Dokumentation

Entwicklung eines Sprachassistenten Online/Offline für Android

eingereicht von

Robert Liefke, Leon Augustat, Thomas Pattoka

Technische Universität Dresden

Fakultät Informatik
Institut für Angewandte Informatik
Lehrstuhl Mensch-Computer-Interaktion

Betreuer:
David Gollasch, M. Sc.

Hochschullehrer:
Prof. Dr. rer. nat. habil. Gerhard Weber

Eingereicht am 14. September 2021

Abstract

Die vorliegende Dokumentation gibt einen Überblick über das Projekt, welches im Rahmen des Komplexpraktikums „Mensch-Computer-Interaktion“ entwickelt wurde. Das Ziel war es, eine Alternative zu kommerziellen Sprachassistenten zu schaffen. Dazu wurden bestehende OpenSource-Anwendungen und -Frameworks kombiniert, um eine neue Lösung zu finden. Aufbauend auf Mycroft als zentrale Verarbeitungseinheit wurde eine Android-Applikation entwickelt, die mit einer großen Menge an Endgeräten kompatibel ist. Dazu wurde zum einen Vosk zur Spracherkennung und zum anderen Larynx zur Sprachsynthese verwendet. Durch den offenen Quellcode der Anwendung kann der Sprachassistent individuell angepasst werden. Außerdem können die Funktionen des Sprachassistenten durch sogenannte „Skills“, wie zum Beispiel die Erstellung eines Termins, erweitert werden.

Inhaltsverzeichnis

Abstract I

1 Einleitung 1
1.1 Motivation . . . 1
1.2 Zielstellung . . . 1

2 Analyse 3
2.1 Funktionale Anforderungen . . . 3
2.2 Nichtfunktionale Anforderungen . . . 3
2.3 Komponentenvergleich . . . 4

3 Konzept 7
3.1 Allgemeiner Aufbau des Sprachassistenten . . . 7
3.2 Ausgewählte Komponenten . . . 8

4 Implementierung 11
4.1 Interaktion und Architektur . . . 11

5 Installation 13
5.1 Installation der App . . . 13
5.2 Einrichtung des lokalen Servers . . . 13

6 Organisation des Projektes 17
6.1 Zeitlicher Ablauf . . . 17
6.2 Verantwortlichkeiten . . . 17

7 Zusammenfassung 19
7.1 Weitere Entwicklungsideen . . . 19

Literatur i

1 Einleitung

1.1 Motivation

Sprachassistenten können den Alltag ungemein erleichtern. Sie können an Termine erinnern, einen Wecker stellen oder nur den Wetterbericht vorlesen. Es existieren zahlreiche kommerzielle Lösungen wie „Alexa“[1] von Amazon, „Siri“[2] von Apple oder der „Google Assistant“[3]. In einer Umfrage im Jahr 2019 gaben 60% der Teilnehmenden an, schon einmal einen Sprachassistenten genutzt zu haben [RES19]. Die kommerziellen Lösungen basieren meist auf proprietärer Software, die nur mit vorhandener Internetverbindung funktionieren. Dadurch entstehen einerseits Bedenken bezüglich des Datenschutzes und andererseits können diese Assistenten bei schlechter Internetverbindung nicht eingesetzt werden. Des Weiteren können die Sprachassistenten nicht beliebig an individuelle Bedürfnisse angepasst werden.

1.2 Zielstellung

Um die Probleme aktueller kommerzieller Sprachassistenten zu adressieren, soll ein neuer Sprachassistent entwickelt werden, der vollständig auf OpenSource-Lösungen basiert. Durch den frei zugänglichen Quellcode soll der Sprachassistent beliebig angepasst werden können, um jeglichen individuellen Bedürfnissen gerecht zu werden. Des Weiteren soll der Sprachassistent offline-fähig sein, sodass einfache Funktionalitäten ohne Internetverbindung zugänglich sind.

