Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 115

Barrierefreies Smarthome

Tobias Ableitner

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Albrecht Schmidt
Betreuer/in:	Prof. Dr. Gottfried Zimmermann
Beginn am:	20. Juni 2016
Beendet am:	20. Dezember 2016
CR-Nummer:	H.5.2

Kurzfassung

Eine Querschnittlähmung kann jeden Menschen treffen, sei es durch durch Unfall oder Erkrankung. Tetraplegikern bleiben im schlimmsten Fall lediglich motorische Fähigkeiten im Bereich von Kopf, Hals und eventuell Schulter erhalten. Für sie – aber auch für Menschen mit vergleichbaren motorischen Einschränkungen – werden kleinste manuelle Verrichtungen plötzlich zu einer – oftmals sogar unlösbaren – Herausforderung. Aus diesem Grund können Tetraplegiker von einem Smarthome und mobilen Endgeräten zu seiner Steuerung erheblich profitieren. Dafür muss jedoch das Endgerät eine barrierefreie Eingabemethode unterstützen. Die Barrierefreiheit bei der Eingabe ist realisierbar in Form von Bedienungshilfen und / oder durch Hilfsmittel-Hardware.

Das Ziel dieser Arbeit ist die Konzeptionierung einer barrierefreien Smarthome-Steuerung für Tetraplegiker sowie deren prototypische Realisierung als Android App. Das Hauptaugenmerk dabei liegt auf einem größtmöglichen Verzicht auf Hilfsmittel-Hardware und auf einer guten Individualisier- und Erweiterbarkeit. Für Ersteres wird unter anderem Face-Tracking basierend auf der Mobile Vision API von Google als Eingabemethode realisiert. Abschließend findet sowohl eine qualitative als auch quantitative Evaluation des Prototyps statt. Verglichen werden dazu die Bandbreite des Face-Trackings mit weiteren barrierefreien sowie klassischen Eingabemethoden. Das Konzept und der Prototyp für die Anwendung zur Steuerung eines barrierefreien Smarthomes eignen sich für betroffene Personen und Angehörige aus der Zielgruppe, welche unterschiedliche Eingabemethoden ausprobieren möchten, ebenso für Entwickler, die an der Erweiterung des Prototyps interessiert sind. Ihnen liefert die Evaluation zudem Erkenntnisse zur Leistungsfähigkeit eines kostengünstigen Face-Trackings sowie weiterer barrierefreien Eingabemethoden.

Abstract

Spinal-cord injury can happen to anyone, whether through traumatic injury or due to illness. Tetraplegics retain mobility – in a worst case scenario – only in the head and neck area, possibly including the shoulder area. For those affected – as for people with similarly limited mobility through other causes – the smallest manual movement becomes a challenge, often an impossible one. Thus tetraplegics may profit immensely by a smart home with mobile controlling devices. However, for that purpose the controlling device needs to support barrier-free input methods. Barrier-free input can be implemented by assisting features in the use of control devices, and/or by specialized hardware.

The aim of this thesis is the conceptualization of a barrier-free smart home control device for tetraplegics, as well as its prototypical implementation as an Android App. The main focus is on doing without specialized hardware as much as possible, and on options to individualize and expand functionality. For the prototype, face-tracking based on Mobile Vision API by Google is used as an input method, among others. In conclusion there is an evaluation process of the prototype regarding criteria of quality and quantity. The evaluation includes comparing the range of performance of face-tracking with other barrier-free, as well as classic input methods. The concept and the prototype for this application for the controlling of barrier-free smart homes are suitable for affected persons and their family members in the target group who would like to try out various methods of input. Software developers may also be interested in expanding the functionality and input methods of the prototype. The evaluation provides insight into the performance of cost-effective face-tracking as well as barrier-free input methods.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Ein 'smartes' Zuhause kann für dessen Bewohner in vielerlei Hinsicht komfortabel sein. Für manche von ihnen mag es darüber hinaus auch ein technisches 'Spielzeug' sein. Ist es jedoch konzipiert, so dass es Tetraplegiker oder andere Bewohner mit vergleichbaren motorischen Einschränkungen selbstständig benutzen können, stellt es eine viel größere Bedeutung dar. Denn das Smarthome ist für diesen Personenkreis ein Zugewinn an Selbstständigkeit und damit an Lebensqualität.

1.1. Motivation

Nach Angaben der Weltgesundheitsorganisation erleiden jedes Jahr zwischen 250000 und 500000 Menschen eine Querschnittslähmung (vgl. [Org13, S. 14-15]). Wyndaele et al. [WW06] kamen in ihrer Studie zu dem Ergebnis, dass es sich bei einem Drittel der betrachteten Querschnittsfälle um eine Tetraplegie handelte (vgl. [WW06]). Bei dieser ist die Verletzung des Rückenmarks so hoch, dass die betroffenen Personen nur noch eingeschränkte oder gar keine motorischen Fähigkeiten mehr in ihren Händen besitzen (vgl. [AA14]). Obwohl der von Wyndaele et al. ermittelte Tetraplegikeranteil nicht repräsentativ für eine weltweite Schätzung ist (vgl. [WW06]), lässt er die Annahme zu, dass eine erhebliche Anzahl an Personen betroffen ist, zumal weitere Quellen in Abschnitt 2.2.3 einen noch höheren Anteil nennen.

Für Tetraplegiker und Menschen mit vergleichbaren Einschränkungen sind im Alltag schon kleine Handgriffe, wie zum Beispiel das Öffnen eines Fensters oder das Umschalten des Fernsehprogramms, nur mit erheblichen Aufwand oder überhaupt nicht bewältigbar. In Folge dessen benötigen sie eine umfangreiche Unterstützung. Deshalb, stellt für sie ein Smarthome, das per Definition Technologien und Dienste im Zuhause integriert, um damit die Lebensqualität in diesem zu steigern, ein weitaus größeres Potenzial dar, als für den Rest der Bevölkerung (vgl. [Sma16] u. [Pec14]).

Allerdings sind dafür einige Voraussetzung zu erfüllen. Zum einen erfolgt die Bedienung des Smarthomes idealerweise über eines oder mehrere mobile Endgeräte. Sie haben den Vorteil, dass sie klein und handlich sind und sich somit am Rollstuhl des Benutzers einfacher anbringen lassen, damit für diesen die Steuerung im Bedarfsfall erreichbar ist. Zum anderen muss die Benutzerschnittstelle barrierefrei sein. Insbesondere für Menschen mit motorischen Einschränkungen an ihren Händen erfolgt die Herstellung der Barrierefreiheit oft noch durch

Hilfsmittel-Hardware und nicht über Bedienungshilfen des Geräts. Im Rahmen der in Abschnitt 3.1 behandelten Interviews, wurde die Erfahrung geäußert, dass Erstere gegenüber vergleichbaren Eingabegeräten für Unversehrte oftmals teurer seien. Dies kann insbesondere Betroffene in ärmeren Regionen der Erde vor Probleme stellen.

1.2. Ziele der Arbeit

Das Ziel dieser Arbeit ist die Planung und prototypische Realisierung einer App zur Steuerung eines barrierefreien Smarthomes, das auf die Belange von Benutzern mit Mobilitätseinschränkungen an den Händen abgestimmt ist. Aus diesem Grund handelt es sich bei der primären Zielgruppe des barrierefreien Smarthomes um Personen, die vom Hals ab querschnittsgelähmt sind.

Die App soll sich zum einen durch eine hohe Individualisierbarkeit auszeichnen, wodurch sie an die Anforderungen sowie Einschränkungen des jeweiligen Benutzers anpassbar ist. Zum anderen liegt ein Schwerpunkt auf ihrer Erweiterbarkeit. Dadurch lassen sich zu einem späteren Zeitpunkt weitere Geräte in das Smarthome integrieren.

Um die Bedienbarkeit für die Zielgruppe sicherzustellen, soll die App zur Steuerung des barrierefreien Smarthomes unter anderem eine Sprachsteuerung sowie Face-Tracking als Eingabemethode anbieten. In Bezug auf die Eingabemethoden liegt der Fokus auf den Kosten. Durch den weitestgehenden Verzicht auf Hilfsmittel-Hardware, wird versucht diese niedrig zu halten. Hierzu wird das Face-Tracking über die Mobile Vision API von Google realisiert (vgl. [Goo16b]). Sie kann über die Frontkamera eines Android Geräts das Gesicht des Benutzers ohne zusätzliche Hardware verfolgen.

Sowohl die Eingabemethoden und von diesen insbesondere das Face-Tracking als auch die Benutzeroberfläche der App sollen nach der Fertigstellung des Prototyps evaluiert werden. Dabei soll zum einen die Bandbreite der Eingabemethoden ermittelt werden und zum anderen inklusive der Benutzeroberfläche eine qualitative Bewertung durch Personen aus der Zielgruppe erfolgen.

1.3. Vorgehensweise

Zur Erreichung der im Abschnitt zuvor genannten Ziele, findet zunächst eine Definition des Begriffs Smarthome statt. Anschließend werden die Grundlagen der Querschnittslähmung und die damit einhergehenden Einschränkungen für die Betroffenen erläutert.

Der nächste Schritt ist eine Analyse des Forschungsstandes zu barrierefreien Smarthomes für Benutzer mit Mobilitätseinschränkungen an den Händen. Bedingt durch die Ziele liegt der Fokus hierbei auf mobilen Endgeräten und dem Face-Tracking.

