Vladimir Zhelezarov
.....
.....
.....

Projektbericht

Thema:

Konzept und prototypische Entwicklung eines hybriden virtuellen Assistenten für einen besseren Schutz der Privatsphäre

.....

.....

Inhaltsverzeichnis

Abbildungsverzeichnis

Programmlistings

1 Einleitung

„Smarte“ oder intelligente Geräte finden immer mehr Verbreitung in der Wirtschaft und in den Haushalten und die Tendenz ist steigend. Zu Hause ist der Kühlschrank schon intelligent genug um sich an unseren Bedürfnissen anzupassen, der Fernseher reagiert auf unsere Interessen und unsere Gewohnheiten. Die Liste hört nicht auf. Was diese modernen eingebetteten Systeme oft gemeinsam haben ist die Möglichkeit mit dem Menschen über einem der für uns einfachsten Mitteln - die Sprache - zu kommunizieren. Was uns leichter fällt, ist aus Rechnersicht nicht unbedingt so einfach. Bei dem heutigen Stand der Technik nutzen Smart-Geräte fast ausschließlich Cloud-Dienste, um ihre Spracherkennung tatsächlich durchzuführen, da immer noch eine Rechenleistung dafür benötigt wird, die aus technischer oder finanzieller Sicht unpraktisch ist komplett in einem Offline-Gerät integriert zu werden. Da eine gute Internet-Verbindung fast überall eine Selbstverständlichkeit ist, funktionieren auch die kommerziell angebotenen Lösungen wie Amazon-Alexa oder Google-Assistant schnell und nahezu ohne Fehler[1]. Damit eröffnen sich viele neue Möglichkeiten für den Menschen mit seiner Umgebung zu interagieren.

Allerdings ist diese Lebensqualitätsverbesserung mit einigen Nachteilen und Gefahren verbunden. Da solche Geräte bei dem Benutzer zu Hause gestellt werden, wird die Privatsphäre bei ständig aufnehmenden und mit dem Internet verbundenen Geräten nicht mehr geschützt wie früher. Die manchmal angebotene Mikrofon-Ausschalttaste steht praktisch im Widerspruch mit dem Komfort-Versprechen der Sprachsteuerung und obwohl der Mehrheit der Benutzer die Problematik bekannt ist, tun nur wenige etwas dagegen[2].

Auch wenn es noch keinen Grund dafür gibt, die Anbieter in Verdacht auf Verstößen zu halten, bleibt die Tatsache, dass praktisch ohne Ausnahme alle kommerziellen Spracherkennungs-Geräte mit proprietärer Software arbeiten. Somit wird der Benutzer gezwungen dem Anbieter sein privates Leben völlig zu anzuvertrauen. Bugs in der Software oder Angriffe von dritter Seite sind leider nicht nur möglich, sondern auch in mehren Fällen tatsächlich passiert, wie wir weiter unten sehen werden.

Das Ziel dieser Arbeit ist ein Kompromiss zwischen der Gemütlichkeit bei der Steuerung der zahlreichen intelligenten Geräten zu Hause, die Kommunikation mit Cloud-basierten Diensten und die gleichzeitige Erhaltung der Privatsphäre zu finden. Dazu muss ein Weg gefunden werden die Aufgaben der Spracherkennung

[1] Vgl. Trojan (2019), S.12
[2] Vgl. Code Computerlove Ltd (2019), Internetquelle

zu trennen und nur das, was wirklich nötig ist, in das Internet zu schicken, und dabei nur mit dem klarem Bewusstsein des Benutzers.

Als erstes müssen die Anforderungen an so einem hybriden Assistenten geklärt werden. Für den Aufgabenbereich der Gerätesteuerung sind offline-Lösungen gut genug. Kompliziertere Aufgaben, wie z.B. eine freie Unterhaltung mit dem Assistenten, sind ohne Cloud-Diensten immer noch undenkbar.

Es müssen auch leider andere Kompromisse gemacht werden. Ein Gleichgewicht ist zwischen Qualität der Spracherkennung, die Benutzerfreundlichkeit und die Komplexität des Projekts zu finden. Geeignete Hardware und Software-Komponenten müssen ausgewählt und an einander angepasst werden. Die richtige Funktion ist mit realen Geräten zu beweisen.

Aus Transparenz- und Vertrauensgründen wird der vorgestellte Assistent ausschließlich Open-Source Software verwenden.

2 Grundlagen

2.1 Von der Akustik zur Spracherkennung

Die Hauptform der Kommunikation beim Menschen ist die natürliche Sprache[1]. Wenn diese Aufgabe unseren stark optimierten Gehirnen leicht fällt, ist es nur in der modernen Zeit dazu gekommen, dass Maschinen vergleichbar gute Ergebnisse bei der Sprachkommunikation liefern[2].

Die Sprachkommunikation basiert auf akustischen Wellen, die unsere Körpermechanismen im Hals - der s.g. Sprechapparat - erzeugen[3]. Variationen im Klang liegen nicht nur an den unterschiedlichen Sprachen, sondern auch an dem jeweiligen Menschen der redet, an seinem physikalischen und psychischen Zustand[4], an der Entfernung zum Aufnahmegerät[5] und an den Gegebenheiten im Raum oder in der Umgebung[6]. Nicht zuletzt haben die potenziellen Hintergrundgeräusche einen großen Einfluss auf die Aufnahme und stellen dadurch ein massives Problem für die korrekte Interpretation der Sprache dar[7].

Bei der Aufnahme der akustischen Wellen werden sie in elektrischen Signalen umgewandelt, die zuerst als direkte Variationen der Spannung abgebildet wer-

[1] Juang; Rabiner (2005), S.2
[2] Vgl. Pfister; Kaufmann (2017), S.21
[3] Vgl. Pfister; Kaufmann (2017), S.12
[4] Vgl. Pfister; Kaufmann (2017), S.328
[5] Vgl. RoboTech srl (2019a), S.14
[6] Vgl. Stotz (2019), S.16pp
[7] Vgl. Deng; Huang (2004), S.71

den. Weil Rechner mit digitalen Signalen arbeiten, erfolgt danach eine Analog-Digital-Umsetzung, um aus der analogen Kurve eine Menge von diskreten Werten abzuleiten[1].

In dieser digitalen Form fängt die tatsächliche Spracherkennung an. Als ersten Schritt findet die Erkennung der Lauten, Pausen und Wörtern aus den aufgezeichneten Daten statt, woraus danach die konkreten Wörter konstruiert werden[2].

Um die Komplexität der Spracherkennung zu minimieren, wird die Spracherkennungssoftware für spezielle Anwendungsfälle oder Szenarien konzipiert, wie zum Beispiel[3]:

- Spracherkennung nur für einzeln gesprochene Wörter;
- Spracherkennung bei überschaubarem Vokabular;
- Sprecherabhängige Spracherkennung;
- Spracherkennung nur für Telefonsignale.

Die beiden Hauptansätze bei der Spracherkennung sind der Mustervergleich und die statistische Spracherkennung[4]:

- Bei dem Mustervergleich, wie der Name vermutet, sind dem System schon gesprochene Muster von verschiedensten Wörtern bekannt. Es erfolgt ein Vergleich mit dem aufgenommenem Wort;
- Bei der statistischen Spracherkennung verfügt das System über eine statistische Beschreibung für die akustische Realisierung von jedem Wort des Vokabulars.

Zusätzlich zu den akustischen Modellen werden auch bei den modernen Systemen Sprachmodelle eingesetzt, die basierend auf Erfahrungswerten die Wahrscheinlichkeit schätzen, ob das vermutlich erkannte Wort in dem Kontext auftauchen könnte[5].

Weil die Sammlung der nötigen Daten und die Aufbereitung dieser Modelle sehr aufwendig ist, hat dies zur Folge geführt, dass die verfügbare Spracherkennungs-Software meistens mit gesperrtem Quellencode arbeitet, was die Entwicklung neuer Software und die Interoperabilität zwischen der vorhandenen Software hindert[6].

[1] Vgl. Pfister; Kaufmann (2017), S.39
[2] Vgl. Pfister; Kaufmann (2017), S.201-202
[3] Pfister; Kaufmann (2017), S.28
[4] Vgl. Pfister; Kaufmann (2017), 327-328
[5] Vgl. Pfister; Kaufmann (2017), S.411-412
[6] Vgl. VoxForge (2021), Internetquelle

2.2 Gebäudeautomatisierung mit Smart-Home Geräten

Intelligente Geräte, die bestimmte Aufgaben ausführen und dafür in vielen Fällen dauerhaft mit dem Internet verbunden sein müssen, werden Smart-Geräte genannt[1]. Die Benutzung solcher Geräte ist sehr populär, wie z.B. der Durchschnitt von zehn Geräten pro Haushalt in Deutschland zeigt[2]. Diese Tendenz ist steigend, zur Zeit zusätzlich angetrieben durch die Lockdown-Gegebenheiten[3]. Etwa ein Drittel dieser intelligenten Geräte sind Smart-Lautsprecher und der Rest sind eingebettete Systeme[4], wovon in Deutschland größtenteils intelligente Lampen und Wetterstationen vertreten sind[5]. Die intelligenten Lautsprecher verfügen auch über ein Mikrofon und Spracherkennung und in diesem Bereich hat sich Amazon mit dem Alexa-Assistenten als Marktherrscher etabliert[6].

Wie wir weiter unten sehen werden ist es typisch, dass die intelligenten Geräte dem Smart-Assistenten über einer Smartphone-App oder einer Web-Schnittstelle bekannt gemacht werden, damit sie nachher gesteuert werden können. Dabei ist die Aufgabe deren Steuerung gar nicht die Hauptrolle des Assistenten, der hauptsächlich nach Musik oder nach einem Wetterbericht gefragt wird[7].

2.3 Smart-Home Geräte Steuerung mit OpenHAB

Die verschiedenen Smart-Geräte aus dem Markt nutzen alle verschiedene Technologien für die Kommunikation und Steuerung. Ein Blick auf dem aktuellem Angebot offenbart ein ganzer Mix von Varianten für das Kommunikationsmedium wie Wireless, ZigBee und Bluetooth. Für die Inbetriebnahme ist fast ausschließlich eine Smartphone-App vom Hersteller notwendig, und für die nachfolgende Steuerung unterscheiden sich auch die Ansätze von einander.

Wenn die Steuerung über Sprachbefehlen mittels einem Smart-Assistenten wie Alexa erfolgt, dann ist der Standardweg wie folgt[8]:

- Sicherstellen, dass das Gerät mit Alexa kompatibel ist;
- Das Gerät mithilfe der Hersteller-App mit dem Heimnetzwerk verbinden und in Betrieb nehmen;

[1] Vgl. Xiaomi Inc. (2021), Internetquelle
[2] S.C. BITDEFENDER S.R.L. (2016), S.3
[3] Vgl. Xiaomi Inc. (2021), Internetquelle
[4] Vgl. Strategy Analytics Ltd. (2020), Internetquelle
[5] Conrad Connect GmbH (2019), Internetquelle
[6] Vgl. Code Computerlove Ltd (2019), Internetquelle
[7] Ebd.
[8] https://www.amazon.com/gp/help/customer/display.html?nodeId=201749240, (Zugriff am 17.02.2021)

- Über die Alexa-App das neue Gerät suchen und integrieren, was meistens über den sogenannten Skills erfolgt.

Die Skills sind Software-Erweiterungen für Alexa, die neue Funktionalitäten, oder Integration mit Smart-Geräten anbieten[1]. Es könnte auch sein, dass manche Geräte sogar ohne Skills direkt mit Alexa kommunizieren können[2].

Bei diesem Vorgehen könnte die Verfügbarkeit von Skills, oder die für sie benötigte dauerhafte Internet-Verbindung ein Problem darstellen. Eine Alternative, zahlreiche verschiedene Smart-Geräte lokal zu verbinden und zu steuern, bietet die Software OpenHAB an[3]. Sie soll als eine Abstraktionsschicht für die Integration und Steuerung von mehr als 200 verschiedenen Technologien und Systemen, sowie tausenden von Geräten dienen[4]. Die ganze Funktion ist ohne Internet-Verbindung vorhanden, es kann aber zusätzlich mit Smart-Assistenten wie Alexa kommuniziert werden.

OpenHAB läuft typischerweise auf einem kleinen Rechner oder Einplatinenrechner wie der Raspberry Pi und wird über einer grafischen Oberfläche im Browser gesteuert. Für die Anbindung von Geräten werden die s.g. „bindings“ aktiviert, welche die konkrete elektrische Kommunikation mit dem Gerät ermöglichen. In der Benutzeroberfläche werden die so erkannten Geräten als „things“ bezeichnet. Die Verknüpfung zwischen einem Ding und einer Bindung ist „channel“ genannt. Zusätzliche Begriffe und detaillierte Information über der Installation und Nutzung der vielen unterstützten Geräten sind in der Dokumentation zu finden[5].

2.4 Offline Spracherkennung mit EasyVR

Auch wenn die Cloud-basierten Assistenten wie Alexa ihre Funktion nahezu perfekt ausführen, ist ihre dauerhafte Internet-Verbindung ein potenzielles Problem, wie weiter unten diskutiert wird. Eine mögliche Lösung mit etwas Kompromiss an der Spracherkennungsqualität kann über eine offline-Lösung gefunden werden.

Der in dieser Arbeit ausgewählte Satz aus Hardware und Software - EasyVR von Fortebit - bietet offline-Spracherkennung in einer sehr kleinen Bauform an[6]. Die Grundversion unterstützt bis zu 256 Sprecher-abhängigen Befehlen, die praktisch in jeder Sprache trainiert werden können. Ein begrenzter Satz aus 26

[1] https://www.amazon.com/Alexa-Skills-Getting-Started-Guide/b?node=15144553011, (Zugriff am 17.02.2021)

[2] https://www.amazon.com/gp/help/customer/display.html?nodeId=G7UUWR5CRDLW8ZUR, (Zugriff am 17.02.2021)

[3] https://www.openhab.org/, (Zugriff am 17.02.2021)

[4] Ebd.

[5] openHAB Foundation e.V. (2020), Internetquelle

[6] https://fortebit.tech/easyvr-3-plus/, (Zugriff am 17.02.2021)

Sprecher-unabhängigen Befehlen in sechs Sprachen, darunter auch Deutsch, ist bereits vorinstalliert. Für das Training und die Benutzung von weiteren Sprecher-unabhängigen Befehlen wird zusätzliche Software kostenpflichtig angeboten. EasyVR kommuniziert und wird gesteuert über einem beliebigem UART 3,3 oder 5 Volt Host-Board. Diese können zum Beispiel der Arduino oder der Raspberry-Pi sein, wobei der Hersteller ein Schild für die leichte Integration mit Arduino-Uno anbietet.