[1] `https://developer.amazon.com/de-DE/alexa`
[2] `https://www.apple.com/de/siri/`
[3] `https://assistant.google.com/intl/de_de/`

2 Analyse

2.1 Funktionale Anforderungen

WakeWord-Erkennung Bei der WakeWord-Erkennung wird mittels eines festgelegten Aktivierungswortes, des sogenannten WakeWord, die Spracherkennung aktiviert. Es soll möglich sein, die Sprachinteraktion mittels eines vorgegebenen Aktivierungswortes zu starten, sodass keine physische Interaktion mit dem Gerät notwendig ist.

Spracherkennung Speech to Text, bzw. die Spracherkennung, wandelt Audiosignale in Text um. Dies ermöglicht Befehle verbal zu formulieren. Es soll möglich sein, dass die Spracherkennung gesprochene Sprache in Text umwandeln kann.

Verarbeiten von Anfragen und Ausführen von Skills Der Sprachassistent soll in der Lage sein aus einem ihm gegebenen Text Intentionen und relevante Informationen abzuleiten. Diese sollen dann von Skills innerhalb des Sprachassistenten verarbeitet werden. Die Skills liefern ihrerseits eine Antwort an den Nutzer oder aber stellen Rückfragen für weitere benötigte Informationen.

Sprachsynthese Die Sprachsynthese, auch Text-to-Speech genannt, beschäftigt sich mit der Generierung von akustischen Sprachausgaben aus Text. Der Sprachassistent sollte dazu fähig sein Text, welchen er von den Skills geliefert bekommt, in eine Sprachausgabe umzuwandeln, welche von einem Menschen verstanden werden kann.

Offline-Skills Skills, welche keine Informationen aus dem Internet abrufen, sollen auch ohne eine Internetverbindung nutzbar sein.

2.2 Nichtfunktionale Anforderungen

Fehlerrate bei WakeWord-Erkennung Die Fehlerrate sollte in einem angemessenen Rahmen bleiben.

Anpassbare WakeWords Das WakeWord sollte veränderbar sein, sodass eigene WakeWords verwendet werden können.

Fehlerrate der Spracherkennung Die Fehlerrate der Spracherkennung sollte minimiert werden, sodass keine wichtigen Informationen verloren gehen.

Eigene Skills Es sollte möglich sein, eigene Skills zu entwickeln und diese in den Sprachassistenten zu integrieren.

Aufrufen von Skills Das Aufrufen eines Skills sollte auf verschiedene Art und Weise möglich, beispielsweise durch verschiedene mögliche Eingabephrasen.

Variabilität der Antworten Skills sollten in der Lage sein, die Antworten in verschiedenen Formen auszudrücken, sodass die Antworten nicht immer den gleichen Text enthalten.

Qualität der Sprachausgabe Die Sprachausgabe sollte möglichst nah an der menschlichen Sprache sein und nicht nach einer computergenerierten Stimme klingen.

2.3 Komponentenvergleich

Im Allgemeinen bestehen Sprachassistenten aus den vier Komponenten: WakeWord-Erkennung, Spracherkennung, Verarbeitung und Sprachausgabe. Für diese Komponenten stehen verschiedene Lösungen zur Verfügung, deren Vor- und Nachteile im Folgenden verglichen werden.

WakeWord-Erkennung		
Framework	**Pro**	**Contra**
Porcupine (Picovoice)	- leichtgewichtig - OpenSource - präzise	- eigene WakeWords nicht kostenfrei - kommerzielle Nutzung kostet Geld
PocketSphinx	- sehr leichtgewichtig - OpenSource - eigene WakeWords	- unpräzise

Spracherkennung		
Framework	**Pro**	**Contra**
Google Cloud STT	- sehr präzise	- nicht OpenSource - nur erste Stunde kostenlos - nicht offline-fähig
Mozilla DeepSpeech	- OpenSource - offline-fähig	- nur für RaspberryPi etc. geeignet (Smartphones nicht leistungsfähig genug)
Vosk (Alpha Cephei)	- OpenSource - leichtgewichtig - präzise - offline-fähig - auf Smartphone lauffähig	
PocketSphinx	- OpenSource - sehr leichtgewichtig - offline-fähig - auf Smartphone lauffähig	- unpräzise