Die darauf folgende Benutzungskontext-Analyse besteht aus Vorbereitung, Durchführung und Auswertung der Interviews mit Personen aus der Zielgruppe des barrierefreien Smarthomes. Die Interviews dienen zum einen zur Ideensammlung im Hinblick auf die Benutzeroberfläche und die Eingabemethoden. Sie dienen andererseits dazu, einen besseren Eindruck von der Lebenssituation der Betroffenen zu bekommen, um daraus verschiedene Anwendungsszenarien für das barrierefreie Smarthome zu entwerfen. Im nächsten Schritt findet anhand der dadurch gewonnen Erkenntnisse die Anforderungsdefinition statt.

Basierend auf den Anforderungen und dem Wissen über die Einschränkungen in Folge einer Querschnittslähmung erfolgt darauf die Konzeptionierung einer Anwendung zur Steuerung des barrierefreien Smarthomes sowie der Eingabemethoden, mittels derer sie sich von den Benutzern barrierefrei bedienen lässt. Nach der Erstellung des Konzepts wird dieses in Form eines Prototyps umgesetzt. Die Konzept-Realisierung startet mit der Auswahl geeigneter Technologien und Systeme für die zur Implementation ausgewählten Anwendungsszenarien. Danach werden die schon im Konzept entworfenen Komponenten verfeinert und im nächsten Schritt implementiert.

Nach der Fertigstellung des Prototyps wird dieser evaluiert. Hierzu gibt es zum einen quantitative und zum anderen qualitative Benutzbarkeitstests. Erstere ermitteln die Bandbreite der Eingabemethoden, um die barrierefreien mit den herkömmlichen, wie zum Beispiel Touch, vergleichen zu können. Um von jeder Eingabemethode deren maximale Leistungsfähigkeit zu erhalten, erfolgt die Durchführung der quantitativen Benutzbarkeitstests mit unversehrten Probanden. Hingegen handelt es sich bei den qualitativen Benutzbarkeitstests um Testpersonen die entsprechend der Zielgruppe Mobilitätseinschränkungen an ihren Händen aufweisen. Sie sollen die Benutzeroberfläche sowie die barrierefreien Eingabemethoden des Prototyps auf seine Alltagstauglichkeit hin bewerten.

Abschließend erfolgt eine Zusammenfassung, welche die gewonnen Ergebnisse aus den Benutzbarkeitstests zusammenfasst und einen Übersicht über den Realisierungsstand der einzelnen Anforderungen im Prototyp gibt. Des Weiteren wird der Mehrwert gegenüber den Android Bedienungshilfen und anderen barrierefreien Eingabemethoden aufgeführt sowie ein Ausblick auf mögliche Erweiterungen des Konzepts und Prototyps gegeben.

2. Grundlagen und verwandte Arbeiten

Das Kapitel Grundlagen und verwandte Arbeiten definiert zunächst den Begriff Smarthome. Im Anschluss daran geht es auf die Thematik Querschnittlähmung ein. Der Schwerpunkt dabei liegt auf ihren Ausprägungen und welche Einschränkungen diese jeweils zur Folge haben, um den Bedarf an unterschiedlichen Eingabemethoden in den folgenden Kapiteln nachvollziehen zu können. Des Weiteren stellt es die Bedienungshilfen des Betriebssystems Android vor, die eine barrierefreie Bedienung von diesem ermöglichen sollen. Das letzte Unterkapitel fasst verwandte Arbeiten zusammen, um einen Überblick über den Stand der Forschung zu geben.

2.1. Smarthome

Die Bezeichnung Smarthome umfasst mehrere Forschungsbereiche. Zu diesen gehören unter anderem die Heimautomatisierung sowie das Selbständige Wohnen. Obwohl der Begriff Smarthome folglich interdisziplinär ist, steht er für ein gemeinsames Ziel, das Leben im Zuhause komfortabler und qualitativer zu machen. Gelingen soll das durch die Integration von Technologien und Services. (vgl. [Pec14, S. 212] u. [Sma16])

Was ein Smarthome ausmacht, ist nicht seine Architektur beziehungsweise das Erscheinungsbild und eine ressourcenschonende Ausstattung, wie zum Beispiel Solaranlagen, sondern die interaktiven Technologien. Dennoch treffen erstere Faktoren häufig auf Smarthomes zu (vgl. [Har03, S. 1-2]). Zu den interaktiven Technologien zählen beispielsweise drahtlose Energie- und Datennetzwerke, intelligente und variable Zuschnitte der Räume sowie Sensoren (vgl. [Pec14, S. 212]). Diese und weitere unterstützen die Bewohner eines Smarthomes bei den gewöhnlichen Arbeiten in diesem sowie dessen Umfeld (vgl. [Pec14, S. 212]). Typische Merkmale dieser Unterstützung sind Komfort als auch Interaktivität (vgl. [Pec14, S. 212]).

Jedoch besteht ein Smarthome nicht nur aus den innerhalb des Gebäudes oder auch Grundstücks installierten Systemen und Anwendungen. Vielmehr umfasst es auch die Benutzung von Diensten in Form von Anwendungen, deren Ausführung zum Beispiel in einem Rechenzentrum erfolgt oder - allgemein ausgedrückt - die Vernetzung mit räumlich betrachtet entfernten Systemen. Sprich ein Smarthome ist keine Inselanwendung, die sich auf das jeweilige Gebäude beschränkt. (vgl. [Sma16])

2.2. Querschnittlähmung

Dieses Unterkapitel beschreibt zunächst den Aufbau der Wirbelsäule. Aufbauend darauf führt es in die unterschiedlichen Formen einer Querschnittlähmung und geht abschließend noch auf die Fallzahlen ein.

2.2.1. Aufbau der Wirbelsäule

Die Wirbelsäule besteht aus insgesamt 33 Wirbeln. Sie unterteilt sich ausgehend von oben in die Hals-, die Brust- und die Lendenwirbelsäule sowie 5 Kreuzwirbeln und das Steißbein, bestehend aus 4 Steißwirbeln. Die Halswirbelsäule setzt sich aus 7 Wirbeln zusammen, welche als C1-C7 bezeichnet werden. Die anschließende Brustwirbelsäule besitzt die 12 Wirbel T1-T12. Die Lendenwirbelsäule besteht aus den 5 Wirbeln L1-L5. Die nach ihr folgenden 5 Kreuzwirbel tragen die Bezeichnungen S1-S5. (vgl. [NW11, S. 158])

Die einzelnen Wirbel schützen das Rückenmark, welches in ihnen verläuft. Sie bilden den Spinalkanal. Entlang der Wirbelsäule treten 31 Nervenpaare aus. Nervenpaare deshalb, da unter jedem Wirbelbogen jeweils ein Nerv zur linken sowie einer zu rechten Seite den Spinalkanal verlässt. Im Detail setzen sich die 31 Nervenpaare aus 8 zervikalen, welche die Halswirbelsäule verlassen, 12 thorakalen zur Brustwirbelsäule gehörenden, jeweils 5 lumbalen und sakralen Nervenpaaren die zur Lendenwirbelsäule beziehungsweise den Kreuzwirbeln gehören sowie einem kokzygealen Nervenpaar des Steißbeins zusammen. (vgl. [NW11, S. 159])

Grundsätzlich haben die Nervenpaare jeweils dieselbe Bezeichnung wie der Wirbel, unter dem sie den Spinalkanal verlassen. Da das Hinterhaupt jedoch auch noch als Halswirbel gilt, aber als C0 gezählt wird, verlassen die zervikalen Nervenpaare C1-C8 den Spinalkanal jeweils oberhalb der gleichnamigen Wirbel. Das 31. Nervenpaar von oben betrachtet verlässt den Rückenmarkskanal auf Höhe des Steißbeins und besizt die Bezeichnung kokzygealer Nerv. (vgl. [NW11, S. 158-159] u. [Org13, S. 5])

Die Abbildung 2.1 veranschaulicht den beschriebenen Aufbau der Wirbelsäule. Die grauweißen Elemente stellen die Wirbelkörper dar und beinhalten ihre Bezeichnung. Die roten Linien zwischen ihnen sind die Nervenpaare. Ihre Bezeichnung befindet sich jeweils am rechten Linienende. Am linken Rand der Abbildung befinden sich die Bezeichnungen für die Wirbelsäulenabschnitte und am rechten jene zur Gruppierung der Nervenpaare.

2.2.2. Formen der Querschnittlähmung

Es gibt unterschiedliche Formen der Querschnittlähmungen. Bei der kompletten Querschnittlähmung ist unterhalb der Verletzungshöhe keine Nervenfunktion mehr vorhanden. Die Betroffene Person hat in Folge dessen von dieser Stelle abwärts keine sensorischen und motorischen