Für die Installation unter Windows ist ein Treiber angeboten. Das Training erfolgt mithilfe der beigefügten Software EasyVR-Commander über einem seriellem Adapterkabel. Nachdem die Befehle auf dem Board programmiert sind, kann es mithilfe vom Schild auf dem Arduino gesteckt werden. Weitere Kommunikation erfolgt über der Arduino-Software.

2.5 Die „Alexa“ Spracherkennung-Software

Amazon bietet externen Firmen kostenlos die Möglichkeit an, Alexa in ihren Produkten zu integrieren[1]. Die Software, in der Form eines SDK - Software Development Kit – besteht aus mehreren Open-Source C++ Bibliotheken und ist auf GitHub verfügbar[2]. Eine Einschränkung ist, dass keine Prime-Musik auf dem Gerät gespielt werden kann, aufgrund von Lizenzbeschränkungen[3]. Um diese Funktion freizuschalten muss eine zusätzliche Genehmigung bei Amazon beantragt werden, die sich ausschließlich auf kommerziellen Projekten bezieht[4].

Der SDK ist modular und besteht aus Komponenten, die sich um alle nötigen Funktionen kümmern. Darunter sind die Aufnahme und Bearbeitung von Audio, die Verbindung zu den Amazon-Cloud Servern und die Interaktion mit dem Benutzer. Eine Beispiel-Software-Applikation mit allen Grundfunktionen - die „Sample-App“ - ist auch vorhanden, um den SDK zu testen und kennenzulernen[5]. Diese App wird als Ausgangspunkt in der vorliegenden Arbeit benutzt, nachfolgend angepasst und wird die Funktion der Online-Komponente bei dem hybriden Assistenten übernehmen. Nach dem Zweck der Arbeit wird die Aktivierungswort-Komponente nicht benutzt, sondern Alexa über einem Signal aus EasyVR aktiviert, wie weiter unten in Einzelheiten diskutiert wird.

[1] https://developer.amazon.com/en-US/alexa/devices/alexa-built-in, (Zugriff am 17.02.2021)

[2] Amazon Inc. (2020), Internetquelle

[3] https://github.com/alexa/avs-device-sdk/issues/556, (Zugriff am 17.02.2021)

[4] https://developer.amazon.com/support/legal/alexa/alexa-voice-service/terms-and-agreements#amazon-music-requirements, (Zugriff am 17.02.2021)

[5] https://developer.amazon.com/en-US/docs/alexa/avs-device-sdk/overview.html, (Zugriff am 17.02.2021)

Die benutzte Version des SDK ist 1.21.0, vom 26. Oktober 2020.

3 Anforderungen am virtuellen Assistenten

Wie schon im vorigem Kapitel betrachtet, werden die virtuellen Assistenten grundsätzlich für folgende Gruppen von Aufgaben eingesetzt[1] [2]:

- Die Steuerung von Geräten zu Hause oder die Abfrage derer Daten;
- Hören von Musik;
- Abfragen von Daten aus einer Suchmaschine;
- Freie Kommunikation mit dem Assistenten.

Von diesen Aufgaben bezieht sich die erste Gruppe auf Kommunikation mit lokalen Geräten und zeigt sich dadurch als ein hervorragender Kandidat für Offline-Steuerung. Die nächsten drei Gruppen sind stark von Online-Diensten abhängig und eine Verbindung mit dem Cloud ist aus diesem Grund notwendig.

Für die hier gestellten Zwecke ergibt es wenig Sinn, wenn die lokalen Geräten direkt angesteuert werden, aber die Spracherkennung über Cloud-Dienste erfolgt. Darum betrachten wir die Steuerung von lokalen Geräten zusammen mit einer Offline-Spracherkennung.

Andersherum ist es sinnvoll für Online Aufgaben auch die Online-Dienste zu verwenden.

Beide Varianten haben ihre Vor- und Nachteile. Die direkt erkennbaren Vorteile sind der bessere Schutz der Privatsphäre bei Offline-Spracherkennung und die allgemein bessere Qualität der Spracherkennung bei Online-Diensten. Betrachten wir im Folgendem die Nachteile.

3.1 Probleme bei Cloud-basierter Spracherkennung

Der Fokus in den folgenden Betrachtungen ist auf dem Marktherrscher Alexa von Amazon. Da die anderen Anbieter auch Cloud-Dienste nutzen, können ähnliche Überlegungen gemacht werden.

Als erstes steht die Voraussetzung für eine gute und stabile Internetverbindung. Wenn diese nicht vorhanden ist, kann auch die Spracherkennung nicht funktionieren[3].

[1] Vgl. Strategy Analytics Ltd. (2020), Internetquelle

[2] Vgl. Code Computerlove Ltd (2019), Internetquelle

[3] http://website-g7g.amazon.co.uk/gp/help/customer/display.html/ref=hp_left_v4_sib/259-3618452-2555505?ie=UTF8&nodeId=GVP69FUJ48X9DK8V, (Zugriff am 17.02.2021)

Dann kommt das Problem mit dem Aktivierungswort - das Gerät muss dauerhaft zuhören und das Gehörte analysieren, um das Aktivierungswort zu erkennen und zügig darauf zu reagieren. Das ist eine komplexe Aufgabe und es kann in manchen Fällen vorkommen, dass „Alexa ein anderes Wort oder ein anderes akustisches Signal als Aktivierungswort interpretiert“[1]. In einer solchen Situation ist das Gerät aktiviert, hört zu und schickt die Aufnahme in die Cloud, allerdings ohne dass der Benutzer es mitbekommt. Amazon bietet die Option an, dass „immer, wenn Audiodaten in die Cloud geleitet werden, ein kurzes akustisches Signal ertönt“[2], diese Option ist aber standardmäßig nicht aktiviert. Es stellt sich die Frage, wieviele Benutzer davon wissen und wieviele dieses Angebot in Anspruch nehmen würden.

Somit kommt es zu dem nächstem potenziellem Problem nämlich was mit der Audioaufnahme in der Cloud passiert. Nach Angaben von Amazon werden die Daten neben der Hauptaufgabe für die tatsächliche Spracherkennung auch zum Trainieren und Verbessern der Systeme verwendet[3]. Es ist die Aufgabe von Amazon diese Daten zu schützen und den Zugriff darauf zu kontrollieren.

Durch Missverständnissen kann es auch zu Problemen kommen. In einem Fall hat eine ganze Serie von falschen Interpretationen seitens Alexa zur Folge gehabt, dass ein ganzes privates Gespräch einer Familie ohne deren Erkenntnis aufgenommen wurde und an einer zufälliger Person aus der Kontaktliste geschickt wurde[4]. Nach der Aussage von Amazon war das eine unwahrscheinliche Folge von Ereignissen, und es wird daran gearbeitet solche Fehler noch unwahrscheinlicher zu machen[5].

Trotz der Bemühungen seitens Amazon ihre Geräte gegen Dritten zu sichern, sind leider schon Präzedenzfälle bekannt. Als Fehler seitens Amazon ist der Fall bekannt, indem ein Deutscher die akustischen Aufnahmen von einer Familie aus der USA bekommen hat[6].

Die letzten Fälle können als Missverständnisse oder Fehler seitens dem Hersteller klassifiziert werden. Offen ist die Frage, ob die Hersteller bewusst in die Privatsphäre eingreifen, ohne dies explizit anzugeben. Dazu liegen zur Zeit keine Beweise vor. Es bleibt aber die Tatsache, dass aufgrund der proprietären Hard– und Software der Assistenten und der Cloud-Dienste, einen hohes Maß an Vertrauen zum Hersteller gefordert wird. Die Funktion der Geräte kann allerdings in

[1] Ebd.
[2] Ebd.
[3] Ebd.
[4] Horcher (2018), Internetquelle
[5] Ebd.
[6] Bleich (2019), Internetquelle

gewissen Massen auch ohne Kenntnis ihrer Quellencode und Hardware-Schaltung getestet werden.

Zum Beispiel hat der Entwickler „electronupdate“ auf seiner Seite ein Tear-Down veröffentlicht, indem er feststellt, dass die Ausschalttaste für das Mikrofon von Echo Flex auch tatsächlich funktioniert[1].

Die Internet-Verbindungen, die Alexa aufbaut, wurden auch analysiert[2]. Auch wenn es unmöglich ist, die Inhalte zu lesen, aufgrund der Verschlüsselung, hat der Autor keine Auffälligkeiten oder einen Grund für Verdacht festgestellt.

Am Ende sind noch die potenziellen Probleme zu erwähnen, die aus der dauerhaften Verbindung mit dem Internet resultieren und die damit möglichen Hackerangriffen. Als Beispiel haben Forscher Probleme bei Cross-Origin Resource Sharing und dadurch die Möglichkeit zu Cross Site Scripting entdeckt[3].

3.2 Einschränkungen bei Offline-Spracherkennung

Die korrekte Funktion einer Spracherkennung hängt direkt von dem Umfang und der Qualität der Sprach-Daten, womit das Modell trainiert wurde, ab. Mit dem Fortschritt der Wissenschaft werden die resultierenden Modelle immer kleiner und die benötigten Hardware-Ressourcen immer weniger, allerdings sind leistungsvolle Cloud-Server immernoch das Standard bei der Spracherkennung[4]. Die Folge davon ist eine wesentlich schlechtere Leistung bei den Offline-Lösungen.

Manche aktuelle Offline-Ansätze versprechen vergleichbar gute Leistung wie Online-Dienste, diese wurden aber in dieser Arbeit nicht getestet. Als Hindernis für deren Nutzung ist der Preis oder die einschränkende Lizenz, wie der Fall bei PicoVoice ist[5], oder die Verfügbarkeit nur auf bestimmte Hersteller-gebundene Hardware, wie das bei Google-Pixel ist[6].

[1] electronupdate (2021), Internetquelle
[2] Trojan (2019), S. 31
[3] Barda et al. (2020), Internetquelle
[4] McGraw et al. (2016), S.5955
[5] Picovoice Inc. (2021), Internetquelle
[6] Coldewey (2019), Internetquelle

4 Konzept und Zweck des hybriden virtuellen Assistenten

4.1 Konzept des hybriden Assistenten

Um die erwünschten Anforderungen zu erfüllen, schlagen wir eine Kombination aus On- und Offline Spracherkennung vor, wobei die Offline-Hardware die Hauptrolle bei der Steuerung übernimmt. Die Offline-Komponente wartet durchgehend darauf das Aktivierungswort zu hören, und reagiert danach auf vordefinierten Befehlen. Die Befehle beinhalten solche für die direkte lokale Steuerung von den verfügbaren Smart-Home Geräten, sowie einem Befehl für die Aktivierung der Alexa-Software. Die Alexa-Software befindet sich auf einer separaten Hardware-Komponente und die Kommunikation erfolgt über Hardware.

Der Vorteil dabei ist die komplette Trennung der Funktionen auf die sprachgesteuerten lokalen Aktivitäten und die Benutzung von Alexa-Diensten bei einer erhöhten Benutzeraufmerksamkeit. Nachteilig ist der Bedarf an etwas mehr Hardware, darunter zwei Mikrofonen-Sätze und zwei Lautsprecher.

Bei dem vorgeschlagenem Konzept wird die Alexa-Software nur aktiviert, wenn der Benutzer das Kommando „Alexa“ gibt. Da dieses Kommando erst nach dem Aktivierungswort für die Offline-Platine kommen darf, minimiert sich das Risiko von Missverständnissen. Die Nutzung eines zusätzlichen Befehls, um Alexa zu starten, kann andererseits als störend vom Benutzer empfunden werden.

4.2 Schutz der Privatsphäre

Mit dem hybriden Assistenten kann sichergestellt werden, dass das Internetverbundene Mikrofon nur dann zuhört, wenn die Alexa-Software explizit angesprochen wurde. Die Mikrofonausschalttaste, die bei den kommerziellen Geräten als optionale Funktionalität angeboten wird, ist in dem hybriden Design praktisch enthalten. Dabei ohne den Bedarf die Hardware per Hand zu bedienen um das Mikrofon auszuschalten.

Die Befehle, die der Ansteuerung der Hausgeräte dienen, werden komplett ohne Cloud-Diensten erledigt. Zwischenmenschliche Gespräche werden garnicht aufgenommen und verlassen die eigenen Wände nicht.

4.3 Sicherheitsaspekte

Ein Teil des hybriden Assistenten ist die Online-Komponente. Diese ist praktisch ein Allzweck-Rechner, der dauerhaft mit dem Internet verbunden ist. Vom Benutzer wird erwartet, dass er selber dafür Aktualisierungen installiert, was in dem Fall mit kommerziellen Alexa-Geräten automatisch ausgeführt wird. Die Problematik ob ein Linux-basierter Allzweck-Computer stärker von Hackerangriffe gefährdet ist als ein kommerzielles eingebettetes System wie z.B. Amazon-Echo, benötigt weitere Betrachtung und wird in dieser Arbeit nicht diskutiert.

Die Gefahr durch einen physikalischen Zugang ändert sich nicht bei dem hybriden Assistenten und bleibt wie bei anderen Computer-Systemen hoch.

4.4 Auswahl der Hardware-Komponenten

Raspberry-Pi

Die zwei Haupt-Softwares, die nach dem Konzept des Assistenten auf dem Raspberry-Pi benötigt werden, sind der Alexa-SDK und OpenHAB. OpenHAB kümmert sich um die direkte Ansteuerung der Hausgeräte und die Alexa-Software kommuniziert mit dem Amazon-Cloud, wenn es vom Benutzer erwünscht ist.

Alexa-SDK erfordert den Raspberry-Pi 3 oder 4[1]. OpenHAB benötigt sogar weniger Ressourcen und funktioniert mit dem Raspberry-Pi Version 2 oder besser[2]. Aus Platz- und Kostengründen haben wir den Pi-Zero W[3] ausgewählt.

EasyVR und Arduino UNO

EasyVR wird mit installierter proprietären Firmware und einem unidirektionalem Mikrofon Horn EM9745P-382[4] geliefert. Optional bestellbar ist ein Shield für eine einfache Hardware-Verbindung mit einem Arduino-Board.

Das EasyVR-Modul ist nicht konzipiert für einen eigenständigen Betrieb, sondern benötigt einen Host-Rechner, der typisch ein Mikrocontroller oder Einplatinencomputer ist. Generell eignet sich dabei jedes System, das seriell kommunizieren kann und das EasyVR-Protokoll spricht[5].