Verarbeitung		
Framework	**Pro**	**Contra**
Mycroft Core	- OpenSource - in vielen Sprachen implementiert - erweiterbar durch Skills - (begrenzt) offline-fähig - kostenfrei	- keine große Auswahl an fertigen Skills
DialogFlow	- sehr ausgereift	- nicht kostenfrei - nicht offline-fähig
Leon	- OpenSource	- nur in English bzw. Französisch - nicht ausgereift

Framework	Pro	Contra
Sprachsynthese		
Google TTS	- basiert auf Machine Learning - kompatibel mit SSML - menschenähnliche Sprachausgabe	- nicht offline-fähig - nicht vollständig kostenlos - nicht OpenSource
Tensorflow TTS		- sehr ressourcenaufwändig
Mozilla TTS	- basiert auf Machine Learning - menschenähnliche Sprachausgabe - offline-fähig	- nicht kompatibel mit SSML
Coqui AI	- basiert auf Machine Learning - menschenähnliche Sprachausgabe - offline-fähig	- kein offizielles deutsches Modell - langsame Synthese - nicht kompatibel mit SSML
MaryTTS	- Anpassungsmöglichkeiten über XML-Format - schnelle Ausgabe - offline-fähig	- robotisch klingende Ausgabe - basiert auf Hidden Markov Model - geringe Auswahl an Anpassungsmöglichkeiten
Larynx	- basiert auf Machine Learning - menschenähnliche Sprachausgabe - OpenSource - vergleichsweise schnelle Synthese - offline-fähig	- nicht kompatibel mit SSML - Sprachausgabe könnte besser sein

3 Konzept

Der allgemeine Lösungsansatz besteht darin, mehrere bereits verfügbare OpenSource-Komponenten miteinander zu kombinieren, um einen neuen Sprachassistenten zu schaffen, der den gegebenen Anforderungen entspricht.

3.1 Allgemeiner Aufbau des Sprachassistenten

Im Allgemeinen besteht der Sprachassistent aus den gleichen Komponenten wie kommerzielle Sprachassistenten: WakeWord-Erkennung, Spracherkennung (Speech-To-Text), Dialogverarbeitung und Sprachausgabe (Text-To-Speech). Dabei interagieren die Komponenten folgendermaßen (siehe auch Abbildung 3.1): Die WakeWord-Erkennung wird permanent im Hintergrund ausgeführt. Sobald diese ein festgelegtes WakeWord erkennt, wird die Spracheingabe aktiviert und die Nutzenden können ihren Befehl oder ihre Frage formulieren, welche dann von der Spracherkennung in Text umgewandelt werden. Dieser Text wird an die Dialogverarbeitung weitergeleitet, welche daraufhin den Befehl ausführt oder die Antwort auf die Frage bestimmt. Danach wird die Antwort von der Dialogverarbeitung an die Sprachausgabe weitergeleitet, damit die Antwort auditiv ausgegeben werden kann.

