arduino Mega 2560 Rev3*. en. URL: https://store.arduino.cc/products/arduino-mega-2560-rev3 [besucht am 07. 04. 2023] [siehe S. 71].

— [o. D.[c]]. *Arduino Reference - Arduino Reference*. URL: https://www.arduino.cc/reference/en/ [besucht am 10. 04. 2023] [siehe S. 7].

Chawla, Sumit [o. D.] *Building Chatbots with Google Dialogflow: Create chatbots with Dialogflow's natural language processing and machine learning capabilities* [siehe S. 34].

Deerwester, Scott u. a. [1990]. »Indexing by latent semantic analysis«. en. In: *Journal of the American Society for Information Science* 41.6. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291097-4571%28199009%2941%3A6%3C391%3A%3AAID-ASI1%3E3.0.CO%3B2-9, S. 391–407. ISSN: 1097-4571. DOI: 10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-4571%28199009%2941%3A6%3C391%3A%3AAID-ASI1%3E3.0.CO%3B2-9 [besucht am 06. 04. 2023] [siehe S. 26].

Diana, Perez-Marin und Pascual-Nieto Ismael [Juni 2011]. *Conversational Agents and Natural Language Interaction: Techniques and Effective Practices: Techniques and Effective Practices*. en. Google-Books-ID: 2nUcqtbCOBcC. IGI Global. ISBN: 978-1-60960-618-3 [siehe S. 31, 33].

Eckart, Carl und Gale Young [Sep. 1936]. »The approximation of one matrix by another of lower rank«. en. In: *Psychometrika* 1.3, S. 211–218. ISSN: 1860-0980. DOI: 10.1007/BF02288367. URL: https://doi.org/10.1007/BF02288367 [besucht am 06. 04. 2023] [siehe S. 26, 27].

Goldberg, Yoav [Apr. 2017]. *Neural Network Methods in Natural Language Processing*. Englisch. Hrsg. von Graeme Hirst. San Rafael, Calif.: Morgan & Claypool Publishers. ISBN: 978-1-62705-298-6 [siehe S. 14].

Google [o. D.] *Speech-to-Text: Automatic Speech Recognition*. en. URL: https://cloud.google.com/speech-to-text [besucht am 26. 03. 2023] [siehe S. 40, 64].

Götz, Felix, Moritz Höckele und Florian Lobert [2022]. *Mischmaschine*. Projektarbeit Software des Studiengangs Mechatronik an der DHBW Karlsruhe [siehe S. 1, 3, 4].

Herczeg, Michael [März 2018]. »9. Zeitverhalten interaktiver Systeme«. de. In: *9. Zeitverhalten interaktiver Systeme*. De Gruyter Oldenbourg, S. 173–182. ISBN: 978-3-11-044686-9. DOI: 10.1515/9783110446869-187. URL: https://www.degruyter.com/document/doi/10.1515/9783110446869-187/html [besucht am 26. 03. 2023] [siehe S. 39].

IBM [März 2023]. *IBM Watson Speech to Text - Overview*. en-us. URL: https://www.ibm.com/cloud/watson-speech-to-text [besucht am 02. 04. 2023] [siehe S. 64].

Isbell, Charles und Paul Viola [1998]. »Restructuring Sparse High Dimensional Data for Effective Retrieval«. In: *Advances in Neural Information Processing Systems*. Bd. 11. MIT Press. URL: https://proceedings.neurips.cc/paper_files/paper/1998/hash/87ec2f451208df97228105657edb717f-Abstract.html [besucht am 06. 04. 2023] [siehe S. 26].

Jansen, Jonas [Feb. 2017]. »Digitalisierung: 8,4 Milliarden vernetzte Geräte im Internet der Dinge«. de. In: *FAZ.NET*. ISSN: 0174-4909. URL: https://www.faz.net/aktuell/wirtschaft/netzwirtschaft/digitalisierung-8-4-milliarden-vernetzte-geraete-im-internet-der-dinge-14865654.html [besucht am 15. 03. 2023] [siehe S. 1].

Jurafsky, Dan u. a. [2009]. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition / Daniel Jurafsky (Stanford University), James H. Martin (University of Colorado at Boulder)*. eng. ISBN: 978-0-13-504196-3 [siehe S. 8, 25].