Es ist dadurch möglich direkt mit dem Raspberry-Pi über eine der folgenden Optionen zu kommunizieren:

[1] Amazon Inc. (2020), Internetquelle
[2] https://www.openhab.org/docs/installation/rasppi.html, (Zugriff am 20.02.2021)
[3] https://www.raspberrypi.org/products/raspberry-pi-zero-w/, (Zugriff am 20.02.2021)
[4] RoboTech srl (2019a), S.12
[5] RoboTech srl (2019a), S.28

- Die Python Software-Bibliothek benutzen[1];
- Die „EasyVR Embedded Lybrary“ benutzen, die einen Arduino-Kern implementiert. Diese ist als „experimentell“ und „offiziell nicht unterstützt“ gekennzeichnet[2];
- Eine eigene Bibliothek implementieren.

Weil der Arduino-Schield eine vergleichsweise einfache und schnelle Möglichkeit anbietet, den gezielten Prototyp zu bauen, wurde dieser zusammen mit der Arduino-Bibliothek für das vorliegende Projekt ausgewählt.

Das Arduino-Board übernimmt somit die direkte Steuerung und Kommunikation mit EasyVR und leitet die Benutzerwünsche zu Raspberry-Pi weiter.

Zusätzliche Hardware

Die Alexa-Software, die auf dem Raspberry-Pi installiert wird, benötigt außer Internet-Verbindung auch ein Mikrofon und einen Lautsprecher. Zusätzlich wäre es wünschenswert Lichtmeldungen wie bei den kommerziellen Alexa-Geräten erzeugen zu können. Nicht zuletzt muss auch an der Verbindung mit dem Arduino gedacht werden.

Eine Möglichkeit alle diese Anforderungen einfach zu erfüllen bietet ist die Raspberry-Pi-Hat von ReSpeaker. Diese kombiniert zwei Mikrofone, Hardwarebearbeitung des Klangs, einen verstärkten Audio-Ausgang, Hardware-Zugang zu zwei GPIO-Pins vom Raspberry-Pi, Hardware-Zugang zu den Pins für I2C-Kommunikation und sogar drei programmierbare LED-Lampen mit RGB-Farbraum[3]. Die Lampen sind APA102 vom Hersteller APA, sie lassen sich nacheinander verketten und sind über einem 2-Draht SPI Protokoll ansprechbar[4].

Da Arduino mit 5 Volt und Raspberry-Pi mit 3,3 Volt arbeitet[5] [6], ist für die Kommunikation über I2C eine Zwischenschaltung notwendig, um den Raspberry-Pi vor der Überspannung zu schützen. Ein mögliches Design, das bidirektional funktioniert und wenige Bauteile benötigt ist auf Abbildung 4 im Anhang dargestellt. Das Design basiert auf einem Konzept von NXP Semiconductors[7]. Da der

[1] RoboTech srl (2019c), Internetquelle
[2] RoboTech srl (2017), Internetquelle
[3] Seeed Technology Co. Ltd (2020), Internetquelle
[4] Vgl. APA Electronic co. LTD (2013), S.2
[5] https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/, (Zugriff am 20.02.2021)
[6] https://www.raspberrypi.org/documentation/hardware/raspberrypi/gpio/README.md, (Zugriff am 20.02.2021)
[7] NXP Semiconductors (2007), S.4

Pi-Zero den Mikrocontroller BCM2835 nutzt[1], ist der Standartzustand der GPIO-Pins *HIGH*[2]. Eine triviale Logik-Schaltung mit einem NPN-Transistor wird dazu dienen, die aktiv-HIGH Signale aus dem Arduino-Board in aktiv-LOW für den Pi umzuschalten. Die Schaltung ist auf Abbildung 5 im Anhang zu finden.

Die Steuerung der Smart-Geräte übernimmt die OpenHAB-Software, die auf dem Raspberry-Pi installiert wird. Für eine Kommunikation mit den angesteuerten Geräten über Bluetooth oder WLAN kann die vorhandene Hardware vom Raspberry-Pi benutzt werden und für Kommunikation über dem ZigBee-Protokoll wurde das ZigBee USB-Gateway ConBee II ausgewählt[3]. Auch wenn es nicht das billigste Modul auf dem Markt ist, hat ConBee eine gute Kompatibilität mit OpenHAB und Raspberry-Pi, vergleichbar bessere Reichweite und umfangreichere Software[4].

Smart-Geräte

Die Auswahl von Smart-Geräten zum Testen vom Konzept wurde von der aktuellen Popularität in den gängigen Verkaufskanälen beeinflusst und besteht aus folgenden Produkten:

- Shelly-1 WiFi Relay Switch von Allterco Robotics LTD, verbunden über WLAN[5];
- Kasa Smart-Steckdose von TP-Link, verbunden über WLAN[6] und
- Hue White & Color Ambiance LED Lampe von Philips, verbunden über ZigBee[7].

Übersicht

Das gesamte Konzept des hybriden Sprachassistenten ist auf Abbildung 1 dargestellt. Die Einzelheiten der beteiligten Software werden weiter im folgenden Kapitel diskutiert.

[1] https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md, (Zugriff am 20.02.2021)
[2] Broadcom Corporation (2012), S.102
[3] Dresden Elektronik Ingenieurtechnik GmbH (2021), Internetquelle
[4] Ergebnisse nach einem eigenen Test-Vergleich mit einem CC2531 USB-Gateway
[5] Allterco Robotics LTD (2019), S.1
[6] TP-Link Technologies Co., Ltd (2020), S.6
[7] Signify GmbH (2021), Internetquelle

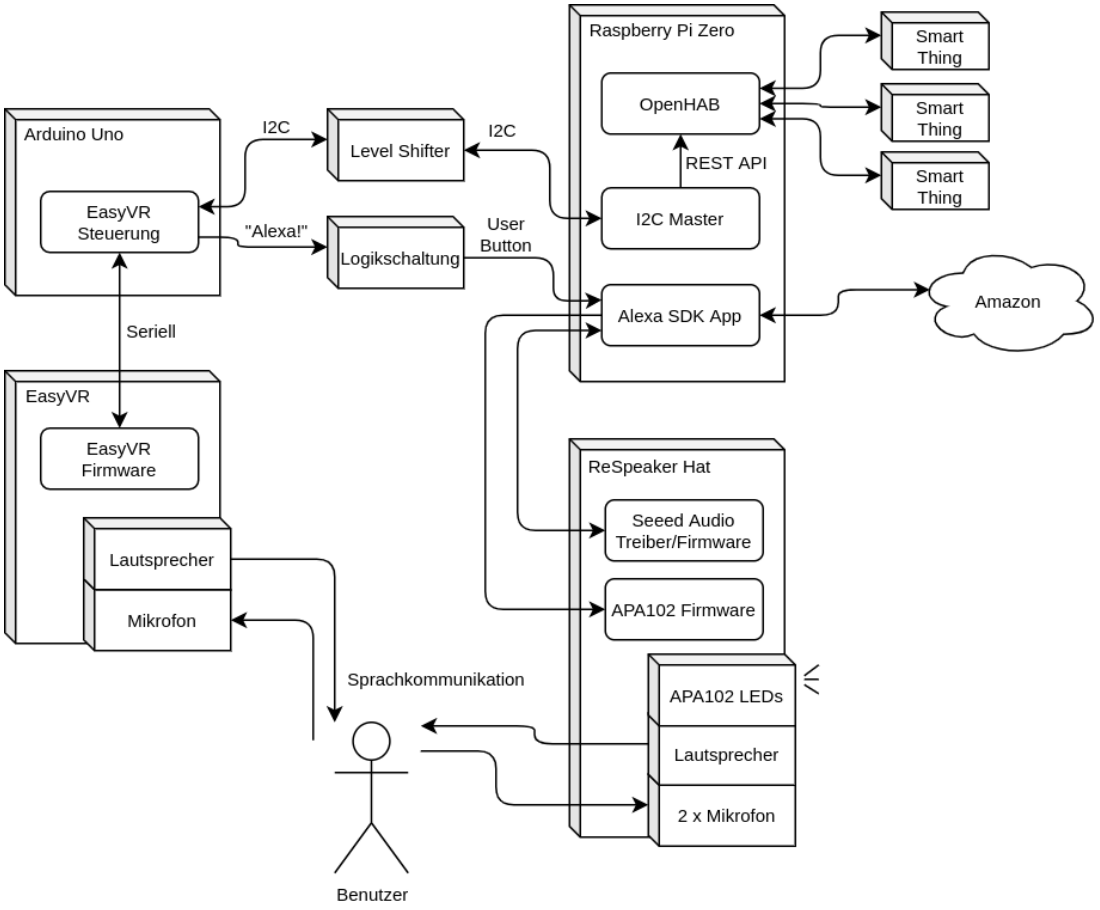

Abbildung 1: Aufbau- und Funktionsübersicht

5 Prototypenentwicklung

5.1 EasyVR: Inbetriebnahme, Training und Tests

Hardware

Die Programmierung und Einstellung vom EasyVR-Modul erfolgt über das mitgelieferte FTDI Kabel. Das Kabel ist praktisch ein Seriell-zu-USB Adapter[1] und braucht entsprechende Treiber, wenn die Installation unter dem Windows Betriebssystem durchgeführt wird. Die Fortebit-Webseite deutet dafür auf die Webseite ftdichip.com[2] hin. Die Installation der Treiber unter Windows erfolgte schnell und ohne Besonderheiten und danach wurde das Kabel als virtuellem COM-Port erkannt. Wie bei anderen Seriellen Geräten führt manchmal ein neues Einstecken zu einer anderen Port-Nummer, was aber ohne Probleme von der EasyVR-Software erkannt wurde.

[1] Vgl. RoboTech srl (2019a), S.85
[2] https://ftdichip.com/drivers/vcp-drivers/, (Zugriff am 20.02.2021)

Für die Klangausgabe muss ein kleiner 8-Ohm Lautsprecher verbunden werden[1]. Im Manual stehen keine Angaben zu der Leistung des Lautsprechers, diese lassen sich aber über dem Standardstromverbrauch für den Audio-Ausgang von 175mA berechnen[2]. Mithilfe vom Ohmschen Gesetz[3] und die Formel für Leistung[4] ergibt sich daraus für den benötigten Lautsprecher:

$$P = U.I = I^2.R = 0,175.8 = 1.4\,Watt$$

Software

Die Software für die Steuerung und Programmierung von EasyVR ist in einer Installationsdatei zusammengepackt. Es wird nur das Betriebssystem Windows unterstützt. Für diese Arbeit wurde die Software-Version 3.14.0.232 benutzt, die zum Zeitpunkt die aktuellste war. Die Datei ist nicht binär signiert und beinhaltet folgende Komponenten:

- „EasyVR Commander“ - das Hauptprogramm für die Programmierung und Steuerung von EasyVR[5];
- „Sensory QuickSynthesis“ - ein Tool für die Erstellung und Bearbeitung von eigenen Sound-Tabellen[6];
- „FluentChip“ - für die Erstellung von eigenen Grammatiken[7] und
- „QuickT2SI“ - eine Software für die Erstellung von Grammatiken, die Sprecherunabhängig sind. Für die Benutzung dieser Software ist eine extra Lizenz benötigt, die kostenpflichtig erworben werden muss[8].

Kurze Tests mit QuickSynthesis mithilfe von verschiedenen Sound-Dateien, Datenraten und Einstellungen haben keine überzeugende Leistung gezeigt. Auch andere, qualitativ höherwertige Lautsprecher erzeugen nur einen verzehrten Ton. Menschliche Sprache ist kaum erkennbar und insgesamt ist die Tonqualität sehr schlecht[9]. Fehlende Hardware für den Klang-Ausgang wie Verstärker könnten der

[1] RoboTech srl (2019a), S.14
[2] RoboTech srl (2019a), S.10
[3] Horowitz; Hill (2017), S.4
[4] Horowitz; Hill (2017), S.2
[5] RoboTech srl (2019a), S.69
[6] RoboTech srl (2019a), S.79
[7] keine Referenz im User Manual; Siehe Abbildung 6 im Anhang.
[8] RoboTech srl (2019a), S.80
[9] nach der persönlichen Einschätzung des Verfassers

Grund dafür sein. Der User Manual erwähnt die Möglichkeit der Anbindung eines externen Verstärkers um die Audio-Qualität zu verbessern, das wurde aber im Projekt nicht getestet. Der vorprogrammierte „Beep“ Ton wird für die Zwecke dieser Arbeit verwendet.

FluentChip und QuickT2SI wurden im Rahmen dieser Projektarbeit nicht getestet.

Aktivierungswort

Das voreingestellte Wort „Robot“ soll Sprecher-unabhängig sein. Ohne der S2TI-Software ist die Programmierung eines weiteren Aktivierungsworts erlaubt, allerdings Sprecher-abhängig[1].

Die Tests mit dem „Robot“ Aktivierungswort hatten keinen Erfolg. Bei den Standardeinstellungen wird das Modul bei vielen anderen Wörter oder Hintergrundgeräuschen, sogar bei Küchengeräuschen oder beim Türschließen aktiviert. Änderungen der Empfindlichkeit[2] haben die Situation auch nicht verbessert - entweder reagiert das Modul gar nicht, oder reagiert auf allen möglichen Geräuschen. Die Entfernung zum Sprecher hat auch keine Verbesserung gezeigt.

Aus diesem Grund wurde ein neues, Sprecher-abhängiges Aktivierungswort trainiert und dabei wurde die „normal“ Einstellung für Empfindlichkeit eingestellt. Mit dem Nachteil, dass nur ein einziger Sprecher somit erkannt werden kann, hat sich die Erkennungsrate wesentlich verbessert und Fehlermeldungen wurden so gut wie ausgeschlossen. Die Reaktion findet auch bei unterschiedlichen Entfernungen zum Sprecher statt, ohne die Einstellungen zu ändern, und somit hat sich dieser Ansatz als gut genug für die Zwecke dieser Arbeit erwiesen.

Befehle

Als ersten Ansatz wurden ausführlichere Befehle getestet, wie etwa „Steckdose-Eins an!“ oder „Lampe Küche an!“. Diese hat aber das Modul nicht akzeptiert und mehrere Befehle aus verschiedenen Kombinationen konnten nicht trainiert werden. Dabei wurde die Fehlermeldung „*Training error: recognition failed*“ angezeigt[3]. Kürzere Befehle, wie „Eins an!“ und „Licht an!“ ließen sich aber gut trainieren.