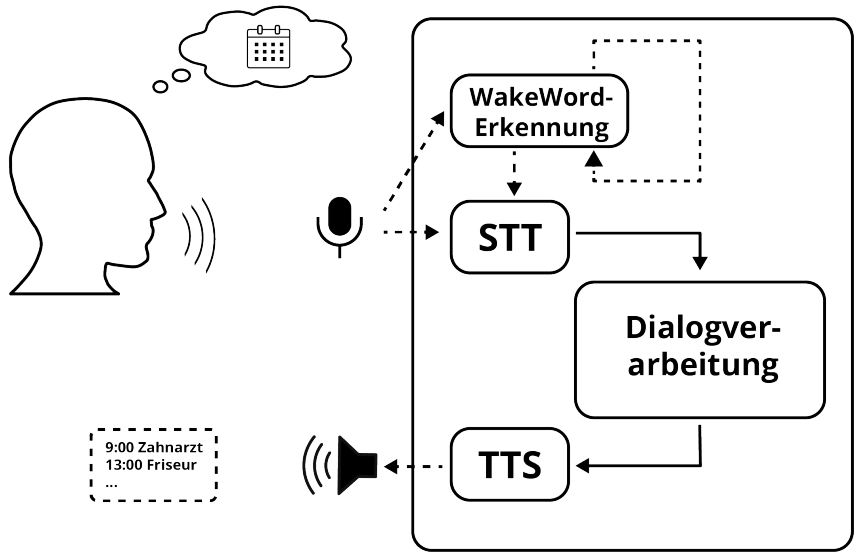

Abbildung 3.1 – Allgemeiner Aufbau des Sprachassistenten

3.2 Ausgewählte Komponenten

Im Folgenden werden die ausgewählten Software-Komponenten kurz vorgestellt und besondere Merkmale hervorgehoben (siehe Abbildung 3.2).

3.2.1 WakeWord-Erkennung

Für die Erkennung des WakeWords wird „Porcupine“ von Picovoice eingesetzt. Die Installation und Einrichtung wird dadurch erleichtert, dass bereits einige WakeWords vorhanden sind, die einfach eingesetzt werden können. Mithilfe der bereitgestellten Demo-Apps[1] kann der Dienst unkompliziert in das Projekt eingebunden werden. Außerdem können eigene WakeWords trainiert werden, um den Sprachassistenten mit ihnen zu aktivieren.

3.2.2 Speech-To-Text

Um die Sprachbefehle in Text umzuwandeln, wird „Vosk“[2] von Alpha Cephei eingesetzt. Durch die Offline-Fähigkeit und den geringen Rechenaufwand bot sich diese Toolkit besonders für das Projekt an. Die Android-Demo[3] erleichtert dabei die Integration in die eigene Anwendung.

3.2.3 Dialogverarbeitung

Für die Verarbeitung der Befehle wird „Mycroft“[4] verwendet, da es sehr ausgereift ist und ständig weiterentwickelt wird. Außerdem können die Funktionen durch eigene „Skills“ erweitert werden.

3.2.4 Text-To-Speech

Der Dienst „Larynx“[5] wird eingesetzt, um die schriftlichen Antworten von Mycroft in Sprache umzuwandeln. Es wurde ausgewählt, da es vollständig offline funktioniert und die Hardware-Anforderungen angemessen sind. Andere Dienste wie „Tensorflow-TTS“ führen zwar zu natürlicheren Ergebnissen, jedoch sind die Hardware-Anforderungen zu hoch bzw. ist die Sprachsynthese mit einer langen Wartezeit verbunden.

[1] https://github.com/Picovoice/porcupine
[2] https://alphacephei.com/vosk/
[3] https://github.com/alphacep/vosk-android-demo
[4] https://mycroft.ai
[5] https://github.com/rhasspy/larynx