Kitt-AI [Apr. 2023]. *Snowboy Hotword Detection*. original-date: 2016-05-10T00:54:30Z. URL: https://github.com/Kitt-AI/snowboy [besucht am 03. 04. 2023] [siehe S. 63].

Koech, Kiprono Elijah [Juli 2022]. *Cross-Entropy Loss Function*. en. URL: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e [besucht am 06. 04. 2023] [siehe S. 38].

Lane, Hobson, Cole Howard und Hannes Max Hapke [2019]. *Natural language processing in action: understanding, analyzing, and generating text with Python / Hobson Lane, Cole Howard, Hannes Max Hapke*. eng. ISBN: 978-1-61729-463-1 [siehe S. 9, 30].

Li, Jiwei u. a. [Sep. 2017]. *Adversarial Learning for Neural Dialogue Generation*. arXiv:1701.06547 [cs]. DOI: 10.48550/arXiv.1701.06547. URL: http://arxiv.org/abs/1701.06547 [besucht am 12. 03. 2023] [siehe S. 35].

Ltd, Raspberry Pi [o. D.] *Raspberry Pi 4 Model B specifications*. en-GB. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/ [besucht am 24. 03. 2023] [siehe S. 41].

Marwedel, Peter [2021]. *Eingebettete Systeme: Grundlagen Eingebetteter Systeme in Cyber-Physikalischen Systemen*. de. Wiesbaden: Springer Fachmedien. ISBN: 978-3-658-33436-9 978-3-658-33437-6. DOI: 10.1007/978-3-658-33437-6. URL: https://link.springer.com/10.1007/978-3-658-33437-6 [besucht am 15. 03. 2023] [siehe S. 1].

Microsoft [o. D.] *Speech to Text – Audio to Text Translation | Microsoft Azure*. en-US. URL: https://azure.microsoft.com/en-us/products/cognitive-services/speech-to-text [besucht am 02. 04. 2023] [siehe S. 64].

Nupur Sharma, Anupriya [o. D.] *Chatbot Development using Machine Learning Techniques* [siehe S. 35].

OpenAI [o. D.[a]]. *About*. en-US. URL: https://openai.com/about [besucht am 01. 04. 2023] [siehe S. 63].

— [o. D.[b]]. *Introducing Whisper*. en-US. URL: https://openai.com/research/whisper [besucht am 01. 04. 2023] [siehe S. 63].

— [Apr. 2023]. *Whisper*. original-date: 2022-09-16T20:02:54Z. URL: https://github.com/openai/whisper [besucht am 02. 04. 2023] [siehe S. 64].

PPA, Python Packaging Authority [o. D.] *pip: The PyPA recommended tool for installing Python packages*. URL: https://pip.pypa.io/ [besucht am 03. 04. 2023] [siehe S. 66].

PySerial [o. D.] *Welcome to pySerial's documentation — pySerial 3.0 documentation*. URL: https://pythonhosted.org/pyserial/ [besucht am 04. 04. 2023] [siehe S. 68].

PyTorch [o. D.] *PyTorch*. en. URL: https://www.pytorch.org [besucht am 03. 04. 2023] [siehe S. 66].

pyttsx3 [o. D.] *pyttsx3 - Text-to-speech x-platform — pyttsx3 2.6 documentation*. URL: https://pyttsx3.readthedocs.io/en/latest/index.html [besucht am 07. 04. 2023] [siehe S. 73].

Q-engineering [o. D.] *Install PyTorch on Raspberry Pi 4 - Q-engineering*. en-GB. URL: https://qengineering.eu/install-pytorch-on-raspberry-pi-4.html [besucht am 04. 04. 2023] [siehe S. 67].

RYGL, Jan u. a. [Juni 2017]. *Semantic Vector Encoding and Similarity Search Using Fulltext Search Engines*. arXiv:1706.00957 [cs]. URL: http://arxiv.org/abs/1706.00957 [besucht am 06. 04. 2023] [siehe S. 24].

SCIKIT-LEARN [o. D.] *scikit-learn documentation — DevDocs*. en. URL: https://devdocs.io/scikit_learn/ [besucht am 12. 03. 2023] [siehe S. 33].