Im Endeffekt, für die zwei Steckdosen, die eine Hue-Lampe und die Ansteuerung von Alexa, wurden insgesamt neun Befehle trainiert. Darunter neben den

[1] Vgl. RoboTech srl (2019a), S.69
[2] RoboTech srl (2019a), S.74
[3] Siehe Abbildung 7 im Anhang

Standardbefehlen zum Ein- und Ausschalten wurden auch „heller“ und „dunkler“ für eine feinere Einstellung der Hue-Lampe, und einfach „Alexa“ für die Aktivierung der Alexa-Software hinzugefügt. Die Liste mit allen Befehlen ist auf Abbildung 8 im Anhang zu finden.

Zurücksetzung

Bei den verschiedenen Tests kann es dazu kommen, wie es in diesem Projekt passiert ist, dass das Modul nicht mehr auf Anfragen reagiert. In solchen Fällen wird von der Kundenbetreuung empfohlen die Firmware neu zu laden, um das Modul zurückzusetzen[1]. Dazu werden folgende Schritte benötigt:

- Der Jumper von der Arduino-Hat muss auf „UP“ umgestellt werden, wenn der Arduino als Programmiergerät benutzt wird. Das FTDI-Kabel darf nicht gleichzeitig mit dem Arduino genutzt werden, aufgrund von seriellen Kollisionen/Konflikten[2];
- Die Firmware ist ein Teil der mitgelieferten Software und kann im Installationspfad vom Commander gefunden werden, welcher standardmäßig *C:\Program Files (x86)\VeeaR\EasyVR Commander* ist. Die in diesem Projekt benutzte Firmware war die einzige Version für EasyVR-Plus in diesem Pfad, und nämlich die Datei *EasyVR3Plus_FW_Rev0.EVRFW*
- Das Menü für Installation/Upgrade von Firmware ist etwas ungünstig im Commander unter *Help/Update Firmware* platziert;
- Nach der Auswahl der Datei, Bestätigung und ein Klick auf *Download* startet das Laden.

5.2 EasyVR: Integration im Projekt

Nachdem das EasyVR-Modul eingestellt und trainiert wurde, bietet der Commander die Option an, einen Basiscode für den Arduino-Hostrechner zu erstellen, unter *File/Generate Code*. Dieser ist ein Arduino-Sketch, in wessen endloser „loop“ Schleife immer wieder nach neu-erkannten Befehlen geprüft wird. Die schon trainierten Wörter werden über ID angesprochen. Die Grundform einer „Action“ Funktion ist vorgelegt. In dieser soll die weitere Bearbeitung stattfinden.

[1] „Fortebit/ROBOTECH Technical Support“, E-Mail Korrespondenz vom 04.12.2020
[2] Ebd.

Betrachten wir die Integration im Sprachassistenten. Die Rolle vom Arduino-Board ist die anerkannten Befehle in geeigneter Art und Weise an dem Raspberry-Pi weiterzuleiten. Darunter zählen die Befehle für die Steuerung der Smart-Geräte und das Einschaltsignal für Alexa.

Für die Mitteilung von Steuerungsbefehlen wurde I2C als Kommunikationsprotokoll ausgewählt. Bei I2C existiert ein Master, der den Bus kontrolliert und einen oder mehrere Slaves, die nach Aufforderung vom Master ihre Daten melden oder Daten vom Master empfangen[1]. Weil die Initiative bei der Kommunikation in unserem Assistenten vom Arduino kommt, da die Daten dort zuerst generiert werden, ist es leicht vorstellbar den Arduino als Master einzustellen. Eine solche Konstellation war aber bei den Tests ohne Erfolg.

Der Raspberry kann nicht als Slave am I2C-Bus der Konfiguration teilnehmen, weil für diese Funktion andere Pins (18/19) benötigt werden[2], als die Pins (2/3), die von der ReSpeaker-Hat zur Verfügung gestellt sind[3]. Die benötigten Pins für den Slave-Modus sind schon intern für Kommunikation mit dem Sound-Modul in Gebrauch.

Somit bleibt nur die Option den Raspberry-Pi als I2C-Master und den Arduino als I2C-Slave zu verwenden. Weil die Daten auf dem Arduino entstehen, kann der Pi nicht wissen ob und wann diese vorliegen und muss deshalb ständig danach fragen - das s.g. Polling-Verfahren[4]. Der Arduino speichert die Daten von EasyVR und leitet diese nach Anfrage an dem Pi weiter.

Für die Kommunikation über I2C wird die Arduino-Bibliothek *Wire* benutzt. Diese definiert die benötigten Pins als A4 für SDA und A5 für SCL[5]. In der *setup*-Funktion wird die Teilnahme am Bus an der Adresse *address* mit *Wire.begin(address)* angekündigt[6] und mit *Wire.onRequest(function)* wird *function* als die Funktion definiert, die immer aufgerufen wird, wenn Daten am Bus anliegen[7].

Etwas mehr Aufmerksamkeit benötigt die Synchronisation zwischen der Hauptschleife und den Interrupts, die bei *onReceive* stattfinden, da diese in eigenen Threads verlaufen[8]. Wenn zwei Threads gleichzeitig auf gemeinsame Daten zugreifen wollen, kann es zu einer Wettlaufsituation kommen, indem die beteiligte

[1] Vgl. Horowitz; Hill (2017), S. 1034
[2] Broadcom Corporation (2012), S.102-103
[3] Seeed Studio (2017), S.1
[4] Vgl. Tanenbaum; Bos (2015), S.354
[5] https://www.arduino.cc/en/Reference/Wire, (Zugriff am 20.02.2021)
[6] https://www.arduino.cc/en/Reference/WireBegin, (Zugriff am 20.02.2021)
[7] https://www.arduino.cc/en/Reference/WireOnReceive, (Zugriff am 20.02.2021)
[8] https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/volatile/, (Zugriff am 20.02.2021)

Variable gleichzeitig geschrieben und gelesen wird, wobei die Ergebnisse nicht vorhersehbar sind[1]. Um diese Fehlkondition zu vermeiden, muss der Zugriff auf die gemeinsamen Daten kontrolliert und geschützt erfolgen. Es müssen dabei zwei Maßnahmen getroffen werden:

- Die gemeinsam-benutzte Variable darf nicht in dem CPU-Register geschrieben, sondern immer und nur aus dem RAM-Speicher abgerufen werden, damit jeder Thread mit der selben Datei arbeitet und nicht mit einer lokalen Kopie. Dieses Verhalten wird sichergestellt, wenn bei der Deklaration der Variable das Arduino-Schlüsselwort *volatile* benutzt wird[2];

- Die Threads, die auf die selbe Variable zugreifen wollen, müssen aufeinander warten. Dafür kann ein ganzer Code-Block als *ATOMIC_BLOCK* definiert werden. Während der Ausführung in diesem Abschnitt sind alle Interrupts ausgeschaltet und der Zugriff wird nur dann freigegeben, wenn kein anderer Thread den Block abarbeitet[3].

Es werden dabei zwei Variablen mit geschütztem Zugriff benötigt - eine boolesche Variable, die das Vorhandensein neuer Daten darstellt und eine *int*-Variable, die den Index des erkannten Wortes speichert.

Die Meldung eines „Alexa“-Aufrufs erfolgt nach unserem Konzept separat und direkt über Hardware. Für diesen Zweck wurde der Allzweck-Pin IO1 von EasyVR ausgewählt. Wenn das Kommando „Alexa!“ erkannt wird, wird für kurze Zeit ein Signal (5 Volt) auf dem Pin ausgegeben. Es ist die Aufgabe des Raspberry-Pi, dieses Signal aufzunehmen und zu interpretieren.

Eine Darstellung der Funktion vom Arduino-Sketch ist auf Abbildung 2 zu finden.

5.3 Raspberry Pi: Inbetriebnahme und Software

Raspberry-Pi OS basiert auf *Debian*[4] und nutzt *bash* als Shell[5]. Für die weitere Arbeit mit dem System werden neben der offiziellen Raspberry-Pi-Dokumentation auch die Handbücher und Referenzen von Debian[6] und *bash*[7] verwendet. Die

[1] Vgl. Tanenbaum; Bos (2015), S.119-121

[2] https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/volatile/, (Zugriff am 20.02.2021)

[3] Free Software Foundation, Inc. (2019), Internetquelle

[4] https://www.raspberrypi.org/documentation/raspbian/, (Zugriff am 20.02.2021)

[5] Vgl. https://www.raspberrypi.org/documentation/linux/usage/bashrc.md, (Zugriff am 20.02.2021)

[6] Software in the Public Interest, Inc. (2021), Internetquelle

[7] Ramey; Fox (2020)

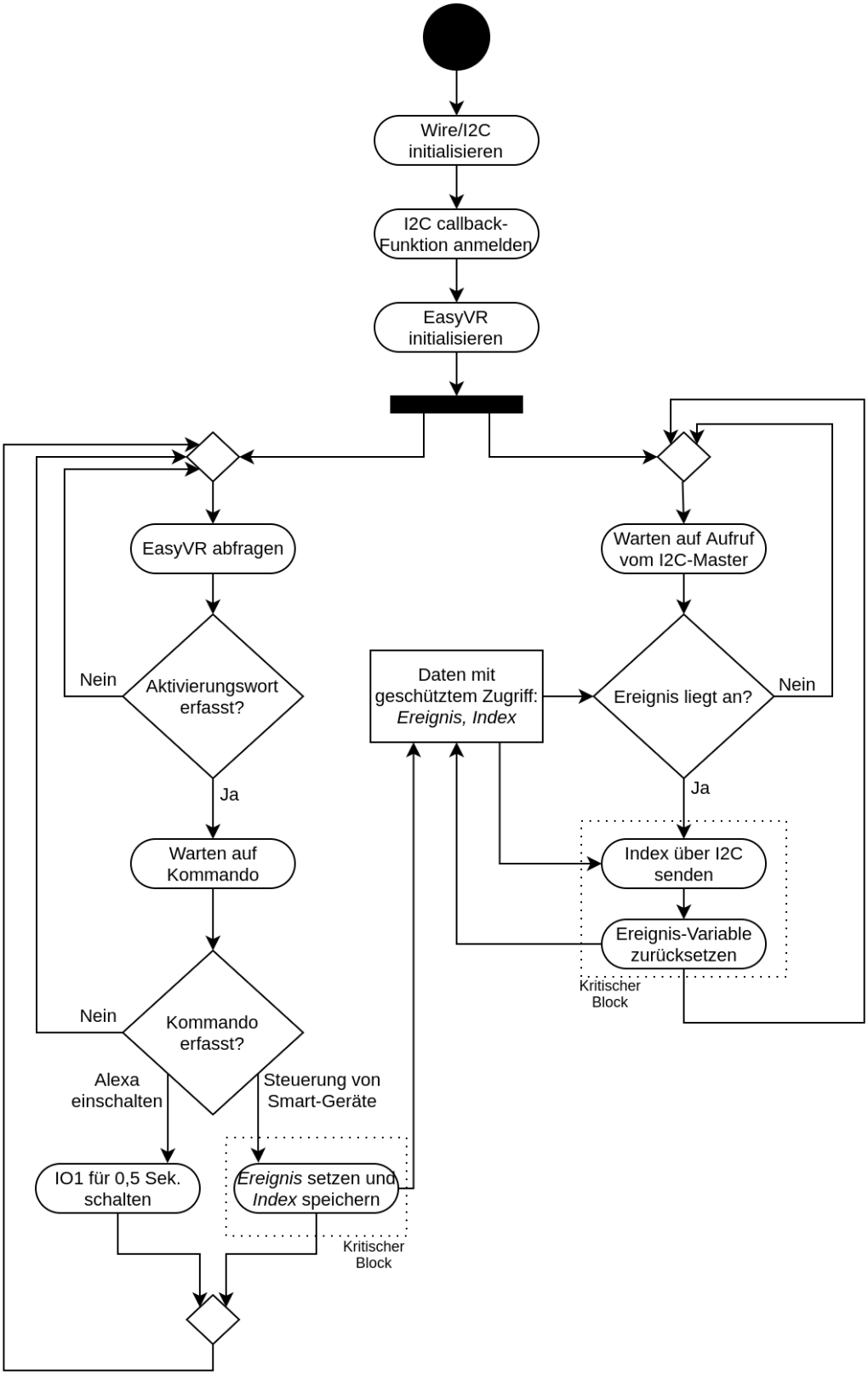

Abbildung 2: Arduino-Sketch für die Kontrolle von EasyVR

Kommunikation mit dem Pi führen wir aus einem Debian-Rechner aus.

Betriebssystem

Als Betriebssystem wird das von Raspberry-Pi empfohlene, *Raspberry-Pi OS* installiert[1]. Die letzte Version, zur Zeit 2020-08.20 laden wir von *https://www.raspberrypi.org/s systems/* herunter. Da für das Projekt keine Desktop-Umgebung oder andere Extra-Software in Gebrauch kommen, reicht uns die „*Lite*“ Version.