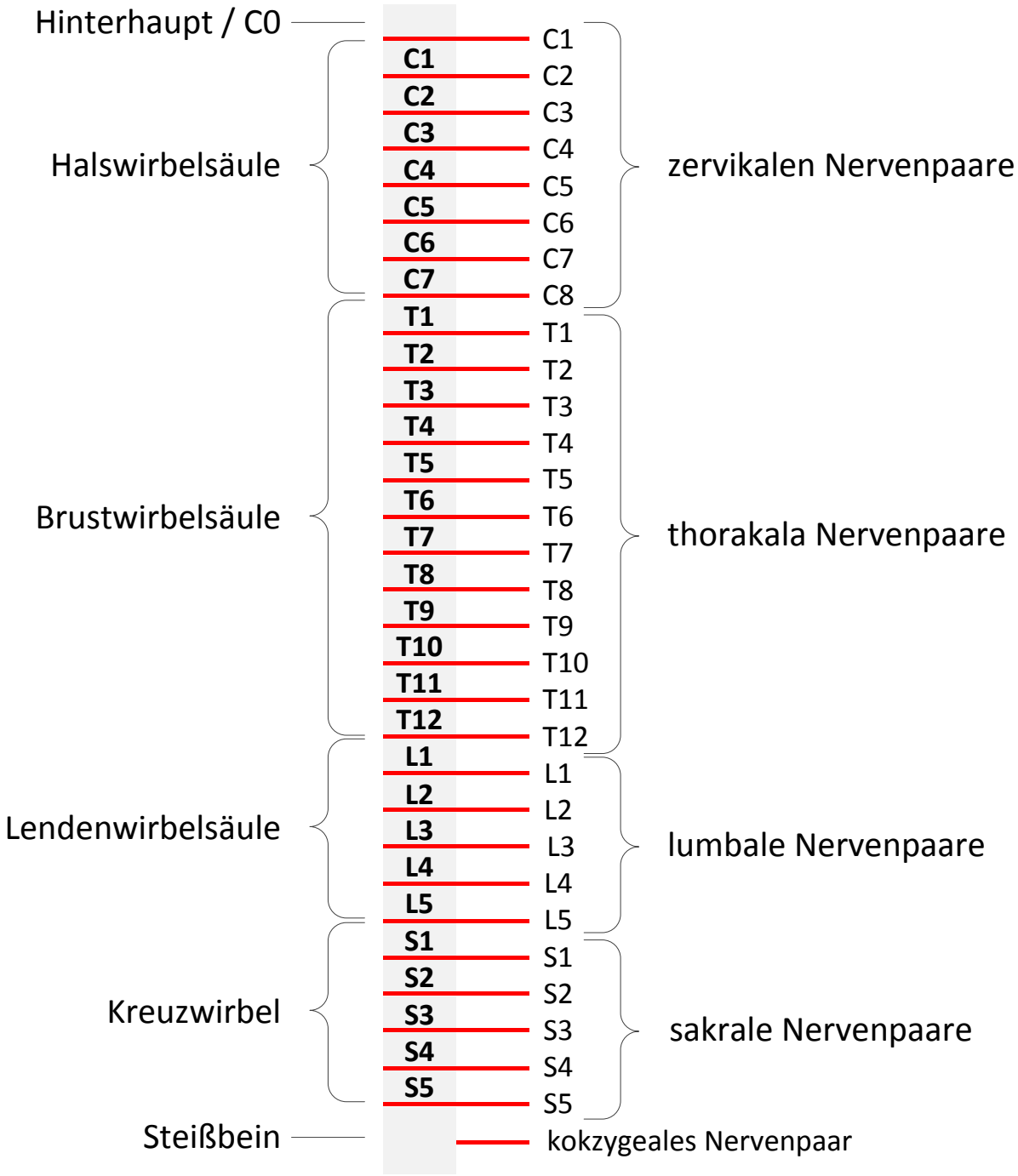

Abbildung 2.1.: Aufbau der Wirbelsäule in Anlehnung an [Org13, S. 5].

Fähigkeiten mehr. Eine komplette Querschnittlähmung wird auch als Querschnittsplegie bezeichnet. Eine inkomplette Querschnittlähmung, beziehungsweise Querschnittsparese liegt vor, wenn unterhalb der Verletzungshöhe noch eine Restfunktion vorhanden ist. Zusätzlich findet eine Unterscheidung durch die Verletzungshöhe statt. Sind die Nervenpaare C1 bis T1 von dieser betroffen beziehungsweise die oberen Extremitäten in Folge dessen eingeschränkt, ist die Rede von einer Tetraplegie beziehungsweise bei einer inkompletten Querschnittlähmung von Tetraparese. Kommt es zur einer Verletzung der Nervenpaare T2-S5, wird diese Form der Querschnittlähmung Paraplegie beziehungsweise Paraparese genannt. (vgl. [AA14])

Aus dem vorherigen Absatz lässt sich schlussfolgern, dass die Auswirkungen einer Querschnittlähmung grundsätzlich von der Verletzungshöhe des Rückenmarks abhängig ist (vgl. [Org13, S. 6]). Die folgende Auflistung bietet einen groben Überblick über diese in Abhängigkeit von den Nervenpaaren. Dabei gilt, dass zu den jeweiligen Auswirkungen noch jene hinzu kommen, welche bei einer Verletzung eines weiter unten liegenden Nervenpaares auftreten.

- Bei einer Verletzung der sakralen Nervenpaare S2-S5 kommt es zu Beeinträchtigungen beim Stuhlgang, der Blasenfunktion sowie der sexuellen Funktionalität. (vgl. [Org13, S. 5-6])
- Wenn die Nervenpaare L1-S1 der Lendenwirbelsäule verletzt sind, ist hierdurch die Beweglichkeit und das Empfingen von der Hüfte abwärts eingeschränkt oder nicht mehr gegeben. (vgl. [Org13, S. 5-6])
- Eine Verletzung der thorakalen Nervenpaare T1-T12 beeinträchtigt die Kontrolle des Rumpfes und der Bauchmuskeln sowie das Gefühl in dieser Körperregion. Des Weiteren sind diese Nerven für die Regulierung der Körpertemperatur verantwortlich. (vgl. [Org13, S. 5-6])
- Die Nervenpaare C4-T1 dienen zur Regulierung der Herzfrequenz sowie der Beweglichkeit der Oberarme. Im Detail betrachtet, wirkt sich eine Verletzung von C8 oder T1 auf die Sensorik und Motorik der Finger aus. Ist eines der Nervenpaare C5-C7 verletzt, sind hiervon die Abschnitte der Arme zwischen Ellbogen und Handgelenk betroffen. (vgl. [Org13, S. 5-6])
- Die zervikalen Nervenpaare C1-C4 sind für die Atmung sowie die Bewegung von Kopf und Nacken zuständig, welche durch eine Verletzung beeinträchtigt werden. (vgl. [Org13, S. 5-6])

Die Ursachen für eine Querschnittlähmung sind entweder traumatische oder nichttraumatische Ereignisse. Erstere können aus diversen Arten von Unfällen resultieren. Zweitere hingegen sind die Folge von Krankheiten - unter anderem Infektionen und Tumore - sowie angeborene Behinderungen. (vgl. [Org13, S. 6])

2.2.3. Verbreitung

Laut einem Bericht der Weltgesundheitsorganisation aus dem Jahr 2013 erleiden jährlich weltweit zwischen 250000 und 500000 Menschen eine Querschnittlähmung. Wie viele querschnittgelähmte Menschen auf der Welt zum Zeitpunkt der Erstellung selbigen Berichts lebten, ist laut diesem unbekannt. (vgl. [Org13, S. 14-15])

Wyndaele et al. [WW06] versuchten in ihrer 2006 veröffentlichten Arbeit anhand von Literatur, die bis in das Jahr 1995 zurückreichte sowie älteren Studien einen Überblick über die weltweite Verbreitung von Querschnittlähmungen zu geben. Eines der Ergebnisse davon war, dass es sich bei einem Drittel der gemeldeten Querschnittlähmungen um eine Tetraplegie handelte. Ihre Daten waren allerdings nicht ausreichend für eine weltweite Schätzung. (vgl. [WW06])

Bezogen auf Deutschland kann von ungefähr 1000 neuen Querschnittlähmungen pro Jahr ausgegangen werden. Ahne et al. [AA14] nennen in einem im Jahr 2014 veröffentlichten Artikel die Zahl 1300 bis 1500 (vgl. [AA14]). Spahn spricht hingegen von ca. 1000 neuen Querschnittlähmungen pro Jahr in Deutschland und, dass davon 40% tetraplegische Fälle sind (vgl. [Spa91, S. 206]). Allerdings wurde das entsprechende Buch schon 1991 veröffentlicht, was bedeutet, dass seine Zahlen mindestens 25 Jahre alt sind (vgl. [Spa91, S. 206]).

Exner kommt auf ähnliche Fallzahlen sowie einen ähnlichen Anteil an Tetraplegikern für Deutschland. Zwischen 1976 und 2003 hatten sie einen Anteil von 37% bei den in diesem Zeitraum 33974 behandelten Fällen in deutschen Spezialeinrichtungen zur Behandlung Querschnittgelähmter. Der Tetraplegikeranteil hat sich in den Jahren 1996 bis 2003 zudem nur um 2% reduziert, weshalb Exner zu der Schlussfolgerung kommt, dass das Verhältnis zwischen Tetraplegikern und Paraplegikern auch zukünftig konstant bleibt. (vgl. [Exn04])

Nach Angaben des statistischen Bundesamtes lebten am 31.12.2013 in Deutschland 17031 Menschen, deren schwerste Behinderung eine Querschnittlähmung ist. Eine Unterscheidung zwischen der Verletzungshöhen nimmt es nicht vor. Jedoch geht aus deren Bericht hervor, dass zusätzlich zu diesen in Deutschland noch 22099 Menschen mit Funktionseinschränkungen an beiden Armen sowie 94096 an beiden Armen und Beinen in Deutschland lebten. Des Weiteren waren 493217 Personen von Funktionseinschränkungen an der Wirbelsäule als auch den Gliedmaßen betroffen. Um welche es sich dabei handelte, geht aus den Zahlen nicht hervor. (vgl. [Sta16])

2.3. Bedienungshilfen in Android

Das Betriebssystem Android bietet schon für bestimmte Benutzergruppen barrierefreie oder zumindest barrierearme Bedienungshilfen (vgl. [Goo16k]). Die folgende Auflistung zählt diese für Android 7.0 auf (vgl. [Goo16k]):