SHMYREV, Nickolay [o. D.] *About CMUSphinx*. URL: http://cmusphinx.github.io/wiki/about/ [besucht am 02. 04. 2023] [siehe S. 63].

SOUNDHOUND [o. D.] *Home*. en-US. URL: https://www.soundhound.com/ [besucht am 02. 04. 2023] [siehe S. 64].

TENSORFLOW [o. D.[a]]. *TensorFlow*. en. URL: https://www.tensorflow.org/ [besucht am 02. 04. 2023] [siehe S. 64].

— [o. D.[b]]. *TensorFlow - Sequence-to-Sequence Models*. URL: https://chromium.googlesource.com/external/github.com/tensorflow/tensorflow/+/r0.7/tensorflow/g3doc/tutorials/seq2seq/index.md [besucht am 12. 03. 2023] [siehe S. 36].

WIRING [o. D.] *Wiring*. URL: http://wiring.org.co/ [besucht am 10. 04. 2023] [siehe S. 6].

WIT.AI [o. D.] *Wit.ai*. URL: https://wit.ai/ [besucht am 02. 04. 2023] [siehe S. 64].

WONG, S. K. M., Wojciech ZIARKO und Patrick C. N. WONG [Juni 1985]. »Generalized vector spaces model in information retrieval«. In: *Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval*. SIGIR '85. New York, NY, USA: Association for Computing Machinery, S. 18–25. ISBN: 978-0-89791-159-7. DOI: 10.1145/253495.253506. URL: https://dl.acm.org/doi/10.1145/253495.253506 [besucht am 06. 04. 2023] [siehe S. 26].

WOUDENBERG, Aswin van [2014]. »A Chatbot Dialogue Manager-Chatbots and Dialogue Systems: A Hybrid Approach«. Magisterarb. Open Universiteit Nederland [siehe S. 30].

ZHANG (UBERI), Anthony [o. D.] *SpeechRecognition: Library for performing speech recognition, with support for several engines and APIs, online and offline.* URL: https://github.com/Uberi/speech_recognition#readme [besucht am 30. 03. 2023] [siehe S. 62].