Nach der Anleitung[2], wurde die Datei auf einer SD-Karte übertragen mit dem Befehl:

```
sudo dd bs=4M if=2020-08-20-raspios-buster-armhf-lite.img of=/dev/sdf conv=
    fsync status=progress
```

Um die Arbeit mit dem Pi zu vereinfachen und auf direkt angeschlossenem Bildschirm und Tastatur zu verzichten, stellen wir den Pi in den Headless-Modus, durch die folgenden Maßnahmen[3]:

1. Einstellungen für die WLAN Verbindung in *wpa_supplicant.conf* eintragen[4]:

   ```
   ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
   update_config=1
   country=DE
   network={
       ssid="WLANSSID"
       psk="WLANPASSWORT"
       scan_ssid=1
   }
   ```

2. Änderungen in *config.txt*, die erlauben bei Bedarf/Fehlerkonditionen direkt über Seriell zu steuern[5]:

   ```
   # Enable UART
   enable_uart=1
   ```

[1] https://www.raspberrypi.org/documentation/raspbian/, (Zugriff am 20.02.2021)
[2] https://www.raspberrypi.org/documentation/installation/installing-images/linux.md, (Zugriff am 20.02.2021)
[3] Vgl. https://www.raspberrypi.org/documentation/configuration/wireless/headless.md, (Zugriff am 20.02.2021)
[4] Ebd.
[5] https://www.raspberrypi.org/documentation/configuration/uart.md, (Zugriff am 20.02.2021)

3. SSH aktivieren, durch Erstellung einer leeren Datei namens „*ssh*“[1].

Erster Start und Grundeinstellungen

Nach dem Einstecken der SD-Karte in den Pi und eine Verbindung mit dem gelieferten Netzteil wird das System gebootet und nach einiger Zeit kann man sich aus einem beliebigem Rechner im selben Netz einloggen mit:

```
ssh pi@raspberrypi.local
```

Damit wir nicht jedes Mal Passwort eingeben müssen, stellen wir die Verbindung mit SSH über einem Public Key ein. Wenn nur diese Art von Verbindung erlaubt ist, verbessert es die allgemeine Sicherheit des Systems[2].

```
# unser ID auf dem Pi übertragen:
ssh-copy-id pi@raspberrypi.local
# in Pi einloggen, jetzt kein Passwort mehr erfordelich:
ssh pi@raspberrypi.local
# mit einem Text-Editor die config-Datei von ssh bearbeiten:
vim.tiny /etc/ssh/sshd_config
# in dieser Datei die Passwort-Anmeldung auschalten mit
# PasswordAuthentication no
```

Die Datei speichern, *vim* verlassen und den SSH-Daemon neu starten:

```
sudo systemctl restart ssh
```

Die nächsten Schritte bei der neuen Installation sind über dem Skript *raspi-config* ausführbar[3]:

1. Neues Passwort einstellen;
2. Netzwerk/Hostname eingeben;
3. Unterstützung für die deutsche Sprache durch Auswahl von *de_DE.UTF-8* in *Localisation/Change Locale*;
4. Zeitzone einstellen als *Europe/Berlin* in *Localisation/Time Zone*;

[1] https://www.raspberrypi.org/documentation/remote-access/ssh/README.md, (Zugriff am 20.02.2021)

[2] https://www.raspberrypi.org/documentation/configuration/security.md, (Zugriff am 20.02.2021)

[3] https://www.raspberrypi.org/documentation/configuration/raspi-config.md, (Zugriff am 20.02.2021)

5. Das Dateisystem expandieren, damit die ganze SD-Karte benutzt wird in *Advanced/Expand Filesystem*;

6. Aktivieren von I2C unter *Interface Options/I2C*.

SWAP-Bereich

Aufgrund der sehr begrenzten Hardware-Ressourcen, insbesondere der RAM-Speicher, ließ sich bei den Tests die Alexa-Software nicht direkt auf dem Raspberry-Pi kompilieren. Um dieses Problem umzugehen, wurde bei dem Pi ein größerer SWAP-Bereich von 2 Gb angelegt. Die Standardinstallation bietet für die Einstellung von *SWAP* den Skript *dphys-swapfile* an[1]. Mit diesem Skript wurden folgende Schritte durchgeführt:

```
# Auschalten von SWAP:
sudo dphys-swapfile swapoff
# Neue Einstellung in der Konfigurationsdatei eingeben
# in dieser Datei: "CONF_SWAPSIZE=2048" einstellen
sudo vim.tiny /etc/dphys-swapfile
# Einstellungen übernehmen:
sudo dphys-swapfile setup
# SWAP wieder einschalten:
sudo dphys-swapfile swapon
# Neustart:
sudo reboot
```

Eine Prüfung der verfügbaren Speicher mit *free*[2] zeigt, dass *SWAP* tatsächlich eingeschaltet ist:

```
pi@pi-Z:~ $ free -m
              total used free shared buff/cache available
Mem: 432 30 339 3 62 351
Swap: 2047 0 2047
```

I2C

Für die Kommunikation über I2C verwenden wir die Python-Software smbus2 in der Version 0.4.1[3]. Die Installation erfolgte problemlos mit:

[1] Franklin (2006), Internetquelle
[2] https://manpages.debian.org/buster/procps/free.1.en.html, (Zugriff am 20.02.2021)
[3] Lindegaard (2021), Internetquelle

```
pip3 install smbus2
```

Wie schon in dem Kapitel für EasyVR erwähnt, muss der Pi über Polling nach Ergebnissen bei dem Arduino fragen. Ein Kompromiss zwischen Reaktionszeit und Rechnerbelastung haben wir mit einer Zeit von 0,5 Sekunden zwischen den Anfragen getroffen. Bei den Experimenten wird keiner der beteiligten Hardware überlastet und gleichzeitig ist die Reaktionszeit unbemerkbar. Der folgende Python-Skript bildet die Grundlage für die Kommunikation über I2C. Ein Abbruch ist durch Konsolenunterbrechung möglich (Ctrl-C).

Listing 1: I2C-Master Grundskript

```
import time
from smbus2 import SMBus

bus = SMBus(1)
done = False
print("Polling started. To quit use Ctrl-C")
while not done:
    try:
        b = bus.read_byte_data(8, 0)
        if b!=0:
            print(b)
        time.sleep(0.5)
    except KeyboardInterrupt:
        bus.close()
        done = True
print()
print("Bye")
```

Der Skript ist noch nicht nutzbar für das Projekt, weil er einfach die empfangenen Daten in die Konsole ausgibt. Bei der endgültigen Version wurden folgende Erweiterungen durchgeführt:

- Die empfangenen Daten sind die Befehle die an *OpenHAB* weitergeleitet werden müssen. Diese Weiterleitung erfolgt über einfache HTTP-Requests, wofür die Python-Bibliothek *request* importiert wurde. Die Requests sind nach dem Modell *http://localhost:8080/rest/items/NAME/BEFEHL* aufgebaut[1];
- Die empfangenen Daten bestehen schließlich aus einem *int*-Index, weshalb auch deren Bedeutungen vom EasyVR-Code übernommen wurden;

[1]https://www.openhab.org/docs/configuration/restdocs.html, (Zugriff am 20.02.2021)

- Bei den Tests ist es manchmal dazu gekommen, dass OpenHAB etwas mehr Zeit braucht, um betriebsbereit zu werden. Wenn der Skript genau zu diesem Zeitpunkt versucht einen HTTP-Request zu senden, bricht die Ausführung mit einer Fehlermeldung ab. Aus diesem Grund ignorieren wir alle HTTP-Fehlermeldungen und versuchen es erneut;

- Weil bei dem aufgebautem Prototyp die Hardware-Verbindungen zwischen dem Arduino und dem Pi aus frei-hängenden Leitungen bestehen, ist es manchmal zu Verbindungsbrüchen gekommen. Dafür werden auch diese Fehler in dem Skript ignoriert.

Der komplette Skript ist im *Anhang 3: I2C-Master Python-Skript* zu finden.

Steuerung der Hardware-Pins

Für die Abfrage oder Ansteuerung von Hardware-Pins haben wir die WiringPi-Bibliothek[1] ausgewählt. Sie ist in C++ geschrieben und dadurch leicht ansprechbar aus dem anderen Code im Projekt. Auch wenn die Bibliothek als „deprecated“ vom ursprünglichen Entwickler gemeldet ist[2], wird die Software weiter in GitHub von der Community „inoffiziell“ gepflegt[3]. Die Installation erfolgt standardmäßig mit[4]:

```
sudo apt install wiringpi
```

5.4 ReSpeaker Hat: Inbetriebnahme, Steuerung und Tests

Die ReSpeaker-Hat muss einige Funktionen erfüllen:

1. Den aus den zwei Mikrofonen und der Hardware-Bearbeitung resultierenden Klang an Pi zur Verfügung stellen;

2. Den Status der Alexa-Software über den drei APA102-LED Lampen anzeigen;

3. Die benötigten Pins aus dem Pi für die I2C-Kommunikation über eine Hardware-Schnittstelle erreichbar machen;

[1] Henderson (2021), Internetquelle
[2] http://wiringpi.com/wiringpi-deprecated/, (Zugriff am 20.02.2021)
[3] Henderson; Contributors (2020), Internetquelle
[4] http://wiringpi.com/download-and-install/, (Zugriff am 20.02.2021)

4. Ein Benutzer-Button zum Starten von Alexa anbieten.

Die 3. und 4. sind schon in der Hardware verbunden und lassen sich durch die WiringPi-Bibliothek ansteuern. Betrachten wir die anderen zwei Funktionen im Einzelnen:

Die zwei Mikrofone

Die mit der ReSpeaker-Hat verfügbaren Software lässt sich leicht nach der Anleitung installieren[1]. Die Funktion der Audio-Aufnahme ist nach einem Neustart sofort vorhanden:

```
git clone https://github.com/respeaker/seeed-voicecard.git
cd seeed-voicecard
sudo ./install.sh
sudo reboot
```

APA102-LED Lampen

In der Anleitung[2] wird ein Python-Skript angeboten, womit die LED-Lampen gesteuert werden können. Dieser hat sich in den Tests aufgrund des hohen Overhead von Python und der schweren und langsamen Kommunikation mit dem C++ Code von Alexa als nicht für unser Projekt geeignet gezeigt.

Als Basis für eine bessere Alternative haben wir die APA102-Bibliothek von Pololu ausgewählt[3]. Diese ist in C++ geschrieben, allerdings für Arduino. Aus diesem Grund waren für die Funktion auf Raspberry-Pi folgende Anpassungen notwendig:

- Umstellung von Arduino.h auf WiringPi.h für die interne Ansteuerung der Ein- und Ausgänge vom Mikrocontroller;
- Die korrekten Pins vom Raspberry-Pi für den Gebrauch von wiringPi einstellen[4];
- Reinigung des Codes durch Entfernung von Referenzen von unbenutzten Aufrufen zu der FastGPIO-Bibliothek, die bei unserem Projekt keine Verwendung findet.

[1] Vgl. Seeed Technology Co. Ltd (2020), Internetquelle
[2] Ebd.
[3] Pololu Corporation (2018), Internetquelle
[4] Vgl. http://wiringpi.com/pins/, (Zugriff am 20.02.2021)

Ein kleines C++ Programm wurde für Überprüfung der korrekten Funktion erstellt:

```
#include "APA102pi.h"

const std::uint8_t dataPin = 12;
const std::uint8_t clockPin = 14;
APA102<dataPin, clockPin> ledStrip;
const std::uint16_t ledCount = 3;
rgb_color colors[ledCount];
const std::uint8_t brightness = 1;

int main(int argc, char* argv[]) {
    colors[0] = rgb_color(255,0,0);
    colors[1] = rgb_color(0,255,0);
    colors[2] = rgb_color(0,0,255);
    ledStrip.write(colors, ledCount, brightness);

    return 0;
}
```

Kompiliert durch:

```
g++ -lpthread -lwiringPi -o blink blink.cpp
```

Die komplette APA102pi.h Bibliothek ist im Software-Anhang zu dieser Arbeit verfügbar.

5.5 Alexa-SDK Sample App: Konfiguration und Kompilierung

Für die Alexa-SDK sind detaillierte Anweisungen für die Installation und Nutzung auf dem Raspberry-Pi[1]. Die Installation erfolgte nach den dort angegebenen Schritten, mit wenigen Anpassungen, die jetzt nacheinander betrachtet werden:

Hardware

In unserem Projekt erfüllt die Hardware alle Voraussetzungen, mit den folgenden Ausnahmen:

- USB 2.0 Mikrofon: diese Funktion ist schon von der ReSpeaker-Hat übernommen;

[1] Amazon Inc. (o.J.), Internetquelle

- Tastatur und Maus: der Pi arbeitet im Headless-Modus und wird komplett über *ssh* gesteuert;
- Lüfter: auch bei längeren Tests hat das Board nie die Temperatur erreicht, bei der die automatische Drosselung aktiviert wird, was über *vcgemncmd* bestätigt wurde[1].

Benutzerkonto

Ein Benutzerkonto ist für die Funktion der SDK erforderlich. Für das Projekt wurde das Amazon-Konto vom Verfasser benutzt. Die Einrichtung eines neuem Alexa-fähigem Produkts ist sehr detailliert von Amazon beschrieben[2] und wird hier nicht weiter betrachtet. Das Ziel dieser Registrierung ist neben der Anmeldung des neuen Geräts, die Erstellung einer Konto- und Produkt-gebundenen Konfigurationsdatei namens *config.json*[3], die dem Projekt zugefügt werden muss, und für die Authentifizierung in Amazon-Cloud notwendig ist.

Software auf dem Raspberry-Pi

Neben den aufgeführten Software-Dependencies fehlte bei unserer Installation *gstreamer1.0-alsa*, festgestellt durch Meldung bei der Kompilierung vom SDK. Zusätzlich fehlte die Software git, die für die Synchronisierung mit der GitHub-Repository notwendig ist.

Beide Pakete wurden problemlos über *apt install* installiert.

Kompilierung der SDK

Wir verzichten auf WakeWord-Engine, weil diese Funktion das EasyVR-Board übernimmt. Somit sieht der reduzierte *cmake*-Aufruf wie folgt aus:

```
cmake /home/pi/sdk-folder/sdk-source/avs-device-sdk \
 -DGSTREAMER_MEDIA_PLAYER=ON \
 -DPORTAUDIO=ON \
 -DPORTAUDIO_LIB_PATH=/home/pi/sdk-folder/third-party/portaudio/lib/.libs/
     libportaudio.a \
 -DPORTAUDIO_INCLUDE_DIR=/home/pi/sdk-folder/third-party/portaudio/include
```

[1] Vgl. https://www.raspberrypi.org/documentation/raspbian/applications/vcgencmd.md, (Zugriff am 20.02.2021)

[2] https://developer.amazon.com/en-US/docs/alexa/alexa-voice-service/register-a-product-with-avs.html, (Zugriff am 20.02.2021)

[3] Vgl. Amazon Inc. (o.J.), Internetquelle

Dank der Verfügbarkeit eines SWAP-Bereichs ist die Kompilierung überhaupt möglich, auch wenn es mehrere Stunden dauert. Positiv zu sehen ist die Tatsache, dass nach Änderungen der Quellendateien, nur die geänderten Dateien neu kompiliert werden. Dadurch ist eine massive Beschleunigung von Folge-Kompilierungen sichergestellt.

Konfigurationsdatei und Autorisierung

Die von Amazon heruntergeladene *config.json* Datei kopieren wir im Ordner */home/pi/sdk-folder/sdk-source/avs-device-sdk/tools/Install* und starten den dort vorhandenen Skript. Dieser erzeugt die benötigte Datei und beim Starten gibt die Software einen Bestätigungscode, den wir auf *https://amazon.com/us/code* eingeben und somit die SampleApp freischalten.

5.6 Alexa-SDK SampleApp: Anpassung und Integration im Projekt

Die von Amazon mit der SDK gelieferte SampleApp dient nur als die Basis des Assistenten und braucht einige Anpassungen um für das Projekt nutzbar zu sein.

WiringPi

Die Hardware-Pins werden über der WiringPi-Bibliothek angesprochen. Diese muss dem Build-System bekannt gemacht werden.