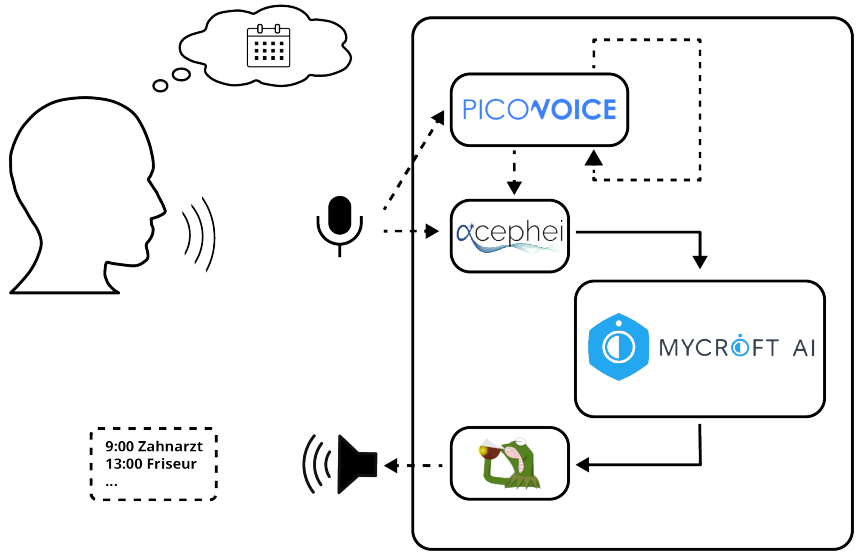

Abbildung 3.2 – Ausgewählte Komponenten

4 Implementierung

Der Sprachassistent wurde implementiert, indem auf der Beispiel-App von Mycroft für Android[1] aufgesetzt wurde. In dieser App wurde zunächst die Google-Spracheingabe durch Vosk ersetzt. Um den Sprachassistenten nutzen zu können, wurde daraufhin der Mycroft-Core auf einem RaspberryPi installiert. Danach wurde die Sprachausgabe der Applikation implementiert, indem ein Larynx-Server auf dem RaspberryPi installiert wurde und eine Schnittstelle dafür in der App geschaffen wurde. Der nächste Schritt bestand in der Implementierung der WakeWord-Erkennung, wofür der Dienst Porcupine in die App eingebunden wurde.

Umwandlung von Zahlen Während der Implementierung fiel auf, dass viele vorprogrammierte Skills von Mycroft nicht mit ausgeschriebenen bzw. gesprochenen Zahlen umgehen können. Um dieses Problem zu lösen, wurde ein simpler Python-Webserver auf dem RaspberryPi implementiert, welcher „text2num“[2] nutzt, um die gesprochenen Zahlen innerhalb der Sprachbefehle in Ziffern umzuwandeln.

4.1 Interaktion und Architektur

Die Interaktion mit dem entwickelten Sprachassistenten läuft meist wie folgt ab:

- WakeWord-Erkennung wird im Hintergrund ausgeführt
- Nutzer:in spricht WakeWord (bspw. „Computer“) aus
- Spracheingabe wird automatisch gestartet
- Nutzer:in spricht Frage oder Befehl aus
- Anfrage wird durch Vosk in Text umgewandelt
- Text wird an text2num-Server gesendet, um gesprochene Zahlen durch Ziffern zu ersetzen
- verarbeiteter Text erreicht App und wird an Mycroft-Core gesendet
- Mycroft-Core verarbeitet Anfrage und sendet Antwort/Bestätigung zurück
- Antwort erscheint als Text in App
- Antwort wird an Larynx-Server gesendet
- Larynx-Server führt Sprachsynthese durch und sendet Audiodatei an App

[1] `https://github.com/MycroftAI/Mycroft-Android`
[2] `https://github.com/allo-media/text2num`

- App spielt Audiodatei ab

In Abbildung 4.1 ist eine beispielhafte Interaktion mit dem Sprachassistenten mit Bezug auf seiner Architektur schematisch dargestellt.