Anhang A

Arduino-Programmcode für die Mischmaschine

```
/*Programm fuer Softwareprojekt "Getraenkemischmaschine"
  DHBW Karlsruhe TMT20B1 (Goetz, Hoeckele, Lobert)

   Pinbelegung am Arduino:
  +5V = 5V
  TX  = pin 0 (RX)
  RX  = pin 1 (TX)
  GND = GND

  Pumpe1 = pin 11
  Pumpe2 = pin 10
  Pumpe3 = pin 9
  Pumpe4 = pin 3
  Pumpe5 = pin 5

  Wasserstandssensor1 = A1
  Wasserstandssensor2 = A2
  Wasserstandssensor3 = A3
  Wasserstandssensor4 = A4
  Wasserstandssensor5 = A5

  Schalter = pin 13

*/
#include <Nextion.h>
//Variablen
int CurrentPage = 0;  //Aktuelle Seite des Displays
int S;                //Welcher pwm wert wird veraendert
int pwm1 = 0;         //pwm Werte
int pwm2 = 0;
int pwm3 = 0;
int pwm4 = 0;
int pwm5 = 0;

int Schalter = 13;  //Fuellen Druckknopf

int pumpe1 = 11;  //Pumpe 1
int pumpe2 = 10;  //Pumpe 2
int pumpe3 = 9;   //Pumpe 3
int pumpe4 = 3;   //Pumpe 4
int pumpe5 = 5;   //Pumpe 5

int Sensor1 = A1;  //Wasserstandssensor 1
int Sensor2 = A2;  //Wasserstandssensor 2
int Sensor3 = A3;  //Wasserstandssensor 3
int Sensor4 = A4;  //Wasserstandssensor 4
int Sensor5 = A5;  //Wasserstandssensor 5
```

```
48 //Nextion Elemente
49 NexButton j0 = NexButton(1, 10, "j0");    //progressbar
50 NexButton j1 = NexButton(1, 11, "j1");    //progressbar
51 NexButton j2 = NexButton(1, 12, "j2");    //progressbar
52 NexButton j3 = NexButton(1, 13, "j3");    //progressbar
53 NexButton j4 = NexButton(1, 14, "j4");    //progressbar
54 NexButton res = NexButton(1, 19, "res"); //resetbutton

56 NexSlider h0 = NexSlider(2, 2, "h0"); //Slider
57 NexButton b3 = NexButton(2, 7, "b3"); //Savebutton

59 NexDSButton bt0 = NexDSButton(3, 4, "bt0");   //DSButton zum Spuelen
60 NexDSButton bt1 = NexDSButton(3, 5, "bt1");   //DSButton zum Spuelen
61 NexDSButton bt2 = NexDSButton(3, 6, "bt2");   //DSButton zum Spuelen
62 NexDSButton bt3 = NexDSButton(3, 14, "bt3"); //DSButton zum Spuelen
63 NexDSButton bt4 = NexDSButton(3, 15, "bt4"); //DSButton zum Spuelen
64 NexButton b2 = NexButton(3, 3, "b2");         //Back im Wartungsmenu

66 NexPage page0 = NexPage(0, 0, "Startseite");   //page
67 NexPage page1 = NexPage(1, 0, "Auswahl");      //page
68 NexPage page2 = NexPage(2, 0, "Einstellung"); //page
69 NexPage page3 = NexPage(3, 0, "Wartung");      //page

71 //char buffer[100] = {0};                      //fuer Text

73 NexTouch *nex_listen_list[] =  //Nextion Elemente einbinden
74   {
75     &j0,
76     &j1,
77     &j2,
78     &j3,
79     &j4,
80     &res,
81     &h0,
82     &b3,
83     &bt0,
84     &bt1,
85     &bt2,
86     &bt3,
87     &bt4,
88     &b2,
89     &page0,
90     &page1,
91     &page2,
92     &page3,
93     NULL
94   };
95 //Events
96 void j0PopCallback(void *ptr)  //prog1 release
97 {
98   S = 1;
99 }
100 void j1PopCallback(void *ptr)  //prog1 release
101 {
102   S = 2;
103 }
104 void j2PopCallback(void *ptr)  //prog1 release
105 {
106   S = 3;
107 }
108 void j3PopCallback(void *ptr)  //prog1 release
109 {
110   S = 4;
111 }
112 void j4PopCallback(void *ptr)  //prog1 release
113 {
114   S = 5;
115 }
116 void resPopCallback(void *ptr)  //reset release -> ruecksetzen auf pwm=0
117 {
118   pwm1 = 0;
119   pwm2 = 0;
120   pwm3 = 0;
121   pwm4 = 0;
```

```
122   pwm5 = 0;
123 }
124 void h0PopCallback(void *ptr)  //  slider release
125 {
126 }
127 void b3PopCallback(void *ptr)  //Save release
128 {
129   uint32_t sl = 0;                //Variable
130   h0.getValue(&sl);               //Sliderwert in Variable speichern
131   for (int i = 0; i < 10; i++) {  //Sicherstellen das Wert richtig uebernommen wurde
132     if (sl == 0) {
133       h0.getValue(&sl);
134     }
135   }
136   if (sl != 0) {
137     if (S == 1) {  //Feststellen welcher in welche PWM der Sliderwert uebernommen werden soll
138       //pwm1 = 125 + ((sl/100) * 130);
139       pwm1 = (sl * 0.5 + 50) * 2.55;  //0-100 in 0-255 umwandeln 127,5
140     } else if (S == 2) {
141       //pwm2 = 125 + ((sl/100) * 130);
142       pwm2 = (sl * 0.5 + 50) * 2.55;
143     } else if (S == 3) {
144       //pwm3 = 125 + ((sl/100) * 130);
145       pwm3 = (sl * 0.5 + 50) * 2.55;
146     } else if (S == 4) {
147       //pwm4 = 125 + ((sl/100) * 130);
148       pwm4 = (sl * 0.5 + 50) * 2.55;
149     } else if (S == 5) {
150       //pwm5 = 125 + ((sl/100) * 130);
151       pwm5 = (sl * 0.5 + 50) * 2.55;
152     }
153   }
154 }

156 void bt0PopCallback(void *ptr)  // DSbutton release
157 {
158   uint32_t bt_0 = 0;    // Variable
159   bt0.getValue(&bt_0);  // Status des DSB in Variable speichern

161   if (bt_0 == 1) {  //DS an -> Pumpe an
162     pwm1 = 255;
163   } else {  //DS aus -> Pumpe aus
164     pwm1 = 0;
165   }
166 }
167 void bt1PopCallback(void *ptr) {
168   uint32_t bt_1 = 0;
169   bt1.getValue(&bt_1);

171   if (bt_1 == 1) {