Bei der Konfiguration von *cmake* in *SampleApp/src/CMakeLists.txt* fügen wir Information über der Bibliothek mithilfe der folgenden Befehle hinzu:

- *find_library*[1], um die Bibliothek zu finden und
- *target_link_libraries*, um diese am Projekt zu verknüpfen[2]

Somit wird die Datei *SampleApp/src/CMakeLists.txt* mit folgenden Zeilen ergänzt:

```
find_library(wiringPi_LIB wiringPi)
target_link_libraries(LibSampleApp ${wiringPi_LIB})
```

[1] Vgl. https://cmake.org/cmake/help/latest/command/find_library.html, (Zugriff am 20.02.2021)

[2] Vgl. https://cmake.org/cmake/help/latest/command/target_link_libraries.html, (Zugriff am 20.02.2021)

LED-Lampen

Der *UIManager* pflegt die internen Zustände der SampleApp[1]. Diese Zustände vereinfachen wir in drei Möglichkeiten:

1. Mikrofon ist aktiv: Alexa hört zu. Alle LED-Lampen sind an;
2. Alexa denkt/ holt Antwort aus der Cloud. Abgebildet durch Lauflicht;
3. Alexa redet. Angedeutet von langsam blinkenden Lichtern.

Für die direkte Steuerung der Lampen nach Änderung des Zustandes fügen wir einen extra Thread im *UIManager* zu, der mit einer einfachen *switch-case* Anweisung die LED-Muster wechselt.

Benutzereingaben

Die Standardeinstellung bei der SampleApp ist Benutzereingaben über der Konsole zu lesen. Diese sind in der Form einfacher Anweisungen, die aus einem Buchstaben bestehen[2] und deren Bedeutung wird dem Benutzer über den Help-Screens erklärt[3]. Dieses Verhalten wollen wir mit den Optionen für Eingabe durch die Hardware ergänzen und somit muss der Code folgende zwei Eingabearten unterstützen:

- Alexa über der Systemkonsole steuern;
- Kommunikation mit Alexa starten, wenn das entsprechende Signal vom *EasyVR* ankommt oder wenn der ReSpeaker-Hat Knopf gedrückt wird.

Die Benutzereingabe lesen wir mit *cin>>variable* ab[4]. Das Problem dabei ist, dass wenn der Benutzer noch nichts eingegeben hat, die weitere Ausführung blockiert, bis die Eingabe erfolgt. Weil das Programm immer bereit sein muss auf die Benutzerwünsche zu reagieren, müssen wir die zwei Arten von Eingaben in zwei verschiedene Threads trennen. Der schon im Code für die Benutzereingabe vorhandene Thread *ConsoleReader::WorkerLoop* wurde so modifiziert, dass er selber nichts liest, sondern auf die Ereignisse aus zwei neuen Threads reagiert. Die beiden neuen Threads kümmern sich entsprechend um die Eingabe über *cin* und über die Ablesung der Hardware-Pins. Ein Signal aus dem EasyVR-Board,

[1] Amazon Inc. (2020), Quellencode: *SampleApp/include/SampleApp/UIManager.h*, Zeile 469-470
[2] Amazon Inc. (2020), Quellencode: *SampleApp/src/UserInputManager.cpp*, Zeile 35-91
[3] Amazon Inc. (2020), Quellencode: *SampleApp/src/UIManager.cpp*, Zeile 52-554
[4] Vgl. Stroustrup (2013), S.93

oder ein Knopfdruck auf der ReSpeaker-Hat wird als den Buchstaben „t“ gemeldet, was die Kommunikation mit Alexa startet (Tap to Talk)[1]. Aus der Sicht der weiteren Ausführung des Programms, besteht dadurch kein Unterschied zwischen ob der Benutzer ein „*t*“ in die Konsole eingegeben, den Knopf gedrückt, oder über EasyVR den Befehl „*Alexa!*“ gegeben hat.

Da der Code in *UserInputManager* an mehreren Stellen die „*cin>>*“ Anweisung verwendet, um Benutzereingaben zu holen, würde das die Ausführung des gesamtem Programms blockieren. Damit es weiter auch auf die anderen Ereignisse reagieren kann, haben wir diese Anweisungen, wenn möglich, mit einem Aufruf von *ConsoleReader::read* ersetzt, der die neu zugefügten Threads verwendet. Es gab leider auch Stellen, in denen das nicht möglich ist. Der Grund ist, dass an diesen Stellen aus *cin* nicht ein *char*-Buchstabe, sondern eine komplette Zeichenkette gelesen werden muss. Unterstützung für die Ablesung von Zeichenketten (Strings) in Zusammenhang mit den Hardware-Ereignissen würde weitere Anpassungen des Codes als Folge haben, die den Umfang dieser Arbeit sprengen würden, deshalb wurde die Unterstützung dieser Funktionalität ausgelassen.

Übersicht

Der komplette Ablauf in Hinsicht auf die Benutzereingaben, inklusive der LED-Rückmeldungen ist auf Abbildung 3 dargestellt. Die neu zugefügten Funktionalitäten sind auf dem Diagramm hervorgehoben.

5.7 OpenHAB: Installation und Inbetriebnahme

OpenHAB hat eine sehr detaillierte und umfangreiche Dokumentation[2], darunter auch Anleitung für die Installation und Inbetriebnahme unter Raspberry-Pi[3]. Auch wenn die empfohlene Installationsmethode die direkte Übernahme einer ISO Datei auf der SD-Karte ist[4], nutzten wir die manuelle Installation aufgrund der besseren Kontrolle und Kompatibilität mit der schon vorhandenen Software.

Die *Zulu* Java-Maschine

Das Einzige, was gesondert als Vorbereitung für *OpenHAB* auf dem Projekt-Pi gemacht werden muss, ist die Installation der Open-Source Java Maschine namens *Zulu*. Die Entwickler von *Zulu* bieten keine APT-Variante der Software für die

[1] Amazon Inc. (2020), Quellencode: *SampleApp/src/UIManager.cpp*, Zeile 85-86
[2] openHAB Foundation e.V. (2020), Internetquelle
[3] https://www.openhab.org/docs/installation/rasppi.html, (Zugriff am 20.02.2021)
[4] Ebd.

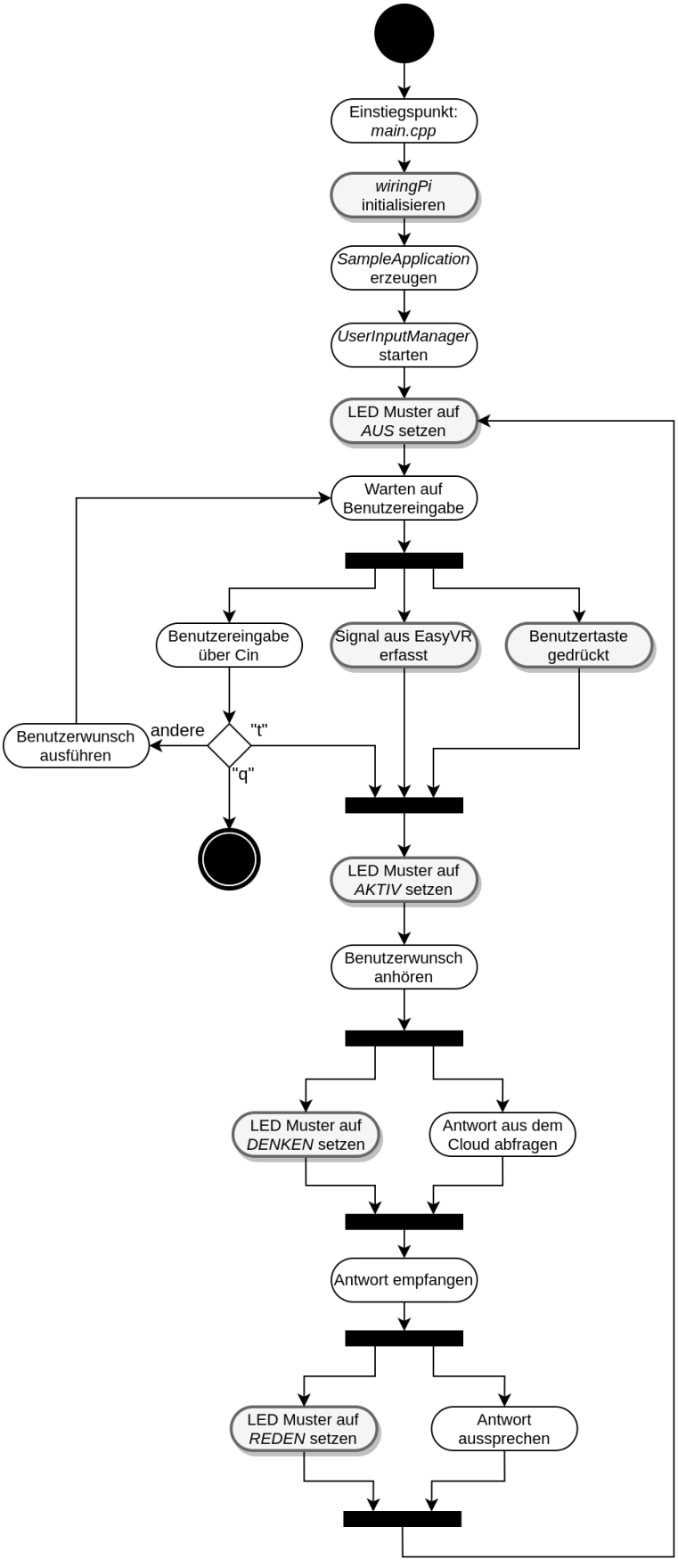

Abbildung 3: Alexa-SDK *SampleApp* mit Modifikationen

ARM-Architektur an[1], also wurde *Zulu* als ein *tar.gz* Archiv heruntergeladen. Die Zulu-Webseite deutet auf eine Anleitung für die Installation unter Raspberry-Pi hin[2], welche wir für das Projekt gefolgt haben. Die folgenden Befehle extrahieren das Archiv und melden *Zulu* als die Standard-Java-Maschine auf dem Pi an.

```
sudo update-alternatives --install /usr/bin/java java ~/zulu/zulu8.50.51.263-
    ca-jdk8.0.275-linux_aarch32hf/bin/java 100
sudo update-alternatives --config java # Zur Verifikation
```

Eine nachfolgende Prüfung bestätigt, dass *Zulu* erfolgreich installiert wurde:

```
pi@pi-Z:~/zulu $ java -version
openjdk version "1.8.0_275"
OpenJDK Runtime Environment (Zulu 8.50.51.263-CA-linux_aarch32hf) (build 1.8.0
    _275-b01)
OpenJDK Client VM (Zulu 8.50.51.263-CA-linux_aarch32hf) (build 25.275-b01,
    mixed mode)
```

Die Version ist größer, als die als Minimum erforderliche für *OpenHAB* 161[3]. Somit ist die Installation erfolgreich.

OpenHAB Installation

Nach der *OpenHAB*-Anleitung[4] laden wir den *APT*-Schlüssel von *OpenHAB*, erlauben Aktualisierungen über *HTTPS*, installieren *OpenHAB* und den Bündel von Add-Ons und starten die Software als ein Service auf dem Pi:

```
wget -qO - \
   'https://bintray.com/user/downloadSubjectPublicKey?username=openhab' | \
   sudo apt-key add -
sudo apt-get install apt-transport-https
sudo apt-get update
sudo apt-get install openhab2
sudo apt-get install openhab2-addons
sudo systemctl start openhab.service
```

OpenHAB ist dann direkt über dem Browser unter der Netzwerkadresse vom Pi erreichbar. Bei den Tests hat die Verbindung über *HTTPS* nicht funktioniert mit der Fehlermeldung:

[1] Azul Systems, Inc. (2021), Internetquelle
[2] Cabè (2016), Internetquelle
[3] https://www.openhab.org/docs/installation/#prerequisites, (Zugriff am 20.02.2021)
[4] https://www.openhab.org/docs/installation/raspi.html#installation-of-openhab, (Zugriff am 20.02.2021)

```
Error code: SSL_ERROR_NO_CYPHER_OVERLAP
```

Weil sich das Projekt im Heimnetzwerk befindet, verzichten wir auf weiterer Recherche zu dieser Problematik und benutzen *HTTP*.

OpenHAB Inbetriebnahme

Nach der *OpenHAB*-Dokumentation[1] wurde für das Projekt „*Standard*“ bei dem „*Initial Setup*“ ausgewählt, und nach kurzer Wartezeit „*Paper-UI*“ für die Benutzeroberfläche.

Für die ausgewählten Smart-Home Geräte müssen wir die s.g. *Bindings* installieren - *OpenHAB*-Komponenten, die eine Schnittstelle für die Kommunikation mit den angeschlossenen Geräten anbieten[2]. Folgende *Bindings* wurden installiert:

- *ZigBee*: für die Kommunikation mit der Philips Hue-Lampe, die das ZigBee-Protokoll verwendet[3];
- *Network*: für die Anbindung von Geräten im lokalen Netzwerk, wie die Shelly-Steckdose[4];
- *TP-Link*: für Unterstützung von Geräten aus dem Hersteller TP-Link, wie die Kasa Smart Steckdose[5];
- *Dresden Elektronik deCONZ Binding*: für die Nutzung des USB-Gateways von Dresden Elektronik[6];
- *Hue*: für die Nutzung der Philips Hue Bridge[7]. Das Projekt nutzt diese Hardware nicht, allerdings wird diese Komponente vom *deCONZ Binding* benötigt[8].

5.8 Anbindung der Smart-Geräte

Für die Anbindung der Smart-Geräte folgen wir die mitgelieferten Anleitungen und die Dokumentation von *OpenHAB*.

[1] https://www.openhab.org/docs/tutorial/, (Zugriff am 12.12.2020)
[2] https://www.openhab.org/docs/concepts/, (Zugriff am 20.02.2021)
[3] https://www.openhab.org/addons/bindings/zigbee/, (Zugriff am 20.02.2021)
[4] https://www.openhab.org/addons/bindings/network/, (Zugriff am 20.02.2021)
[5] https://www.openhab.org/addons/bindings/tplinksmarthome/, (Zugriff am 20.02.2021)
[6] https://www.openhab.org/addons/bindings/deconz/, (Zugriff am 20.02.2021)
[7] https://www.openhab.org/addons/bindings/hue/, (Zugriff am 20.02.2021)
[8] https://www.openhab.org/addons/bindings/deconz/, (Zugriff am 20.02.2021)

Philips Hue-Lampe

Für die Kommunikation mit der Lampe, weil keine Hue-Bridge vorhanden ist, benutzen wir den deCONZ USB Gateway. Die Installation von diesem erfolgte nach den Anweisungen vom Hersteller[1] durch Zufügen vom Benutzer in der *dialout*-Gruppe, zufügen des APT-Schlüssels von deCONZ und die nachfolgende Installation der Software über APT:

```
sudo gpasswd -a $USER dialout
wget -O - http://phoscon.de/apt/deconz.pub.key | sudo apt-key add -
sudo sh -c "echo 'deb http://phoscon.de/apt/deconz \
          $(lsb_release -cs) main' > \
          /etc/apt/sources.list.d/deconz.list"
sudo apt update
sudo apt install deconz
```

Da der Pi im Headless-Modus arbeitet, kann die Software nicht von dem Desktop aus genutzt werden, sondern über dem Browser, durch die Eingabe der Adresse der Phoscon-App *http://phoscon.de/app*[2].