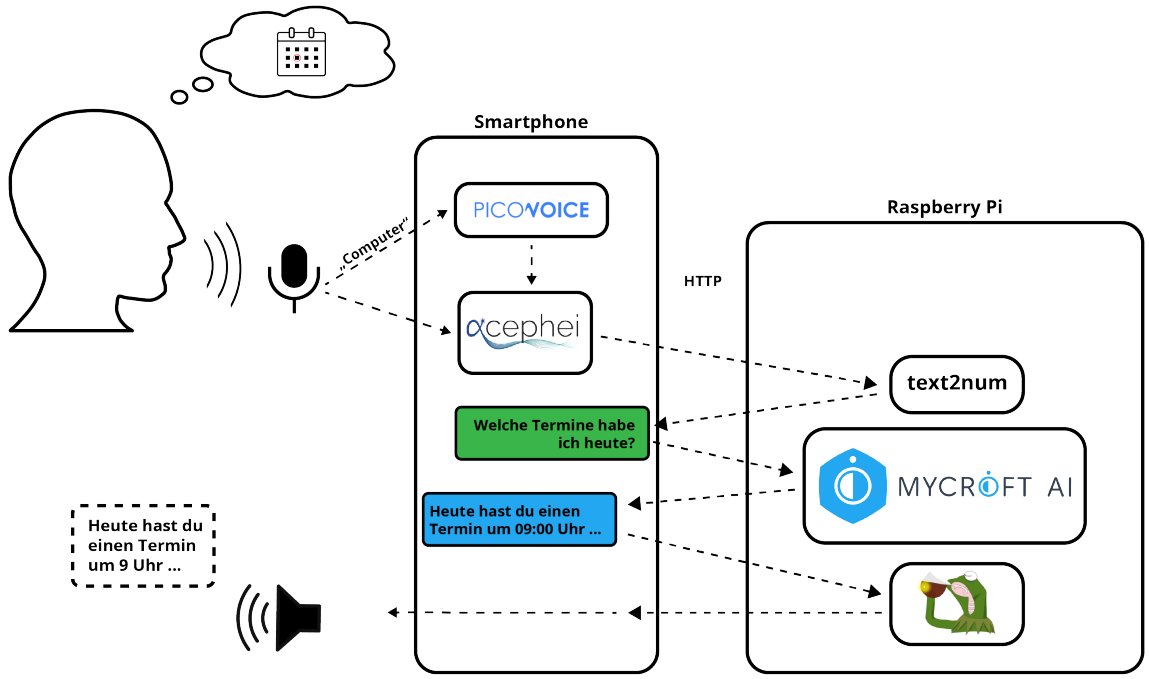

Abbildung 4.1 – Architektur des Sprachassistenten

Rückfragen Die Interaktion mit dem Sprachassistenten muss nicht zwangsweise aus einem reinen Frage-Antwort-Dialog bestehen. Die Skills können ebenfalls Rückfragen implementieren, um bspw. fehlende Informationen zu erfragen.

5 Installation

In diesem Kapitel wird die Installation der Android-App und die Einrichtung des lokalen Servers erläutert.

5.1 Installation der App

Über APK-Datei Wenn keine Veränderungen am aktuellen Stand der Applikation vorgenommen werden sollen, kann die App über eine bereits kompilierte apk-Datei direkt in Android installiert werden.

Manuelle Installation Um verschiedene Einstellungen anzupassen, empfiehlt sich die manuelle Installation über Android Studio. Dazu muss das Projekt in Android Studio importiert werden und kann anschließend kompiliert und auf einem (virtuellen) Smartphone installiert werden.

5.1.1 Einrichtung der App

Nach der Installation muss in den Einstellungen der App lediglich die IP-Adresse des lokalen Servers konfiguriert werden.

5.2 Einrichtung des lokalen Servers

Insgesamt werden auf dem lokalen Server 3 Dienste ausgeführt. Der zentrale Mycroft-Core stellt das gesamte Dialogmanagement für den Sprachassistenten bereit. Des Weiteren wird ein Larynx-Server für die Sprachsynthese ausgeführt. Ein weiteres nötiges Hilfsmittel stellt ein einfacher Python-Webserver dar, auf dem gesprochene Zahlen in Ziffern umgewandelt werden, das für die Spracheingabe benötigt wird.

Die Installation aller Dienste wurde auf einem RaspberryPi 4 mit Raspberry Pi OS 64 bit getestet, welches eine modifizierte Debian-Distribution ist. Die entsprechenden Images sind im offiziellen Downloadverzeichnis[1] der Raspberry Pi Foundation in den Ordnern „raspios_lite_arm64/“ bzw. „raspios_arm64/“ (mit zusätzlicher grafischer Oberfläche) zu finden.

Nachfolgend wird die Installation aller Teile beschrieben und zusätzlich auf die relevanten Dokumentationen verwiesen.