172     pwm2 = 255;
173   } else {
174     pwm2 = 0;
175   }
176 }
177 void bt2PopCallback(void *ptr) {
178   uint32_t bt_2 = 0;
179   bt2.getValue(&bt_2);

181   if (bt_2 == 1) {
182     pwm3 = 255;
183   } else {
184     pwm3 = 0;
185   }
186 }
187 void bt3PopCallback(void *ptr) {
188   uint32_t bt_3 = 0;
189   bt3.getValue(&bt_3);

191   if (bt_3 == 1) {
192     pwm4 = 255;
193   } else {
194     pwm4 = 0;
195   }
```

```
196 }
197 void bt4PopCallback(void *ptr) {
198   uint32_t bt_4 = 0;
199   bt4.getValue(&bt_4);

201   if (bt_4 == 1) {
202     pwm5 = 255;
203   } else {
204     pwm5 = 0;
205   }
206 }
207 void b2PopCallback(void *ptr)  //back release
208 {
209   pwm1 = 0;  //sicherstellen das beim verlassen des Wartungsmenus wieder alle Pumpen aus sind
210   pwm2 = 0;
211   pwm3 = 0;
212   pwm4 = 0;
213   pwm5 = 0;
214 }
215 void page0PushCallback(void *ptr)  // page init
216 {
217   CurrentPage = 0;  //             //aktuelle Seite merken
218 }
219 void page1PushCallback(void *ptr) {
220   CurrentPage = 1;
221 }
222 void page2PushCallback(void *ptr) {
223   CurrentPage = 2;
224 }
225 void page3PushCallback(void *ptr) {
226   CurrentPage = 3;
227 }

229 int getValueAt(String s, int vpos) {
230   int pos = 0;
231   int valueCount = 0;
232   for (int i = 0; i < s.length(); i++) {
233     if (s.charAt(i) == ',' || i == s.length() - 1) {
234       String value = "";
235       int bound = (i == s.length() - 1 ? i + 1 : i);
236       for (int j = pos; j < bound; j++) {
237         value.concat(s.charAt(j));
238       }
239       if (valueCount == vpos) {
240         return value.toInt();
241       }
242       pos = i + 1;
243       valueCount++;
244     }
245   }
246   return -1;
247 }

249 void setup() {
250   S = 0;                      //kein pwm Wert auswaehlen
251   pinMode(Schalter, INPUT);  //Eingaenge und Ausgaenge definieren
252   pinMode(Sensor1, INPUT);
253   pinMode(Sensor2, INPUT);
254   pinMode(Sensor3, INPUT);
255   pinMode(Sensor4, INPUT);
256   pinMode(Sensor5, INPUT);
257   pinMode(pumpe1, OUTPUT);
258   pinMode(pumpe2, OUTPUT);
259   pinMode(pumpe3, OUTPUT);
260   pinMode(pumpe4, OUTPUT);
261   pinMode(pumpe5, OUTPUT);

263   // 16(TX2) and 17(RX2)
264   Serial2.begin(115200);          //Kommunikation starten(Baud:115200)
265   delay(500);                     //Vorgang kann etwas dauern
266   Serial2.print("baud=115200");  //Nextion auf gleiche Baudrate stellen
267   Serial2.write(0xff);            //3 Endtags damit Nextion weiss alles wurde gesendet
268   Serial2.write(0xff);
269   Serial2.write(0xff);
```

```
270   //Events einbinden
271   j0.attachPop(j0PopCallback);
272   j1.attachPop(j1PopCallback);
273   j2.attachPop(j2PopCallback);
274   j3.attachPop(j3PopCallback);
275   j4.attachPop(j4PopCallback);
276   res.attachPop(resPopCallback);

278   h0.attachPop(h0PopCallback);
279   b3.attachPop(b3PopCallback);

281   bt0.attachPop(bt0PopCallback);
282   bt1.attachPop(bt1PopCallback);
283   bt2.attachPop(bt2PopCallback);
284   bt3.attachPop(bt3PopCallback);
285   bt4.attachPop(bt4PopCallback);
286   b2.attachPop(b2PopCallback);

288   page0.attachPush(page0PushCallback);
289   page1.attachPush(page1PushCallback);
290   page2.attachPush(page2PushCallback);

292   // Initiate serial communication with the Raspberry on standard serial port
293   Serial.begin(9600);
294 }

296 void loop() {
297   nexLoop(nex_listen_list);           //Nextion events ueberpruefen
298   if (digitalRead(Schalter) == HIGH)  //Fuellen Knopf an -> PWM Werte an Pumpen geben
299   {
300     if (analogRead(Sensor1) <= 800) {  //Read max: 1024
301       analogWrite(pumpe1, pwm1);
302     }
303     if (analogRead(Sensor2) <= 800) {
304       analogWrite(pumpe2, pwm2);
305     }
306     if (analogRead(Sensor3) <= 800) {
307       analogWrite(pumpe3, pwm3);
308     }
309     if (analogRead(Sensor4) <= 800) {
310       analogWrite(pumpe4, pwm4);
311     }
312     if (analogRead(Sensor5) <= 800) {
313       analogWrite(pumpe5, pwm5);
314     }
315   } else if (Serial.available() > 0) {
316     String data = Serial.readStringUntil('\n');  // e.g. "50,50,0,0"
317     Serial.println("Received " + data);
318     analogWrite(pumpe1, (getValueAt(data, 0) * 0.5 + 50) * 2.55);
319     analogWrite(pumpe2, (getValueAt(data, 1) * 0.5 + 50) * 2.55);
320     analogWrite(pumpe3, (getValueAt(data, 2) * 0.5 + 50) * 2.55);
321     analogWrite(pumpe4, (getValueAt(data, 3) * 0.5 + 50) * 2.55);
322     delay(3000);  // let the "button" be "pressed" for three seconds
323   } else //Fuellen Knopf aus -> Pumpen aus
324   {
325     analogWrite(pumpe1, 0);
326     analogWrite(pumpe2, 0);
327     analogWrite(pumpe3, 0);
328     analogWrite(pumpe4, 0);
329     analogWrite(pumpe5, 0);
330   }
331 }
```