- TalkBack ist eine Sprachausgabe für die auf der Benutzeroberfläche dargestellten Inhalte. Des Weiteren gibt es akkustisches Feedback zu Benutzereingaben. TalkBack zielt somit darauf ab, die Bedienung für sehbehinderte und blinde Menschen zu erleichtern. (vgl. [Goo16k])
- Außerdem unterstützt Android eine aktualisierbare Braillezeile. Diese ist zudem in Kombination mit TalkBack verwendbar. Dadurch ist sowohl eine Interaktion mit dem Android-Gerät als auch Texteditierung gewährleistet. (vgl. [Goo16k])
- Die App Voice Access, die sich zum Zeitpunkt der Erstellung dieser Arbeit noch in der Betaphase befindet, soll das Betriebssystem dem Benutzer via Sprachbefehlen zugänglich machen. Mittels diesen ist es möglich, innerhalb von Android und den installierten Apps, zu navigieren und Texteingaben vorzunehmen. Die Sprachsteuerung soll Benutzern, welche ihre Hände nicht mehr zur Bedienung nutzen können, diese ermöglichen. (vgl. [Goo16k])
- Benutzer, die Schwierigkeiten bei der Wahrnehmung der Inhalte haben, finden ebenfalls Unterstützung. Es ist möglich Untertitel zu aktivieren, die Kontrast- und Farbeinstellungen anzupassen sowie die Schrift zu vergrößern. Darüber hinaus ist noch eine Art Lupenfunktion vorhanden. (vgl. [Goo16k])
- Ab Android 5.0 ermöglicht es der Schalterzugriff den Benutzern, welche ihre Hände nur noch eingeschränkt benutzen können, beispielsweise Tetraplegikern mit einer tiefen Verletzungshöhe im Bereich der Halswirbelsäule, mittels Eingabegeräten, welche Tastatursignale senden, die sichtbaren Menüelemente zu scannen und auszuwählen. Bei den Eingabegeräten kann es sich um eine herkömmliche Maus und Tastatur, Hilfsmittel-Hardware oder Tasten des Android-Geräts handeln. Zwischen 1 und 5 Tasten sind mit Funktionen belegbar. Dies ist bei der Gruppenauswahl nützlich, da es je Taster eine Gruppe gibt und sich dadurch die Auswahl des gewünschten Elements beschleunigt. Neben der Gruppenauswahl gibt es noch das automatiche Scannen mit 1 Taster, welches alle Elemente der Reihe nach fokussiert und bei Betätigung des Tasters das fokussierte auswählt. Das Scannen ist aber auch schrittweie möglich, wenn 2 Taster verfügbar sind. Jeweils einer ist dann für den Fortschritt und die Betätigung des fokussierten Elements zuständig. Des Weiteren lassen sich noch spezielle Menüs auswählen, mit deren Hilfe derer weitere Eingaben, wie zum Beispiel das Scrollen, möglich sind. (vgl. [Goo16k], [Goo16g], [Goo16h] u. (vgl. [Goo16j]))

2.4. Verwandte Arbeiten

Aus den zurückliegenden Jahren existiert eine Reihe von Arbeiten, welche sich mit barrierefreien Eingabemethoden für Tetraplegiker sowie vergleichbare Benutzergruppen befassen.

Caltenco et al. führten eine Befragungen von Tetraplegikern zu deren Nutzungsverhalten im Hinblick auf Eingabegeräte und den damit verbundenen Erfahrungen durch. Hierbei zeigte sich, dass die genutzten Eingabemethoden in erster Linie auf die Einbindung des Benutzers seiner Hände verzichten. Diejenigen Benutzer, die noch händische motorische Fähigkeiten besitzen, gaben an Joysticks oder Handsticks in Kombination mit einer Tastatur zu verwenden. Hinsichtlich der Eigenschaften von den verwendeten Eingabemethode ergab die Befragung, dass die Geschwindigkeit von Blick-Trackern, Mund-Joysticks sowie Tastatursticks befriedigend ist, jene von den Mundsticks allerdings nicht. Zu den Blick-Trackern ist das Ergebnis insofern zwiespältig, als dass er einerseits als diskrete und andererseits als unzuverlässige Eingabemethode bewertet wurde. Darüber hinaus sei die Fehlerkorrektur schwierig. Bei den meisten anderen Eingabemethoden hingegen nicht. Auch seien diese zuverlässiger. Dafür haben die Sprachsteuerung sowie über das Kinn bediente Joysticks das Problem, dass sie nicht diskret sind. (vgl. [CBJA12])

Aufgrund der zuvor genannten Ergebnisse der Befragung findet als nächstes eine Betrachtung bisheriger Forschungsarbeiten zu barrierefreien Eingabemethoden statt, welche keine motorischen Fähigkeiten in den Händen erfordern.

Williams et al. testeten zwei barrierefreie Eingabemethoden für Tetraplegiker beziehungsweise Benutzer mit vergleichbaren Einschränkungen als Alternative zur Cursor-Steuerung mittels einer Maus. Dabei handelte es sich zum einen um einen am Kopf des Benutzers befestigten Lagesensors und zum anderen um eine Elektromyografie von 3 Muskeln im Hals- und Kopfbereich. Über Letztere war eine Steuerung des Cursors ähnlich zu einem Joystik möglich. Für die Tests nutzten sie unversehrte Probanden. Messungen erfolgten quantitativ. Als dritte Eingabemethode testeten sie eine herkömmliche Maus. Bei der unter anderem durchgeführten Bandbreitenmessung der 3 Eingabemethoden zeigte sich, dass die Maus die beiden anderen um ca. das Fünffache übertrifft. Der Lagesensor wies eine etwas bessere Bandbreite als die Elektromyografie auf. Zudem stellte Ersterer vom Benutzungserlebnis aus betrachtet den besten Mausersatz. Die Elektromyografie hingegen war schneller, aber erreichte nicht die selbe Präzision wie der Lagesensor. (vgl. [WK08])

Chen entwickelte eine Mausalternative in Form eines Headsets zur Steuerung des Cursors. Hierfür besitzt dieses 2 Neigungssensoren, welche die Bewegungen des Kopfes erkennen und den Cursor, entsprechend horizontal und vertikal bewegen. Über einen Touch-Switch an der Backe kann der Benutzer einen Mausklick durchführen, indem er diese aufbläst. Eine Evaluierung fand mit versehrten sowie unversehrten Probanden statt. Es kam zu keinen erheblichen Abweichungen zwischen diesen beiden Gruppen. Ein Vergleich mit anderen Eingabemethoden fand leider nicht statt. (vgl. [Che01])

Grigorescu et al. entwickelten einen Roboterassistenten für Menschen mit Mobilitätseinschränkungen an den Armen, der sich beispielsweise in Form eines künstlichen Arms an einem Rollstuhl befestigen lässt. Für die Steuerung setzten sie eine nichtinvasive Kopf-Computer-Schnittstelle ein. Die Nichtinvasivität erreichten sie durch 5 in jeweils unterschiedlichen Frequenzen blickende LEDs, die über einen Displayrand verteilt waren. Fokussiert der Benutzer eine von ihnen, findet im Gehirn eine visuelle Stimulation statt, welche sogenannte

Steady State Visually Evoked Potentials Signale auslöst. Diese unterscheiden sich in Folge der verschiedenen Blinkfrequenzen voneinander und sind mittels Elektroenzephalografie messbar. Wenn der Benutzer 1 von den 5 LEDs fokussiert, entspricht dies der Betätigung einer der 4 Pfeiltasten oder der Eingabetaste. Damit ist es möglich, einen Cursor zu steuern und Elemente auszuwählen. (vgl. [GLF+12])

Jeong et al. konstruierten eine Maussteuerung für Tetraplegiker, über das Zusammenbeißen der Zähne in Kombination mit einem kleinen am Kopf des Benutzers befestigten Elektromyographie-Geräts. Beim Zähnezusammenbeißen wird unterschieden, ob dieser nur die linken beziehungsweise rechten oder alle zusammenbeißt. Des Weiteren wird zwischen der Dauer unterschieden. Dadurch ist es möglich, den Mauszeiger um seine eigene Achse zu drehen, ihn in die Richtung zu bewegen, in die er zeigt sowie die linke und rechte Maustaste zu betätigen. Bei einem Test mit lediglich einem Probanden stellte sich heraus, dass dieser im Vergleich zu einer herkömmlichen Maus, für die selben Aufgaben im Durchschnitt ca. die siebenfache Zeit benötigte. (vgl. [JKS05])

Kauhanen et al. fanden in ihrer Studie heraus, dass die Elektroenzephalografie im Vergleich zur Magnetoenzephalographie für eine Kopf-Computer-Schnittstelle besser geeignet ist. Einer der Gründe hierfür ist, dass sie weniger anfällig für magnetisches Rauschen ist. Die Magnetoenzephalographie hätte jedoch den Vorteile, dass sich seine Signale einfacher interpretieren lassen. (vgl. [KNL+06])

Jedoch scheint auch die Elektroenzephalografie nicht bedingungslos geeignet zu sein. Denn Felzer et al. hatten die Motivation eine muskelbasierte Steuerung der Maus für Personen zu entwickeln, die ohne Elektroenzephalografie möglich ist. Ihr Ansatz nutzt die Erkennung einer einzelnen sowie doppelten Kontraktion eines Muskels mittels eines auf ihm angebrachten Sensors, wodurch es weniger Probleme mit Rauschen im Vergleich zur Elektromyografie gibt. Die Verarbeitung der beiden Eingabesignale übernimmt ein endlicher Automat. Durch geschickte Anordnung der Zustände und Übergänge in diesem, gelang es ihnen dadurch, den Mauszeiger in alle 4 Richtungen bewegen sowie 2 verschiedene Mausklicks umsetzen zu können. In einer ersten Studie untersuchten sie, ob potenzielle Benutzer in der Lage sind, das Eingabekonzept zu verstehen. Die Probanden, welche trotz ihrer Einschränkungen das System bedienen konnten, benötigten zwischen 5 und 15 Minuten, bis sie es eigenständig nutzen konnten. (vgl. [FN08])