5.2.1 Mycroft-Core

Die Installation des Mycroft-Cores wird automatisiert durch einen Assistenten durchgeführt. Um die Installation zu starten, müssen folgende Befehle in einer beliebigen Kommandozeile ausgeführt werden:

[1] `https://downloads.raspberrypi.org/`

```
1    git clone https://github.com/MycroftAI/mycroft-core.git
2    cd mycroft-core
3    bash dev_setup.sh
```

Während der Installation werden einige Optionen abgefragt, wobei die empfohlenen Einstellungen übernommen werden können. Im Anschluss an die Installation kann der Mycroft-Core initial über folgenden Befehl im „mycroft-core“-Ordner gestartet werden:

```
1    ./start_mycroft.sh debug
```

Daraufhin öffnet sich ein Kommandozeilen-Interface, in dem nach einiger Zeit ein Code angezeigt wird, mit dem der Mycroft-Core mit Mycroft-Home verknüpft werden kann. Dafür muss jedoch zuerst online ein Konto bei Mycroft-Home[2] erstellt werden. Daraufhin kann der angezeigte Code online eingegeben werden. Die Verknüpfung mit Mycroft-Home ist empfehlenswert, weil damit die API-Keys für Online-Funktionen wie bspw. das Wetter automatisch generiert werden.

Im Anschluss kann das Kommandozeilen-Interface mit „Strg + C“ wieder beendet werden.

Als letzter Schritt muss Mycroft-Home noch auf Deutsch umgestellt werden, indem in der Datei „mycroft-core/mycroft/configuration/mycroft.conf“ die Zeile „”lang”: ën-us”“ durch „”lang”: ”de-de”“ ersetzt wird.

Um alle Mycroft-Services im Hintergrund zu starten, kann folgender Befehl im „mycroft-core“-Ordner ausgeführt werden:

```
1    ./start-mycroft.sh all
```

Eine detaillierte Anleitung ist außerdem auf der Webseite von Mycroft[3] zu finden.

5.2.2 Larynx-Server

Der lokale Larynx-Server kann auf verschiedene Arten eingerichtet werden. Die einfachste Installation ist über Docker-Images möglich. Um den Server mit einer deutschen Sprache zu installieren, muss folgender Befehl ausgeführt werden:

```
1    docker run -it -p 5002:5002 rhasspy/larynx:de-de
```

Des Weiteren kann Larynx über ein Debian Package installiert werden. Dazu muss das notwendige Paket im GitHub-Repository des Larynx-Projektes heruntergeladen werden[4] und kann bspw. über folgenden Befehl installiert werden:

```
1    sudo apt install ./larynx-tts_0.5.0_arm64.deb
```

Welches Paket konkret benötigt wird, hängt von der Systemarchitektur ab und ist ebenfalls im GitHub-Repository[5] aufgeschlüsselt. Nach der Installation muss einmalig das Webinterface des

[2] https://home.mycroft.ai/
[3] https://mycroft-ai.gitbook.io/docs/using-mycroft-ai/get-mycroft/linux
[4] https://github.com/rhasspy/larynx/releases
[5] https://github.com/rhasspy/larynx#debian-installation

Larynx-Servers aufgerufen werden, um die nötigen Sprachen zu installieren. Der Server kann mit folgendem Befehl gestartet werden:

```
1    larynx-server
```

Die Installation kann zusätzlich über die Python-Paketverwaltung pip erfolgen, wofür alle Anweisungen ebenso im GitHub-Repository[6] zu finden sind.