Nach Erstellung eines Passworts und der Übernahme aller Standard-Einstellungen, haben wir die Hue-Lampe angeschlossen und nach kurzer Zeit erschien sie in der Phoscon-App Oberfläche. Die Lampe wurde zu einer allgemeinen Gruppe zugefügt. Weitere Einstellungen wurden nicht betrachtet, weil eine Steuerung über der Phoscon-App kein Teil des hybriden Assistenten ist.

Um externe Software wie die OpenHAB an der Phoscon-App zu verbinden, muss der Button „*App verbinden*“ unter „*Menü > Einstellungen > Gateway > Erweitert*“ geklickt werden. Danach sind 60 Sekunden für die Anbindung verfügbar[3].

Bei einer erneuten Suche mit *„autodiscovery*“ in der OpenHAB-Benutzeroberfläche erschien die Hue-Lampe auch dort und wurde als *Thing* unter dem Namen „Hue Lampe“ zugefügt.

Shelly

Die Shelly-Steckdose kann über einer App konfiguriert werden, aber auch direkt über WLAN[4]. Nach dem Anschluss der Steckdose erzeugt sie ein eigenes WLAN-Netzwerk, in dem wir uns verbinden können und über Web-Interface an

[1] https://www.phoscon.de/de/conbee2/install#raspbian, (Zugriff am 20.02.2021)
[2] https://www.phoscon.de/de/app/doc, Internetquelle
[3] Ebd.
[4] Vgl. Allterco Robotics LTD (2019), S.1

der Adresse http://192.168.33.1 die Einstellungen wie Name und Passwort des eigenes Heimnetzwerks eingeben können[1].

Das Zufügen bei *OpenHAB* erfolgte schnell nach einer erneuten Gerätesuche.

TP-Link Kasa Smart Plug

Bei der TP-Link Steckdose besteht keine Option Einstellungen vorzunehmen, ohne eine zusätzliche App zu installieren[2]. Allerdings kann bei der App auf die Erstellung eines Benutzerkontos verzichtet werden. Nach der Eingabe der Daten des Heimnetzwerks und erneuter Suche in OpenHAB wurde diese Steckdose auch problemlos erkannt und angebunden.

6 Ergebnisse und Ausblick

Die Ergebnisse mit dem in dieser Arbeit gebautem Prototyps waren bei der Mehrzahl an Tests erfolgreich[3].

Als Schwachstelle hat sich die Spracherkennung von *EasyVR* gezeigt, was manchmal dazu führte, dass Befehle nicht verstanden wurden, oder dass das Board sporadisch auf externen Geräuschen reagierte. Der einfache Zustandsautomat mit den zwei Zuständen Aktivierungswort und Folgebefehl hat nicht immer richtig funktioniert. Bei manchen Verläufen hat das Board einfach „*Error*“ in die serielle Konsole vom Arduino ausgegeben, ohne weiteren Hinweisen auf was der Fehler ist oder woran er liegen kann. Wie schon in den vorigen Kapiteln diskutiert, wurde bei dem Prototyp die Sprecher-abhängige Spracherkennung verwendet, da die andere Variante praktisch unbrauchbar ist. Leider konnten die Fehlermeldungen auch so nicht komplett vermieden werden.

Die enttäuschenden Ergebnisse könnten unter anderem an den Erfahrungen mit Online-Diensten liegen, die heutzutage nahezu perfekt sind. So ein Vergleich wäre natürlich für den *EasyVR* sehr unfair, angesichts seiner sehr kompakten Form, begrenzten Ressourcen und die lokale Verarbeitung der Daten.

Unter Berücksichtigung des Mangels an Angeboten für Offline-Spracherkennung auf dem Markt und des geringem Anschaffungspreises für den *EasyVR*, verdient das Board eine insgesamt gute Note für seine Leistung. Ein Kompromiss an der Qualität der Spracherkennung wird mit der Erhaltung der Privatsphäre abge-

[1] Ebd.

[2] TP-Link Technologies Co., Ltd (2020), S.6

[3] Ein Video mit dem Prototypen im Betrieb ist auf https://vimeo.com/493701809 und im Anhang dieser Arbeit verfügbar.

kauft. Es liegt am Benutzer die Entscheidung zu treffen, ob sich dies für ihn lohnt.

Ein anderer Schwachpunkt des hybriden Assistenten ist die Inbetriebnahme und die Einstellungen, welche viel die Linux-Konsole, Hardware-Kenntnisse oder zusätzliche Software benötigen. Somit ist die Inbetriebnahme leider komplexer als zum Beispiel die eines Amazon-Echo Geräts. Da der Prototyp bei dieser Arbeit nur als einen konzeptuellen Beweis dienen soll, bestehen viele Möglichkeiten für Verbesserungen. Ein kommerzielles Gerät kann als eine zusammengebaute Hardware samt vorinstallierter Software angeboten werden. Es muss überlegt werden, wie Aktualisierungen oder Änderungen am System in einer benutzerfreundlichen Weise durchgeführt werden könnten.

Bei der beteiligten Software bleiben auch noch Wünsche offen. Die Sammlung an Skripten und Bibliotheken kann zusammengefasst und besser koordiniert werden. Eine komplette Änderung des Konzepts für die Erfassung der Benutzereingabe kann notwendig sein, damit auch alle Einstellungen in der *SampleApp* vom Benutzer erreicht werden können.

7 Zusammenfassung

Auch wenn die Spracherkennung der kommerziell angebotenen Systeme schon so gut wie perfekt ist, kommt dies mit einigen Problemen und Gefahren, durch die massive Invasion der Privatsphäre. Die Geräte befinden sich zu Hause und oftmals in mehreren Räumen und hören immer zu. Auch wenn noch keine Verstöße seitens der Anbieter bekannt sind, kommt es zu Problemen aus technischem Charakter oder zu Angriffen aus dem Netz. Der Wert der beinhalteten Information, die ein Smart-Assistent sammelt ist schwer einzuschätzen, gerade weil es um das private Leben des Benutzers geht. Leider ist die moderne Offline-basierte Spracherkennung noch nicht auf dem Niveau seiner Online-Kollegen und davon gibt es vergleichsweise weniger Angebote auf dem Markt.

Der in dieser Arbeit entwickelte Prototyp macht eine Kombination aus beiden Ansätzen und versucht Spracherkennung mit Erhaltung der Privatsphäre zu kombinieren. Die Betrachtung, dass es garnicht nötig ist, viele Befehle ins Internet zu schicken, nur um die lokal-vorhandenen Geräte zu steuern, führte zu der Verwendung eines Offline-Spracherkennungs-Boards für diese Aufgaben und auch für die Kontrolle der Online-Komponente. Sie besteht aus der Alexa-SDK Software, die auf einem Einplatinenrechner - dem Raspberry-Pi läuft.

Um die Benutzererfahrung zu verbessern, wurde der Pi mit einer Hat mit zwei

Mikrofonen und zusätzlicher Hardware für eine bessere Audioerfassung, sowie auch LED-Lampen um den Status der Alexa-Software anzuzeigen.

Drei aktuell-populäre Smart-Geräte wurden über WLAN und ZigBee angeschlossen um die Funktion zu testen. Deren direkte Steuerung übernimmt die Open-Source Software für Hausautomatisierung OpenHAB.

Schwachstellen wurden bei der Offline-Spracherkennung, bei der Benutzerfreundlichkeit der Installation und bei der Integration und dem Zusammenspiel der beteiligten Software und Hardware festgestellt.

Zusätzliche Einschränkungen entstehen von der Benutzung der Alexa-SDK Software, wie zum Beispiel die Unmöglichkeit Prime-Musik abzuspielen.

Weitere Arbeit, die über dem Umfang dieser Arbeit hinaus geht, ist auf jeden Fall notwendig. Ein Vergleich mit anderen Systemen haben wir nicht betrachtet - für die Offline-Spracherkennung sowie auch für die Online-Komponente. Auch wenn es nicht so viele gibt, wird ein Vergleich und weitere Recherche für die Verbesserung des Konzepts hilfreich sein. Bei der Online-Komponente war der Fokus in der Arbeit mehr auf Alexa, die anderen Produkte wurden im Hintergrund gelassen.

Bei der verwendeten Software und Bibliotheken wurde das Minimum gemacht, um die Funktion des Prototypen herzustellen. Weitere Anpassungen und Verbesserungen können durchaus die Benutzererfahrung verbessern.

Wie schon im letzten Kapitel erwähnt, wurde eine Schnittstelle für die benutzerfreundliche Kommunikation bei der Installation außer Sicht gelassen. Der Aufbau einer solchen Schnittstelle würde aber den Umfang bei weitem überspringen.

Auf Tests mit verschiedenen Benutzern und Szenarien wurde auch verzichtet. Solche Tests können zu weiterer Optimierung und Verbesserung des Systems führen und ein besseres Bild für die Anwendungsmöglichkeiten und Einschränkungen des hybriden Assistenten darstellen.