Lund et al. zeigten in ihrer Arbeit, wie Betroffene ohne den Einsatz ihrer Hände, sondern lediglich ihrer Zunge mittels Ferromagnetismus Eingaben vornehmen können. Dazu entwickelten sie eine zungenbasierte Rollstuhlsteuerung. Sie besteht aus einer Art SZahnspange", welche 18 Sensoren besitzt. 10 in Form von Tasten und 8 als Joystick. Über ein ferromagnetisches Piercing in der Zunge kann sie der Benutzer ansprechen. Die Eingabebefehle werden drahtlos an eine externe Steuereinheit gesendet, welche diese nutzt, um den eigentlichen Joystick des Rollstuhls zu simulieren. (vgl. [LCC+10])

Bian et al. entwickelten eine Maussteuerung anhand der Nasenposition. Des Weiteren nutzten sie den Mund des Benutzers für Gesten. Die Gesten wurde dazu genutzt, um einen Mausklick

zu tätigen oder die Bewegung des Cursors ein- bzw. auszuschalten. Letzteres hat den Vorteil, dass der Bewegungsspielraum der Nase nicht auf die komplette Benutzeroberfläche abgebildet werden muss, sondern der Benutzer nach dem Ausschalten der Cursorbewegung zunächst seine Nase wieder in eine angenehme Position bringen kann, um danach die Cursorbewegung wieder einzuschalten und diesen weiter in Richtung Ziel bewegen zu können. Ein weiterer Vorteil hiervon ist, dass keine Kalibrierung notwendig ist. Die Verwendung einer Tiefenkamera hatte den Vorteil, dass sie unabhängig von den Lichtverhältnissen ist, im Gegensatz zu einer RGB Kamera. Bei einem Leistungstest zeigte sich, dass je nach verwendeter Mundgeste zur Ausführung des Mausklicks, die Probanden ungefähr doppelt so lange für die Testaufgaben benötigten, wie mit einer herkömmlichen Maus. (vgl. [BHCM14])

Aus Abschnitt 2.2.2 geht hervor, dass bei einer niedrigeren Verletzungshöhe, die Tetraplegie nicht ganz so gravierend ist, wodurch die Betroffenen in manchen Fällen noch eingeschränkte motorische Fähigkeiten besitzen. Aus diesem Grund gibt es neben den zuvor aufgeführten handlosen Eingabemethoden Ansätze, die verbliebenen motorischen Fähigkeiten bestmöglich zu nutzen.

Ueno entwickelte dazu eine Benutzerschnittstelle bestehend aus 2 Tastern zur Steuerung des Mauscursors auf einem Smartphone. Mit einem der beiden Taster kann der Benutzer zwischen 8 Modi wechseln und mit dem zweiten innerhalb von diesen eine Aktion ausführen. Über LEDs erhält der Benutzer unter anderem Feedback zu den ausgewählten Modi. In einem Modus ist beispielsweise die Bewegung des Mauscursors steuerbar. Die Aktion dieses Modus ist, dass der Mauszeiger die Richtung jeweils um 90 Grad im Uhrzeigersinn ändert. Die 7 anderen Modi gibt es unter anderem um eine Maustaste zu betätigen. (vgl. [Uen14])

Tanimoto et al. konstruierten eine Maus für Tetraplegiker mit einer Querschnittlähmung im unteren Halswirbelbereich. Dadurch, dass sie noch ihre Schultern und Arme bewegen können, ist für sie beispielsweise die Benutzung einer Ballmaus möglich. Jedoch benötigen sie dafür zum Teil beide Hände oder sind beim Bewegen des Eingabegerät in Kombination mit Klickoperationen langsam. Um die Eingabegeschwindigkeit zu erhöhen, kombinierten sie eine Switch-Box mit einem optischen Maussensor. Über erstere führt der Benutzer mit einer Hand die Mausklicks aus. Zweitere erkennt die Bewegung des Benutzers zweiter Hand, worüber sich der Cursor bewegen lässt. (vgl. [TRF+05])

Für manche Benutzer mit eingeschränkten motorischen Fähigkeiten an den Händen ist es trotzdem möglich, ein Touchdisplay als Benutzerschnittstelle zu verwenden.

Um sie dabei unterstützen zu können evaluierten Froehlich et al. 5 verschiedene Interaktionskonzepte an Touchdisplays mit einer erhöhten Umrandung. Diese erleichtert die Auswahl von Menüelemente an den Rändern und Ecken mittels eines Stylus, da ihn diese abbremsen. Eines der Interaktionskonzepte war eine normale Touch-Eingabe. Die anderen 4 waren speziell auf die Umrandung ausgelegt. Bei der statistischen Auswertung, konnte keines hervorstechen. Es zeigte sich jedoch, dass 2 der versehrten Probanden von einem Interaktionskonzept besonders profitierten. Bei diesem befinden sich die Buttons jeweils in den Display-Ecken. Ihre Betätigung erfolgt, indem der Benutzer mit dem Stylus auf dem Touchdisplay in die entsprechende

Ecke wischt und ihn danach anhebt. Diese Verfahren war deshalb so erfolgreich, da die Hand während der Eingabe sowohl vom Touchdisplay als auch der Ecke stabilisiert wird. Auch sind Fehlerkorrekturen durch Ändern der Richtung einfach möglich. (vgl. [FWK07])

Guerreiro et al. untersuchten in ihrer Studie unterschiedliche Eingabemethoden für Tetraplegiker an einem mobilen Touchgerät. Schwerpunkt war die Bestimmung der Zielgröße sowie -position auf dem Display. Bei den Eingabemethoden handelte es sich um die klassische Touch-Eingabe sowie die Gesten Ziel durchwischen, ausschneiden sowie wischen in eine bestimmte Richtung. Bei dem Ziel handelte es sich um einen Kreis mit wahlweise 7, 12 und 17mm Durchmesser. Die Studie kommt zu dem Ergebnis, dass für Tetraplegiker bei allen Eingabemethoden bis auf das Ausschneiden eine Zielgröße von 12mm ausreichend ist. Für Letztere sind 17mm erforderlich. Nichtsdestotrotz war die Fehlerrate mehr als doppelt so hoch, wie in vergleichbaren Studien mit unversehrten Probanden. Die Zielposition auf dem Display hatte keine Auswirkungen auf die Fehlerhäufigkeit. Jedoch bietet die Displayumrandung Stabilität, welche die Genauigkeit positiv beeinflusst. Das klassische Touch sowie das Durchwischen des Ziels waren am erfolgreichsten. Die Mehrheit der Probanden würde Ersteres bevorzugen. Zusammenfassend betrachtet, kamen Guerreiro et al. zu der Schlussfolgerung, dass eine gemeinsame touchbasierte Benutzerschnittstelle für unversehrte Benutzer und jenen mit motorischen Einschränkungen an den Händen entwickelbar ist. Des Weiteren können Tetraplegiker mittels herkömmlichen Touch Ziele am unteren Displayrand sowie in der Nähe ihres bevorzugten Arms leichter auswählen. Einen erheblichen Einfluss hat zudem die Position des Displays. (vgl. [GNJG10a] u. [GNJG10b])

Da das Eye-Tracking und Blickgesten jedoch nicht nur von Tetraplegikern sowie ähnlich eingeschränkten Benutzern verwendet wird, gibt es eine Reihe von Arbeiten, die sich mit diesen Eingabemethoden ohne den Kontext Barrierefreiheit beschäftigen und dennoch für die Herstellung von dieser hilfreich sein können.

Lee et al. entwickelten in 2012 einen Eye-Tracker für ein Smartphone. Hierfür befestigten sie eine Webcam mit Zoomlinse und 3 Infrarot LEDs an diesem. Dies war nötig, da die damals übliche VGA-Auflösung nicht ausreichend war. Des Weiteren erfolgte die Bildverarbeitung nicht auf dem Smartphone sondern einem Computer. Zusätzlich zur Blickverfolgung konnten sie unter anderem auch eine Blinzelerkennung umsetzen. In der Studie erreichten die Probanden mit dem Eye-Tracker auf einer Benutzeroberfläche des Smartphones mit 5x4 Feldern zudem eine Trefferquote von 94,6%. (vgl. [EM13])

Drewes et al. analysierten in einer Benutzerstudie, wie sich Eye-Tracking zur Bedienung von Mobiltelefonen nutzen lässt. Diese stellen im Vergleich zum Eye-Tracking an Computern eine Herausforderung hinsichtlich der Lichtintensität sowie Kalibrierungsproblemen. Aus diesem Grund testen sie neben dem klassischen Eye-Tracking mit Auswahl durch Verweilen auch ein Verfahren, welches lediglich aus Blickgesten Eingabebefehle ableitete. Da hierfür nur die relativen Koordinaten erforderlich sind, ist eine Kalibrierung nicht erforderlich. Des Weiteren ist der Gestenansatz hardwareressourcenschonender. In der Benutzerstudie hatte keiner der Probanden größere Schwierigkeiten bei der Nutzung der Blick-Gesten. Jedoch gab die Mehrheit

von ihnen an, das klassische Eye-Tracking zu bevorzugen. Dies erfolgte über eine externe Kamera ebenso fand die Verarbeitung nicht auf dem Mobiltelefon statt. (vgl. [DLS07])