Listing A.1: Arduino-Programmcode für die Mischmaschine

Anhang B

Spracherkennung auf dem Raspberry Pi

```
1  from ctypes import *
2  import speech_recognition as sr
3  import pyttsx3 as tts
4  import serial
5  import time
6  import language_model

8  ERROR_HANDLER_FUNC = CFUNCTYPE(None, c_char_p, c_int, c_char_p, c_int, c_char_p)
9  def py_error_handler(filename, line, function, err, fmt):
10         pass
11 c_error_handler = ERROR_HANDLER_FUNC(py_error_handler)

13 asound = cdll.LoadLibrary("libasound.so")
14 asound.snd_lib_error_set_handler(c_error_handler)

16 r = sr.Recognizer()
17 m = sr.Microphone()
18 with m as source:
19         r.adjust_for_ambient_noise(source)

21 engine = tts.init()
22 voices = engine.getProperty("voices")
23 engine.setProperty("voice", voices[1].id)

25 ser = serial.Serial("/dev/ttyACM0", 9600, timeout=5)
26 ser.reset_input_buffer()
27 time.sleep(5)


30 while True:
31         asound.snd_lib_error_set_handler(c_error_handler)
32         print("Start listening...")
33         with m as source:
34                 audio = r.listen(source)
35         asound.snd_lib_error_set_handler(None)
36         text = ""
37         try:
38                 recognized_text = r.recognize_whisper(audio, language="german", model="tiny")
39                 text = recognized_text
40                 print("Recognized text: " + text)
41         except sr.UnknownValueError:
42                 print("Whisper could not understand audio.")
43         except sr.RequestError as e:
44                 print("Could not request results from Whisper; {0}".format(e))

46         if text.lower().strip().startswith("mischmaschine"):
47                 print("Recognized hot word. Will output answer and command.")
```

```
48                  answer, cmd = language_model.get_response(text)
49                  engine.say(answer)
50                  engine.runAndWait()
51                  engine.stop()

53                  ser.write(cmd.encode("utf-8"))
54                  serial_answer = ser.readline().decode("utf-8").rstrip()
55                  print(serial_answer)
```