Anhang

Anhang 1: Hardware-Schaltungen

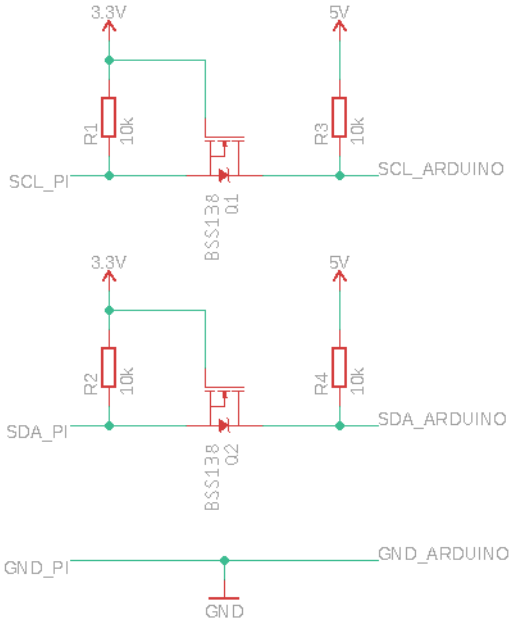

Abbildung 4: Bidirektionaler Level Shifter für I2C

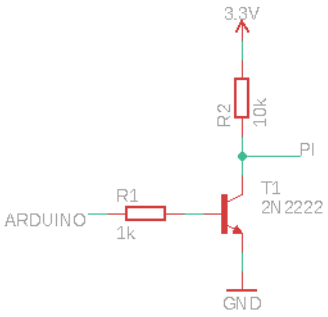

Abbildung 5: Logik-Schaltung

Anhang 2: EasyVR

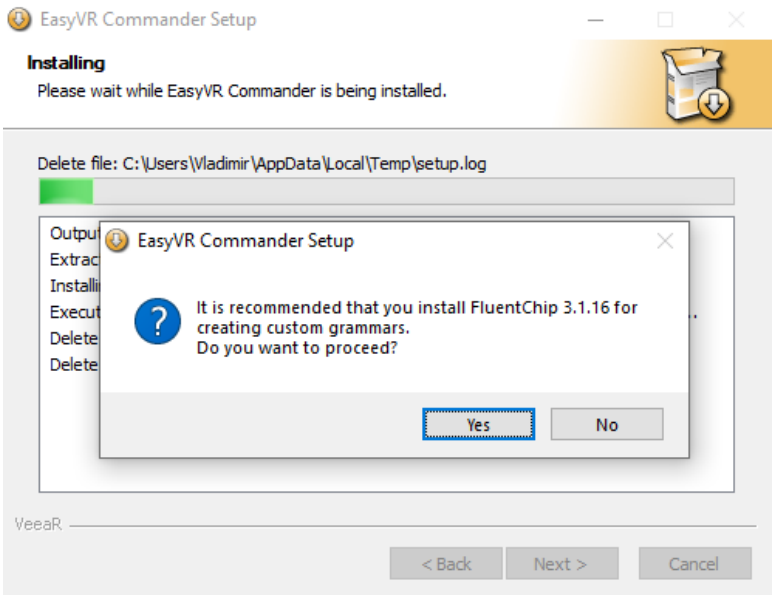

Abbildung 6: EasyVR: Installation von FluentChip

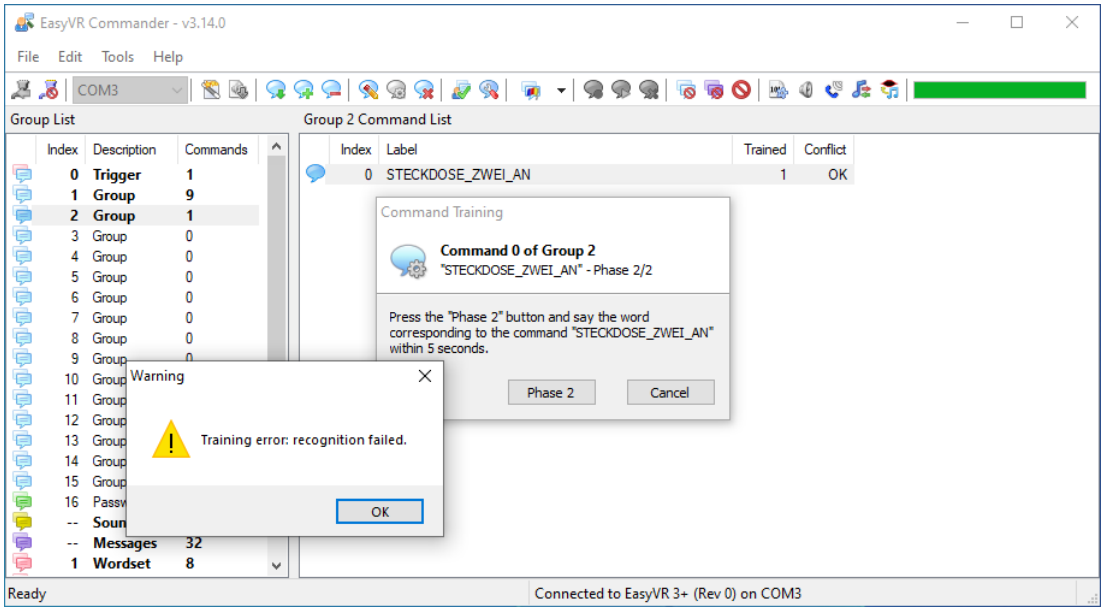

Abbildung 7: EasyVR: Training von längeren Befehle

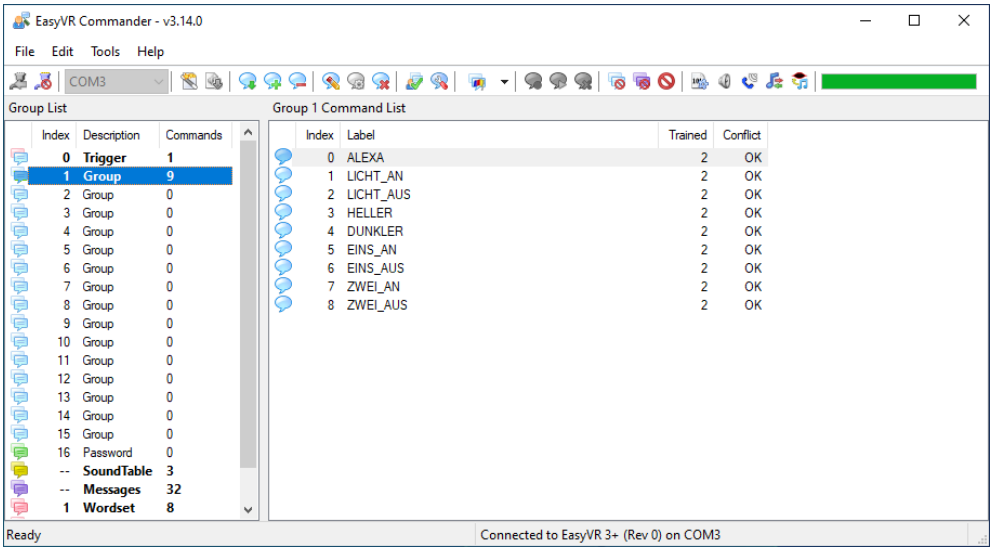

Abbildung 8: EasyVR: Alle Befehle

Anhang 3: I2C-Master Python-Skript

Listing 2: I2C-Master endgültige Version

```
#!/usr/bin/python3
import time, requests
from smbus2 import SMBus

# taken from the EasyVR software
G1_LICHT_AN = 1
G1_LICHT_AUS = 2
G1_HELLER = 3
G1_DUNKLER = 4
G1_EINS_AN = 5
G1_EINS_AUS = 6
G1_ZWEI_AN = 7
G1_ZWEI_AUS = 8

# taken from OpenHAB settings
#OH_url_base = 'http://pi-z.local:8080/rest/items/'
OH_url_base = 'http://localhost:8080/rest/items/'
OH_url_licht = 'HueLampe_Color'
OH_url_eins = 'TPLINK_Power'
OH_url_zwei = 'Shelly_Power'
OH_cmd_an = 'ON'
OH_cmd_aus = 'OFF'
OH_cmd_heller = 'INCREASE'
OH_cmd_dunkler = 'DECREASE'

def rest_request(url, payload):
    x = requests.post(url = url, data = payload)
    # TODO: check response
```

```
bus = SMBus(1)
done = False
print("Polling started. To quit use Ctrl-C")
while not done:
    try:
        b = bus.read_byte_data(8, 0)
        if (b == G1_LICHT_AN):
            rest_request(OH_url_base + OH_url_licht, OH_cmd_an)
        elif (b == G1_LICHT_AUS):
            rest_request(OH_url_base + OH_url_licht, OH_cmd_aus)
        elif (b == G1_HELLER):
            rest_request(OH_url_base + OH_url_licht, OH_cmd_heller)
        elif (b == G1_DUNKLER):
            rest_request(OH_url_base + OH_url_licht, OH_cmd_dunkler)
        elif (b == G1_EINS_AN):
            rest_request(OH_url_base + OH_url_eins, OH_cmd_an)
        elif (b == G1_EINS_AUS):
            rest_request(OH_url_base + OH_url_eins, OH_cmd_aus)
        elif (b == G1_ZWEI_AN):
            rest_request(OH_url_base + OH_url_zwei, OH_cmd_an)
        elif (b == G1_ZWEI_AUS):
            rest_request(OH_url_base + OH_url_zwei, OH_cmd_aus)

        # print debug message
        if (b != 0):
            print(b)
        time.sleep(0.5)

    # sometimes we need to wait for OpenHAB to become ready - just keep trying
    except requests.HTTPError as e:
        print(e)

    # this gets thrown by problems with the I2C bus, so we just keep retrying
    except OSError as e:
        print(e)

    except KeyboardInterrupt:
        bus.close()
        done = True
print()
print("Bye")
```

Literaturverzeichnis

Allterco Robotics LTD (2019)
Shelly-1 WiFi Relay Switch USER GUIDE, o.O.

Amazon Inc. (2020)
AVS Device SDK, https://github.com/alexa/avs-device-sdk, (commit 6840059) (Zugriff am 02.12.2020)

Amazon Inc. (2021)
Help & Customer Service, https://www.amazon.com/gp/help/customer/display.html (Zugriff am 12.02.2021)

Amazon Inc. (o.J.)
Set Up the AVS Device SDK on Raspberry Pi, https://developer.amazon.com/en-US/docs/alexa/avs-device-sdk/raspberry-pi.html (Zugriff am 28.12.2020)

APA Electronic co. LTD (2013)
Super LED: Product Features, o.O.

Arduino S.r.l. (2021)
Language Reference, https://www.arduino.cc/reference/en/ (Zugriff am 20.02.2021)

Azul Systems, Inc. (2021)
Open Source Java for Embedded Systems & ISVs, https://www.azul.com/downloads/zulu-community/?package=jdk (Zugriff am 20.02.2021)

Barda, D.; Zaikin, R.; Shriki, Y. (2020)
Keeping the gate locked on your IoT devices: Vulnerabilities found on Amazon's Alexa, Check Point Research, https://research.checkpoint.com/2020/amazons-alexa-hacked/ (Zugriff am 12.02.2021)

Bleich, H. (2019)
Alexa, wer hat meine Daten?, c't 1/2019, https://www.heise.de/select/ct/2019/1/1546323197251453 (Zugriff am 12.02.2021)

Broadcom Corporation (2012)
BCM2835 ARM Peripherals, o.O.

Cabè, B. (2016)
Installing the Zulu open source Java Virtual Machine on Raspberry Pi, https://blog.benjamin-cabe.com/2016/04/05/installing-the-zulu-open-source-java-virtual-machine-on-raspberry-pi (Zugriff am 20.02.2021)

Code Computerlove Ltd (2019)
Voice assistant survey 2019, https://www.codecomputerlove.com/blog/voice-assistant-survey-2019/ (Zugriff am 12.02.2021)

Coldewey, D. (2019)
Google's new voice recognition system works instantly and offline (if you have a Pixel), TechCrunch, https://techcrunch.com/2019/03/12/googles-new-voice-recognition-system-works-instantly-and-offline-if-you-have-a-pixel/ (Zugriff am 12.02.2021)

Conrad Connect GmbH (2019)
Bundesweite Smart Home-Analyse: Diese Geräte nutzt Deutschland, https://conradconnect.com/de/bundesweite-smart-home-analyse (Zugriff am 12.02.2021)

Deng, L.; Huang, X. (2004)
Challenges in adopting speech recognition, in: The Association for Computing Machinery(Hrsg): Communications of the ACM, 47, S. 69–75, New York

Dresden Elektronik Ingenieurtechnik GmbH (2021)
ConBee II: Das universelle Zigbee USB-Gateway, https://www.phoscon.de/de/conbee2 (Zugriff am 20.02.2021)

electronupdate (2021)
AMAZON ECHO FLEX: MICROPHONE MUTE, REAL OR FAKE?, https://electronupdate.blogspot.com/2021/01/amazon-echo-flex-microphone-mute-real.html (Zugriff am 12.02.2021)

Franklin, N. (2006)
dphys-swapfile Auto-setup/-activation System, http://neil.franklin.ch/Projects/dphys-swapfile/ (Zugriff am 20.02.2021)

Free Software Foundation, Inc. (2019)
<util/atomic.h> Atomically and Non-Atomically Executed Code Blocks, https://www.nongnu.org/avr-libc/user-manual/group___util___atomic.html (Zugriff am 20.02.2021)

Henderson, G. (2021)
Wiring Pi: GPIO Interface library for the Raspberry Pi, http://wiringpi.com/reference/ (Zugriff am 20.02.2021)

Henderson, G.; Contributors (2020)
Unofficial WiringPi Mirror, https://github.com/WiringPi/WiringPi (Zugriff am 20.02.2021)

Horcher, G. (2018)
Woman says her Amazon device recorded private conversation, sent it out to random contact, KIRO 7 News, https://www.kiro7.com/news/local/woman-says-her-amazon-device-recorded-private-conversation-sent-it-out-to-random-contact/755507974/ (Zugriff am 12.02.2021)

Horowitz, P.; Hill, W. (2017)
The art of electronics, Third Edition, New York

Howard, D. M.; Murphy, D. T. (2007)
Voice science, acoustics, and recording, San Diege, Oxford, Brisbane

Huang, X.; Baker, J.; Reddy, R. (2014)
A historical perspective of speech recognition, in: The Association for Computing Machinery(Hrsg): Communications of the ACM, Volume 57, S. 94–103, New York

Juang, B.-H.; Rabiner, L. R. (2005)
Automatic speech recognition–a brief history of the technology development, Atlanta, Santa Barbara

Kitware, Inc.; Contributors (2020)
CMake Documentation, https://cmake.org/cmake/help/latest/ (Zugriff am 20.02.2021)

Lindegaard, K.-P. (2021)
smbus2, https://pypi.org/project/smbus2/ (Zugriff am 20.02.2021)

McGraw, I. et al. (2016)
Personalized speech recognition on mobile devices, in: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), S. 5955–5959, Shanghai

Mozilla Corporation (o.J.)
Common Voice: Frequently Asked Questions, https://commonvoice.mozilla.org/en/faq (Zugriff am 03.02.2021)

NXP Semiconductors (2007)
AN10441: Level shifting techniques in I2C-bus design, o.O.

openHAB Foundation e.V. (2020)
OpenHAB Documentation, https://www.openhab.org/docs/ (Zugriff am 12.12.2020)

openHAB Foundation e.V. (2021)
OpenHAB Add-on Reference, https://www.openhab.org/docs/ (Zugriff am 20.02.2021)

Pfister, B.; Kaufmann, T. (2017)
Sprachverarbeitung: Grundlagen und Methoden der Sprachsynthese und Spracherkennung, 2., aktualisierte und erweiterte Auflage, Berlin

Picovoice Inc. (2021)
Picovoice, https://picovoice.ai/pricing/ (Zugriff am 12.02.2021)

Pololu Corporation (2018)
APA102/SK9822 library for Arduino, https://github.com/pololu/apa102-arduino, (Version 3.0.0) (Zugriff am 20.02.2021)

Ramey, C.; Fox, B. (2020)
Bash Reference Manual, o.O.

Raspberry Pi Foundation (o.J.)
Raspberry Pi Documentation, https://www.raspberrypi.org/documentation/ (Zugriff am 28.12.2020)

RoboTech srl (2017)
EasyVR Library for Embedded Systems, https://fortebit.tech/download/13228/ (Zugriff am 20.02.2021)

RoboTech srl (2019a)
EasyVR 3 (Plus) User Manual, o.O. Release 1.0.17

RoboTech srl (2019b)
EasyVR Arduino library, https://github.com/fortebit/EasyVR-Arduino, (Version 1.11.1) (Zugriff am 20.02.2021)

RoboTech srl (2019c)
EasyVR-Python, https://github.com/fortebit/EasyVR-Python, (Version 1.11.1) (Zugriff am 20.02.2021)

RoboTech srl (o.J.)
EasyVR - SI/SD Commands and functions, o.O. Application Note (2.0)

S.C. BITDEFENDER S.R.L. (2016)
Security Awareness in the Age of Internet of Things: A 2016 Bitdefender Study,

Seeed Studio (2017)
ReSpeaker 2-Mics Pi HAT, Circuit Diagram, o.O.

Seeed Technology Co. Ltd (2020)
ReSpeaker 2-Mics Pi HAT, https://wiki.seeedstudio.com/ReSpeaker_2_Mics_Pi_HAT/ (Zugriff am 20.02.2021)

Signify GmbH (2021)
White & Color Ambiance: Einzelpack E27, https://www.philips-hue.com/de-de/p/hue-white—color-ambiance-einzelpack-e27/8718699673109 (Zugriff am 20.02.2021)

Software in the Public Interest, Inc. (2021)
Debian Manpages, https://manpages.debian.org/ (Zugriff am 20.02.2021)

Stotz, D. (2019)
Computergestützte Audio- und Videotechnik, 3. Auflage, Berlin, Heidelberg

Strategy Analytics Ltd. (2020)
Strategy Analytics at CES: Most Homes Are Now Smart Homes, https://www.businesswire.com/news/home/20200107005962/en/Strategy-Analytics-at-CES-Most-Homes-Are-Now-Smart-Homes (Zugriff am 12.02.2021)

Stroustrup, B. (2013)
The C++ programming language, Fourth Edition, Upper Saddle River et al.

Tanenbaum, A. S.; Bos, H. (2015)
Modern operating systems, Boston et al.

TP-Link Technologies Co., Ltd (2020)
TP-LINK User's Manual: Wi-Fi Smart Plug HS100; Wi-Fi Smart Plug with Energy Monitoring H S110, o.O.

Trojan, W. (2019)
Sprachsteuerung von IoT-Projekten mit Amazon Alexa, Aachen

VoxForge (2021)
About VoxForge, http://www.voxforge.org/home/about (Zugriff am 12.02.2021)

Xiaomi Inc. (2021)
New survey finds 70more than half used smart devices, https://blog.mi.com/en/2021/01/05/new-survey-finds-70-of-consumers-improved-home-during-covid-19-more-than-half-used-smart-devices/ (Zugriff am 12.02.2021)

Ich versichere, dass ich den beiliegenden Projektbericht selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet habe.

Schneverdingen, 20.02.2021

_____

(Vladimir Zhelezarov)