Hingegen schlagen Dybdal et al. basierend auf den Ergebnissen ihrer Studie vor, dass Gesten die geeignetste Auswahlart für berührungslose Eingaben auf kleinen Bildschirmen sind. In dieser analysierten sie, ob Verweilen oder eine Geste zur Auswahl auf einem mobilen Endgerät geeigneter ist. Hierzu testeten sie beide mittels Touch, Gaze-Tracking sowie dem Accelerometer. Zweiteres erfolgte über eine externe Kamera und Computer. Touch war bei beiden Auswahlarten am schnellsten und fehlerarmsten. Beim Gaze-Tracking war die Geste hingegen das bessere Auswahlverfahren, insbesondere bei kleinen Zielen. Zu berücksichtigen ist, dass die Probanden Gesten anstrengender empfanden. (vgl. [DAH12])

Kangas et al. fanden in einer Benutzerstudie heraus, dass sich Blickgesten auf mobilen Endgeräten mittels vibrotaktilen Feedback als Eingabebestätigung verbessern lassen. Hierzu nutzten sie einen Gaze-Tracker. Die Gestenerkennung erfolgte auf einem Computer, welcher diese anschließend an ein Smartphone weiterleitete. Aus der Befragung der Probanden ging hervor, dass die Testaufgaben durch das vibrotaktile Feedback einfacher und komfortabler zu lösen waren, als ohne. Die Auswertung der Testergebnisse ergab zusätzlich, dass ohne Feedback bis zu 15% mehr Blickgesten erforderlich waren. (vgl. [KAR+14])

Agustin et al. zeigten in ihrer Studie, dass ein guter Eye-Tracker nicht teuer sein muss. Dazu verglichen sie einen mit einer 20$ Webcam, mit eingebauten Infrarot-LEDs selbstgebauten Eye-Tracker mit 2 kommerziellen sowie einer optischen Maus. Der Vergleich erfolgte via Eye-Typing und Zielauswahl. In Ersterem erzielten alle 4 Eingabegeräte nahezu die selben Werte. Bei der Zielauswahl hingegen, erreichte der selbstgebaute Eye-Tracker den höchsten Durchsatz. (vgl. [SSHH09])

Bulbul et al. entwickelten einen Face-Tracking Algorithmus, der mit den Einschränkungen von mobilen Geräten zurechtkam. Als Interaktionsbeispiel entwickelten sie unter anderem eine Anwendung, die es ermöglichte mittels Kopfbewegungen durch ein Bild zu scrollen. (vgl. [BCC])

Die zuvor genannten Arbeiten nutzten größtenteils zusätzlich Hardware, um eine barrierefreie Eingabemethode für Tetraplegiker anzubieten. Ähnlich verhält es sich bei den beschriebenen Studien zum Eye- und Face-Tracking. Diese nutzten oftmals externe Kameras bzw. Tracker. Jedoch zeigten zum einen Augustin et al. mit ihrer Arbeit, dass ein leistungsfähiges Eye-Tracking keine teure Kamera voraussetzt und Bulbul et al. veranschaulichten mit ihrem Face-Tracking Algorithmus, dass sich dieses schon vor mehreren Jahren auf einem mobilen Endgerät realisieren lies (vgl. [SSHH09] u. [BCC]). Aufbauend darauf wird in den folgenden Kapiteln dieser Arbeit versucht, eine Anwendung zur Steuerung von einem barrierefreien Smarthome für Tetraplegiker mittels Face-Tracking zu realisieren, welche auf einem mobilen Endgerät lauffähig ist, auf zusätzliche Hardware verzichtet und damit den Ansatz mit den niedrigen Kosten von Agustin et al. aufnimmt (vgl. [SSHH09]).

3. Benutzungskontext-Analyse

Dieses Kapitel beschreibt zunächst die Interviews, die zur Ideenfindung sowie zur Unterstützung bei der Definition der Anforderungen, durchgeführt wurden. Im Anschluss daran erfolgt diese für eine Anwendung zur Steuerung eines barrierefreien Smarthome für Menschen mit Mobilitätseinschränkungen an ihren Händen.

3.1. Interviews

Interviews sind ein zentrales Element innerhalb von Forschungsprojekten und haben somit einen erheblichen Einfluss auf den Erfolg oder auch das Scheitern eines solchen Projektes. Besonders wichtig ist es, geeignete Interviewpartner auszuwählen sowie diese im weiteren Verlauf richtig zu interviewen. (vgl. [KGM12, S. 83])

Folglich behandelt dieses Unterkapitel zunächst die Auswahl der Interviewpartner sowie die Erstellung des Fragebogens für die Interviews. Des Weiteren berichtet es über deren Verlauf und schließt mit ihrer Auswertung ab.

3.1.1. Auswahl der Interviewpartner

Die Auswahl der richtigen Interviewpartner ist wichtig, denn das Interviewen von Personen ist nur zielführend, wenn diese auch der Zielgruppe des späteren Produktes, also dem barrierefreien Smarthome für Menschen mit Mobilitätseinschränkungen im Bereich der Hände angehören (vgl. [KGM12, S. 83]). Des Weiteren bietet es sich an, die ausgewählten Interviewpartner nicht nur in die Phase der Anforderungsdefinition mit einzubeziehen, sondern auch in die Benutzbarkeitstests zu einem späteren Zeitpunkt, wodurch sich der Aufwand für die Suche nach Probanden reduziert. Für die in Kapitel 6 beschriebenen Benutzbarkeitstests ist es wichtig, Probanden zu haben, welche im Umgang mit vergleichbaren Anwendungen vertraut sind, um evaluieren zu können, welche Vor- und Nachteile das barrierefreie Smarthome gegenüber diesen besitzt (vgl. [KGM12, S. 85]). Aus den genannten Gründen wurden für die Interviews im Hinblick auf die Benutzbarkeitstests Interviewpartner beziehungsweise Testpersonen gesucht, welche ihre Hände nicht mehr benutzen können. Hierzu bot sich die Vorgehensweise von Kuniavsky an, zunächst die Zielgruppe zu definieren, im Anschluss daran Probanden zu suchen, welche dieser angehören und sie zur Teilnahme zu überzeugen (vgl. [KGM12, S. 84]).

Definition der Zielgruppe

Die Zielgruppendefinition stellt sicher, dass sich ein Projekt beziehungsweise ein Produkt nicht in die falsche Richtung entwickelt, indem es von Interviewpartnern beeinflusst wird, auf die es gar nicht abzielt. Vermeiden lässt sich das durch eine präzise Definition der zukünftigen Benutzer. Dies erfolgt idealerweise ausgehend von demographischen Rahmenbedingungen sowie bisherigen Erfahrungen im Umgang mit ähnlichen Produkten und engt die Zielgruppe darüber hinaus weiter ein. (vgl. [KGM12, S. 84-85])

Folgende Kriterien definierten die Zielgruppe für die Interviews:

- Volljährigkeit: Sie erleichtert die verwaltungstechnische Abwicklung bezüglich dem Datenschutz.
- Tetraplegie im Bereich C1-T1, sodass die Hände nicht mehr zur Bedienung von elektronischen Geräten benutzbar sind (vgl. [Org13, S. 5-6]). Es kommen jedoch genauso Personen mit vergleichbaren motorischen Einschränkungen in Frage.
- Keine bis wenig geistige Einschränkungen, um möglicherweise daraus resultierende Verständnisprobleme bei den späteren Benutzbarkeitstests ausschließen zu können. Des Weiteren muss es möglich sein, im Zuge des Interviews ein Gespräch führen zu können.
- Eine grundsätzlich offene Einstellung gegenüber neuen Technologien, wie zum Beispiel Smartphones und Tablets.
- Im Hinblick auf die Benutzbarkeitstests ist es von Vorteil, wenn die Person ihren Kopf bewegen (insbesondere drehen) und mit den Augen blinzeln kann. Da die Nervenpaare im Bereich C1-C4 für die Kopfbewegungen verantwortlich sind, sollte die Verletzung bei den Probanden wiederum auch nicht zu hoch sein (vgl. [Org13, S. 5]).

Ursprünglich war es geplant, dass die Zielgruppe des barrierefreien Smarthomes Tetraplegiker sind, die ihren Kopf noch bewegen können. Im Rahmen der im nächsten Abschnitt beschriebenen Suche nach geeigneten Interviewpartnern stellte sich jedoch heraus, dass viele Personen zwar ihre Hände nicht mehr hinreichen nutzen können, aber nach medizinischen Kriterien keine Querschnittlähmung haben. Stattdessen sind ihre Einschränkungen auf andere Krankheiten und / oder Verletzungen zurück zu führen. Aus diesem Grund ist in der Definition der Zielgruppe nicht ausschließlich von vom Hals ab Querschnittgelähmten die Rede, sondern zusätzlich auch von jenen, die ihre Hände nicht mehr benutzen können.

Suche nach den Interviewteilnehmer

Obwohl mehrere Einrichtungen angefragt wurden, gestaltete sich die Suche zunächst schwierig. Zum einen antworteten nicht alle Angefragten und zum anderen hatten auch nicht alle solche Personen in ihren Einrichtungen / Verbänden, welche die Kriterien der Zielgruppendefinition erfüllten. Letztendlich fanden sich 5 Personen, die im Groben die Kriterien der Zielgruppendefinition erfüllten und bereit zur Teilnahme an einem Interview waren.