Listing B.1: Spracherkennung auf dem Raspberry Pi

Anhang C

Sprachverarbeitung auf dem Raspberry Pi

```
1  import nltk
2  import pickle
3  import numpy
4  import json
5  import random
6  import re

8  from keras.models import load_model
9  from nltk.stem import WordNetLemmatizer

11 lemmatizer = WordNetLemmatizer()
12 nltk.download('punkt')
13 nltk.download('wordnet')

15 model = load_model('dialog_model.h5')
16 intents = json.loads(open('patterns.json').read())
17 words = pickle.load(open('words_for_pattern.pkl','rb'))
18 classes = pickle.load(open('model_classes.pkl','rb'))

20 def parse_line(line):
21     reservoirs = [0] * 4
22     line = re.sub(r'[,.!?]', '', line)
23     words = line.split()
24     for i in range(len(words)):
25         if words[i] == 'Behaelter' or words[i] == 'Reservoir':
26             for x in range(i+1, len(words)):
27                 if words[x].isdigit():
28                     index = int(words[x]) - 1
29                     break
30             for x in range(i-1, 0, -1):
31                 if words[x].isdigit():
32                     reservoirs[index] = int(words[x])
33                     break
34     return tuple(reservoirs)

36 def clean_up_sentence(sentence):
37     sentence_words = nltk.word_tokenize(sentence)
38     sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
39     return sentence_words

41 def bow(sentence, words):
42     sentence_words = clean_up_sentence(sentence)
43     bag = [0]*len(words)
44     for s in sentence_words:
45         for i,w in enumerate(words):
46             if w == s:
47                 bag[i] = 1
```

```
48      return(numpy.array(bag))

50  def predict_class(sentence, model):
51      p = bow(sentence, words)
52      res = model.predict(numpy.array([p]))[0]
53      ERROR_THRESHOLD = 0.75
54      results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
55      results.sort(key=lambda x: x[1], reverse=True)
56      return_list = []
57      for r in results:
58          return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
59      return return_list

61  def get_dialog_response(ints, intents_json):
62      tag = ints[0]['intent']
63      list_of_intents = intents_json['intents']
64      for i in list_of_intents:
65          if(i['tag']== tag):
66              result = random.choice(i['responses'])
67              break
68      return result

70  def get_response(msg):
71      ints = predict_class(msg, model)
72      if not ints:
73          ints.append({"intent": "noanswer", "probability": ""})
74      nums = [0] * 4
75      tag = ints[0]['intent']
76      if(tag == "order"):
77         nums = parse_line(msg)
78      res = get_dialog_response(ints, intents)
79      result = ','.join(str(num) for num in nums) + '\n'
80      return res, result
```