5.2.3 Python-Webserver

Für die Anwendung wird ein einfacher Python-Webserver benötigt, welcher Zahlenwerte, die von der Spracherkennung als Wörter erkannt werden, in Ziffern umwandelt. Dies ist notwendig, da die Skills des Sprachassistenten nicht mit ausgeschriebenen Zahlen umgehen können. Um die Installation des Webservers durchzuführen, muss zunächst das „text2num“-Projekt[7] heruntergeladen und die Setup-Datei ausgeführt werden:

```
1    git clone https://github.com/allo-media/text2num.git
2    python3 setup.py develop
```

Alle Instruktionen sind außerdem in der Dokumentation[8] des „text2num“-Projektes zu finden. Anschließend muss die Datei „text2num_server.py“ aus dem Thomicroft-Projektordner „server“ in das „text2num“-Projektverzeichnis kopiert werden. Innerhalb der „text2num_server.py“-Datei **muss** die IP-Adresse auf die IP-Adresse des Servers angepasst werden. Der Python-Webserver wird nun über folgenden Befehl gestartet:

```
1    python3 text2num_server.py
```

[6] https://github.com/rhasspy/larynx#python-installation
[7] https://github.com/allo-media/text2num
[8] https://text2num.readthedocs.io/en/stable/

6 Organisation des Projektes

6.1 Zeitlicher Ablauf

Kalenderwoche	Komponente(n)
18 - 20	Anforderungsanalyse
21 - 23	Auswahl der Komponenten
23 - 25	Installation Mycroft-Core & Import der Mycroft-Beispiel-App
25 - 26	Implementierung STT (Vosk)
26 - 29	TextToSpeech
29 - 31	WakeWord-Erkennung
32 - 33	Ergänzung text2num
33 - 36	Finalisierung der Dokumentation & Vorbereitung der Präsentation

6.2 Verantwortlichkeiten

Im Allgemeinen wurden zwar Verantwortlichkeiten für die einzelnen Teile des Projektes festgelegt, jedoch waren alle Teilnehmer an der Entwicklung aller Komponenten beteiligt. Nachfolgend werden die Verantwortlichkeiten der einzelnen Komponenten aufgeführt.

Person	Komponente
Robert Liefke	App-Entwicklung allgemein, STT
Thomas Pattoka	Backend-Server-Implementierung, TTS
Leon Augustat	Grafisches Design, WakeWord-Erkennung, Skills

7 Zusammenfassung

Insgesamt wurde ein Sprachassistent entwickelt, der einige Vorteile gegenüber aktuellen kommerziellen Lösungen bietet:

- basiert auf OpenSource-Anwendungen
- vollständig offener Quellcode
- offline-fähig
- individuell anpassbar & erweiterbar

Aufgrund dieser Eigenschaften kann das Vertrauen in den Sprachassistenten gefördert und seine Funktionen individualisiert werden.

7.1 Weitere Entwicklungsideen

Um den Sprachassistenten weiter zu verbessern, könnten noch einige Veränderungen vorgenommen und neue Funktionen implementiert werden, welche außerhalb des Umfangs des Projektes liegen:

Austausch der Sprachsynthese Die Sprachsynthese könnte bspw. durch „TensorflowTTS“ ersetzt werden, um menschenähnliche Ergebnisse bei der Sprachsynthese zu erhalten. Dafür müsste jedoch die Rechenleistung des lokalen Servers erhöht werden oder ein TFlite-Model trainiert werden, was mehrere Tage bis Wochen in Anspruch nehmen kann.

Implementierung weiterer Skills Es könnten weitere Skills implementiert werden, um den Funktionsumfang des Sprachassistenten zu erweitern und ihn somit nützlicher für den Alltag zu machen.

Individualisierung Um die Individualisierbarkeit zu verbessern, könnte ein Optionsmenü innerhalb der App erstellt werden, in dem einfache Parameter des Sprachassistenten direkt angepasst werden können.

Literatur

[RES19] SPLENDID RESEARCH. *Haben Sie schon einmal ein Gerät mit Hilfe einer Sprachsteuerung bedient?* Statista. 18. Apr. 2019. URL: https://de.statista.com/statistik/daten/studie/1031256/umfrage/umfrage-zur-verwendung-von-sprachsteuerungen-in-deutschland/ (besucht am 31. 08. 2021) (siehe S. 1).