Da die 5 Interviewpartner die Kriterien der Zielgruppendefinition nur bedingt erfüllten, war eine Lockerung von diesen notwendig. Der Grund ist darauf zurückzuführen, dass nur 1 von den 5 Personen ihre Hände aufgrund einer Querschnittlähmung vom Hals ab nicht mehr benutzen kann. Bei den 4 anderen Personen und damit dem Großteil, ist jeweils bei 2 eine Krankheit oder eine schwere Kopfverletzung in der Vergangenheit ursächlich. Bei letzterer Ursache ist es so, dass durch die Schwere der Verletzung nicht nur Mobilitätseinschränkungen sondern auch geistige Einschränkungen die Folge sind. Dies führte zu der Erkenntnis, dass die Zielgruppe eines barrierefreien Smarthomes für Benutzer mit einer Querschnittlähmung von Hals ab, eigentlich viel größer ist, da auch Menschen davon profitieren würden, die ihre Hände aus anderen Gründen als einer Querschnittlähmung, nicht mehr benutzen können. Des Weiteren war es sinnvoll, das ursprüngliche Kriterium „keine geistige Einschränkungen“ insoweit zu lockern, dass Interviews noch möglich sind, aber im Gegenzug nicht alle Interviewpartnern auch für die zum späteren Zeitpunkt stattfindende Bandenbreitenmessung der unterschiedlichen Eingabemethoden geeignet sind und für diese deshalb noch zusätzliche Probanden gesucht werden müssen. Eine solche Aufsplittung in unterschiedliche Gruppen, um sicherzustellen, dass es überhaupt potenzielle Interviewpartner beziehungsweise Testpersonen gibt, empfiehlt auch Kuniavsky (vgl. [KGM12, S. 87]).

Kuniavsky empfiehlt zu dem, im Vorfeld der Interviews zunächst ein paar Basisinformationen über die Interviewpartner mittels eines per E-Mail zugesandeten Fragebogens zu ermitteln (vgl. [KGM12, S. 88]). Diesen Zwischenschritt hat der Autor ausgelassen, da er für die betroffenen Personen einen zu hohen Aufwand dargestellt hätte. Stattdessen werden diese Informationen am Anfang des im folgenden Unterkapitels beschriebenen Fragebogens erhoben.

Ebenfalls weggefallen ist das von Kuniavsky vorgeschlagene Screening zur Auswahl der von der zur Verfügung stehenden Interviewpartnern am besten geeignetsten (vgl. [KGM12, S. 94-95]). Die Gründe hierfür waren, dass bei lediglich 5 Interviewpartnern eine engere Auswahl wenig zielführend gewesen wäre und der Autor zudem davon ausgehen konnte, dass sie wegen ihrer unmittelbaren Betroffenheit ausreichend motiviert sind.

3.1.2. Erstellung des Fragebogens für die Interviews

Der Fragebogen orientiert sich an der von Kuniavsky vorgeschlagenen Interviewstruktur, welche verbildlicht mit der Form einer Sanduhr vergleichbar ist. Diese sieht vor, dass ein Interview mit allgemeinen Fragen startet und im weiteren Verlauf die primäre Forschungsfrage immer weiter konkretisiert. Abschließend folgen nochmals breiter aufgestellte Fragen, um einen Überblick und eine Zusammenfassung zu erhalten. Im Detail handelt es sich dabei um folgende 6 Phasen: Einleitung, Aufwärm-Phase, Allgemeine Fragen, Schwerpunkt-Fragen, Rückblick sowie Abschluss. (vgl. [KGM12, S. 117-118])

Im Folgenden wird der Aufbau des Fragebogens anhand der 6 Phasen detaillierter beschrieben. Der Fragebogen befindet sich zusätzlich im Anhang A.1 dieser Arbeit:

- Einleitung:
  Die Einleitung beinhaltet die Vorstellung des Interviewers. Des Weiteren soll sie sicherstellen, dass dieser neutral wahrgenommen wird. (vgl. [KGM12, S. 118]

  Hierzu beinhaltet der Fragebogen zunächst einmal die Vorstellung des Autors und Ablaufs. Darüber hinaus werden in dieser Phase auch die Formalitäten abgehandelt. Dies umfasst die Unterzeichnung der Einverständniserklärung durch den jeweiligen Interviewpartner sowie die Auszahlung der Aufwandsentschädigung an diesen.

- Aufwärm-Phase:
  Die Aufwärm-Phase soll den Interviewpartner in die Situation des Interviewtwerdens einführen und dessen Aufmerksamkeit auf das Themengebiet des Interviews lenken (vgl. [KGM12, S. 118]). Als Einstieg wird in dieser Phase die noch ausstehende Ermittlung der Basisinformationen gewählt. Bei diesen handelt es sich primär um demographische Daten sowie der Frage, ob dieser schon einmal an einem Interview teilgenommen hat. Laut Kuniavsky ist das wichtig zu wissen, da Personen, die schon einmal an einer Befragung teilgenommen haben, dazu tendieren, die Untersuchung mittels ihren Antworten zu unterstützen, was zu einer Verfälschung des Ergebnisses führen kann (vgl. [KGM12, S. 101]).

  Im Anschluss daran bilden Fragen zu bisherigen Nutzung von elektronischen Geräten den Übergang zum Thema Smarthome. Konkret geht es darum, welche Hardware und Software der Interviewte bisher nutzt und wie intensiv er das tut. Auch geht der Fragebogen in dieser Phase darauf ein, welche der verwendeten Geräte und Programme speziell auf die Bedürfnisse der jeweiligen Behinderung(en) abgestimmt sind.

- Allgemeine Fragen:
  Diese Phase lenkt den Fokus auf das eigentliche Thema des Interviews, also das Produkt oder Forschungsprojekt zu dem der Interviewte befragt wird (vgl. [KGM12, S. 118]).

  Der Fragebogen trägt dem Rechnung, indem er zunächst einmal das Hintergrundwissen des Interviewpartners und seine Erfahrungen auf dem Gebiet der Smarthomes ermittelt. Darüber hinaus führt er das Thema Barrierefreiheit mit sehr offen formulierten Fragen im Bezug darauf, wie ein ideales Smarthome aus Sicht der interviewten Person auszusehen hätte, ein.

- Schwerpunkt-Fragen:
  Die Schwerpunkt-Fragen eines Interviews gehen im Detail auf das Produkt oder die Idee ein, indem dem Interviewpartner Fragen zu einzelnen Aspekten, beispielsweise Funktionalitäten, gestellt werden und er um seine eigene Einschätzung gebeten wird (vgl. [KGM12, S. 118]).

  Für die Interviews im Rahmen dieser Arbeit bedeutet dies, dass der Moderator in dieser Phase Fragen zu der Bedienung sowie Hardware eines barrierefreien Smarthomes stellt. Im Bereich der Bedienung geht es darum, festzustellen, ob es unter den Interviewpartnern eine bevorzugte Eingabemethode gibt und was aus ihrer Sicht die Vor- und Nachteile

einer Sprachsteuerung und des Face-Trackings sind. Darüber hinaus gibt es Fragen zu der Menüführung. Bezüglich der Hardware konzentriert sich der Fragebogen auf das Finden einer geeigneten Platzierung für das Tablet / Smartphone, das die Anwendung zur Steuerung des barrierefreien Smarthomes ausführt sowie um die Displaygröße.

Weiterhin geht der Fragebogen in der Phase Schwerpunkt-Fragen auf den Aspekt der Privatsphäre ein, um festzustellen, ob und wenn ja welche Bedenken die Interviewten bei einem Smarthome im Allgemeinen sowie der Steuerung via Sprache oder Face-Tracking haben, da hierfür das Mikrofon beziehungsweise die Kamera des aktiv sein müsste.

- Rückblick:
  Während des Rückblicks soll der Interviewpartner das vorgestellte Produkt oder die Idee im Kontext zu den Problemen die es beziehungsweise sie lösen soll bewerten. Dieser Teil des Interviews beabsichtigt eine allgemeinere Bewertung durch den Interviewten, die sich im Gegensatz zu der vorherigen Phase nicht mehr so sehr auf einzelne Aspekte konzentriert. (vgl. [KGM12, S. 118])

  Der Fragebogen nutzt diesen Abschnitt der Interviews, um den Interviewpartnern die Möglichkeit zu geben, Themen anzusprechen, die ihrer Meinung nach wichtig, aber noch nicht berücksichtigt sind. Zudem ermöglicht er es ihnen, Verbesserungsvorschläge sowie mögliche Probleme zu nennen. Dem Autor erschien darüber hinaus diese Phase des Interviews als am besten geeignet, um den interviewten Personen Fragen bezüglich ihrer körperlichen Einschränkungen zu stellen. Dies war nötig um festzustellen, welche Eingabemöglichkeiten für das barrierefreie Smarthome geeignet sein könnten. Beispielsweise ist es für die Entwicklung des Face-Trackings wichtig, dass der Benutzer seinen Kopf bewegen und blinzeln kann. Als am besten geeignet erschien dieser Zeitpunkt deshalb, da diese Fragen, die für den Interviewten womöglich unangenehm oder zumindest sehr persönlich sind und nur erforderlich sein würden, wenn der Autor sie sich im bisherigen Verlauf des Interviews nicht schon selbst durch Beobachtungen beantworten kann.

- Abschluss:
  Diese Phase schließt das Interview ab. Sie sieht keine gesonderten Fragen vor. (vgl. [KGM12, S. 118])

  Der Fragebogen weicht hiervon nicht ab. Der Abschluss wird dazu genutzt, sich bei dem Interviewpartner zu bedanken und um ihm einen Ausblick auf das weitere Vorgehen für die Erstellung dieser Arbeit zu geben. Letzteres soll auch dazu dienen, die Interviewpartner zur Teilnahme an den Benutzbarkeitstests zu motivieren.