Listing C.1: Implementierung des Dialogsystems

Anhang D

Schaltplan

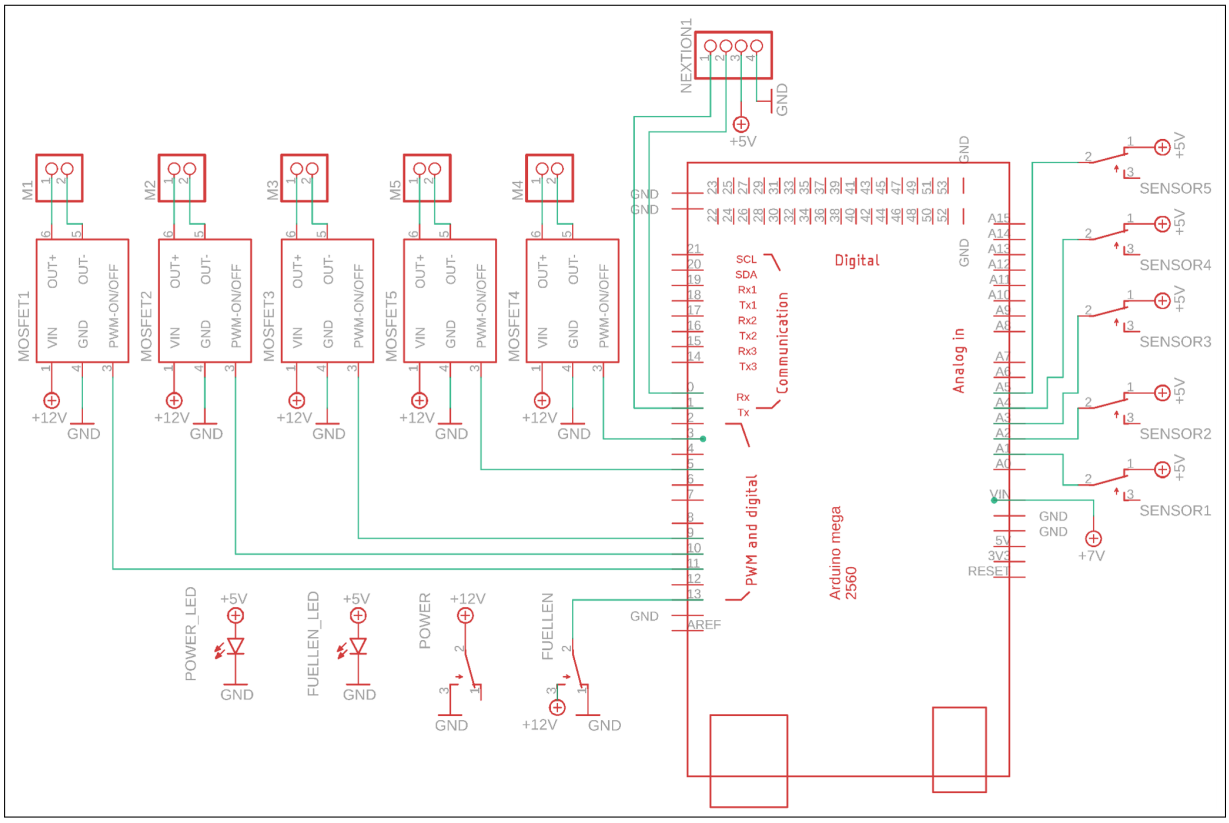

Abbildung D.1: Schaltplan

Anhang E

Deep Learning Modell für die Klassifizierung

```
import nltk
import json
import pickle
import numpy
import random

from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import SGD

from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

nltk.download('punkt')
nltk.download('wordnet')

words_for_pattern=[]
model_classes = []
documents = []
ignore_words = ['?', '!']
data_file = open('patterns.json').read()
intents = json.loads(data_file)

for intent in intents['intents']:
    for pattern in intent['patterns']:
        w = nltk.word_tokenize(pattern)
        words_for_pattern.extend(w)
        documents.append((w, intent['tag']))
        if intent['tag'] not in model_classes:
            model_classes.append(intent['tag'])

words_for_pattern = [lemmatizer.lemmatize(w.lower()) for w in words_for_pattern if w not in ignore_words]
words_for_pattern = sorted(list(set(words_for_pattern)))

model_classes = sorted(list(set(model_classes)))

pickle.dump(words_for_pattern, open('words_for_pattern.pkl', 'wb'))
pickle.dump(model_classes, open('model_classes.pkl', 'wb'))

training = []
output_empty = [0] * len(model_classes)
for doc in documents:
    bag = []
    pattern_words = doc[0]
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]
    for w in words_for_pattern:
        bag.append(1) if w in pattern_words else bag.append(0)
```

```
49      output_row = list(output_empty)
50      output_row[model_classes.index(doc[1])] = 1
51      training.append([bag, output_row])

53  random.shuffle(training)
54  training = numpy.array(training)

56  train_x = list(training[:,0])
57  train_y = list(training[:,1])

59  model = Sequential()
60  model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
61  model.add(Dropout(0.5))
62  model.add(Dense(64, activation='relu'))
63  model.add(Dropout(0.5))
64  model.add(Dense(len(train_y[0]), activation='softmax'))

66  sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
67  model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

69  hist = model.fit(numpy.array(train_x), numpy.array(train_y), epochs=500, batch_size=5, verbose=1)
70  model.save('dialog_model.h5', hist)
```

Listing E.1: Deep Learning Modell für die Klassifizierung