Unabhängig von den einzelnen Phasen wurde während der Vorbereitung der Interviews beim Formulieren der Fragen darauf geachtet, dass diese sich jeweils nur auf ein Thema beziehen, also im Endeffekt nicht nach 2 Sachverhalten fragen. Dies vereinfacht die Auswertung der Interviews, da sich hierdurch die Antwort eindeutig auf eine Frage oder ein Thema bezieht und auch garantiert ist, dass die Antwort vollständig ist. (vgl. [KGM12, S. 121])

Genauso vermeidet der Fragebogen Fragen, die sich mit „Ja“ oder „Nein“ beantworten lassen. Das hat den Vorteil, dass der Interviewte seine tatsächliche Meinung, welche womöglich zwischen „Ja“ und „Nein“ liegt, nicht an eine von den 2 Antwortmöglichkeiten anpassen und somit einen Kompromiss schließen muss (vgl. [KGM12, S. 122]).

Des Weiteren wird soweit möglich vermieden, dass der Interviewteilnehmer durch spätere Fragen das Gefühl vermittelt bekommt, dass er eine vorherige Frage falsch beantwortet hat (vgl. [KGM12, S. 121]). Dies ist nur deshalb bedingt möglich, da die Fragen – wie von Kuniavsky empfohlen – wo immer möglich, offen formuliert sind, es also keine vorgegebene Auswahl an Antworten gibt (vgl. [KGM12, S. 121]). Der Grund hierfür ist, dass die Interviewteilnehmer beispielsweise zunächst gefragt werden, mittels welcher Eingabemethode sie das barrierefreie Smarthome am liebsten bedienen würden, was eine offene Frage ist. Jedoch gibt es im Rahmen dieser Arbeit auch hardwaretechnische, finanzielle sowie aus der Zielsetzung resultierende Einschränkungen, weshalb nicht sämtliche theoretisch möglichen Eingabemethoden realisierbar sind. In Folge dessen umfasst der Fragebogen im Anschluss an die offenen Fragen auch welche zur Sprachsteuerung und dem Face-Tracking, was in gewisserweise dem ein oder anderen Interviewten das Gefühlt vermitteln kann, die Frage nach der besten Eingabemethode falsch beantwortet zu haben.

Die schon angesprochenen offenen Fragen haben den Vorteil, dass der Interviewteilnehmer seine tatsächliche Meinung äußern kann (vgl. [KGM12, S. 121]). Hätte er nur die Wahl zwischen vorgegebenen Antwortmöglichkeiten, würde er aus ihnen jene auswählen, die derjenigen am nächsten kommt, die er eigentlich gerne geben würde (vgl. [KGM12, S. 121]). Exemplarisch lässt sich das anhand der Frage nach dem Gerät, auf welchem die Anwendung für barrierefreie Smarthome ausgeführt werden soll, verdeutlichen. Würde der Interviewpartner bei dieser Frage nur gefragt, ob er ein Tablet oder Smartphone bevorzugt, könnte es passieren, dass das von ihm favorisierte Gerät weder ein Smartphone noch ein Tablet ist, aber diese Information verloren geht, da er sich für 1 von den 2 Antwortmöglichkeiten entscheiden muss.

Neben den offenen Fragen empfiehlt Kuniavsky auch Fragen mit einem offenen Ende (vgl. [KGM12, S. 102]). Er empfiehlt diese für die Vorauswahl der Interviewteilnehmer, da sich hierdurch deren Artikulierfähigkeit sehr gut bestimmbar ist (vgl. [KGM12, S. 102]). Da im Rahmen dieser Arbeit keine Vorauswahl nötig und möglich war, hat es sich angeboten, im Rahmen der Interviews zu erfassen, wie gut sich der jeweilige Teilnehmer artikulieren kann, indem insbesondere in der Retrospective Phase Fragen gestellt wurden, die den Einstieg in eine Diskussion ermöglichten. Für die Auswertung der Interviews sind die dadurch gewonnen Erkenntnisse hilfreich, da sie eine Erklärung dafür sein können, warum es bei den offenen Fragen teilweise schwierig war, ausführliche Antworten zu erhalten. Mit dieser Problematik beschäftigt sich das folgende Unterkapitel welches den Verlauf der Interviews beschreibt, unter anderem.

3.1.3. Verlauf der Interviews

Die 5 Interviews fanden auf 2 Tage verteilt statt. Die Termine hierfür richteten sich nach den Wünschen und zeitlichen Möglichkeiten der Interviewteilnehmer, so wie es auch Kuniavsky vorschlägt (vgl. [KGM12, S. 104]). Jeder Interviewte erhielt eine Aufwandsentschädigung in Höhe von 15€, was die Wertschätzung für die Mitarbeit zum Ausdruck bringen sollte (vgl. [KGM12, S. 108]). Des Weiteren dient eine Aufwandsentschädigung auch zur Motivation der Interviewpartner (vgl. [KGM12, S. 108]). Im vorliegenden Fall war die Motivation der Teilnehmer schon alleine durch ihre unmittelbare Betroffenheit sehr hoch.

Während der Interviews achtete der Autor darauf, die Teilnehmer direkt anzusprechen, sprich sie zum Beispiel zu fragen, wie für sie persönlich ein ideales Smarthome aussieht und nicht allgemeiner, wie ein ideales Smarthome aussieht. Dies präzisiert die Fragen und beugt Missverständnissen sowie in Folge dessen unzuverlässigen Antworten vor (vgl. [KGM12, S. 120]). Zusätzlich wurden die Interviews, nach vorheriger Einverständniserklärung des Interviewten, audiovisuell aufgezeichnet. Dies entlastete den Autor, da er weniger protokollieren musste und ermöglichte es, das jeweilige Interview im Nachhinein besser auszuwerten und zu verhindern, dass er Antworten aufgrund falscher Erinnerungen missinterpretiert.

Alle 5 Interviewteilnehmer waren sehr engagiert und haben sämtliche Fragen verstanden, sprich es gab keine Frage, die aufgrund geistiger Probleme unbeantwortbar war. Lediglich bei einem Probanden waren die Fragen ausführlicher und mehrmals zu erklären. Problematisch hingegen sind die offenen Fragen und solche mit offenem Ende gewesen, bei welchen es um Visionen ging beziehungsweise die eine Diskussion starten sollten. Beides konnten 3 der 5 Interviewteilnehmer nicht entwickeln. Die offenen Fragen, welche eigentlich den Vorteil haben, dass die interviewte Person in ihrer Antwort nicht eingeschränkt und beeinflusst wird und somit den besten Beitrag zu den Ergebnissen liefern sollten, bewirkten also tendenziell eher das Gegenteil. Als Ursache hierfür vermutet der Autor unter anderem folgende Gründe:

- Ein Proband äußerte, dass er nichts Neues mehr brauche und es sich auch nicht kaufen würde, weil er bis zur Fertigstellung womöglich gar nicht mehr lebe oder kurz nach der Anschaffung sterben würde und diese sich dann nicht mehr gelohnt hätte.
- Bei mehreren Interviewpartnern entstand der Eindruck, dass sie keine Visionen mehr entwickeln können, da sie aufgrund geistiger Unterforderung im Alltag in diesem Bereich weit abgebaut haben. Ursächlich könnte hierfür sein, dass die Bewohner über den Tag verteilt nicht viel selbst erledigen und planen müssen, beziehungsweise es aufgrund der Gesamtsituation auch nicht können. Offensichtlich wurde das massiv bei den Fragen, wie das ideale Smarthome für sie aussehen würde und in welchen Bereichen es sie unterstützen könnte. Insbesondere bei letzterer Frage hätte der Autor im Vorfeld damit gerechnet, dass ihn die Teilnehmer des Interviews in Folge ihrer Situation mit Ideen und Wünschen überhäufen.
- Des Weiteren gibt es vermutlich auch ein Abfinden mit der aktuellen Situation. Eine interviewte Person brachte zum Beispiel als Gegenargument zu Anwendungsbeispielen

eines barrierefreien Smarthome, welche der Autor eingebracht hat, dass diese derzeit das Pflegepersonal abdeckt und es dadurch keinen Bedarf gibt.

3.1.4. Auswertung der Interviews

Die folgende Auswertung der Interviews ist unterteilt nach den Aspekten Demographie, Erfahrung mit elektronischen Geräten, Ideen für ein Smarthome, Bedienung, Hardware sowie Privatsphäre.

Demographie

Folgende demographische Daten lassen sich aus den Interviews gewinnen:

- Bei den 5 Teilnehmern handelte es sich um 2 Frauen und 3 Männer.
- Die jüngste Person war 32 und die älteste 70 Jahre alt. Das durchschnittliche Alter der Interviewteilnehmer betrug 47,6 ±13,48 Jahre.
- Der Bildungshintergrund, den die Teilnehmer mitbrachten, war breit gestreut. Folgende jeweils höchste Bildungsabschlüsse gab es: 1 Hauptschulabschluss, 1 Abitur, 2 Berufsausbildungen und 1 Diplom

Erfahrung mit elektronischen Geräten

Eine interviewte Person gab an, keine elektronischen Geräte zu nutzen. Jedoch bedient sie eigenständig einen Fernseher, wie sich herausstellte. 4 von 5 gaben an, elektronische Geräte zu nutzen. Mit Abstand am häufigsten genannt wurden der Fernseher sowie der Computer beziehungsweise Laptop. Des Weiteren nutzen diese 4 Personen Radios, Stereoanlagen oder CD-Player zum Konsum audiovisueller Medien. Die Benutzung von Smartphone und Handys ist hingegen nicht so sehr verbreitet. Eine interviewte Person nutzt ein Handy und würde auch gerne ein Smartphone benutzen, aber glaubt, dass sie dieses aufgrund ihrer körperlichen Einschränkungen nicht bedienen kann. Eine weitere interviewte Person nutzt sowohl Handys als auch Smartphones. Insgesamt lässt sich sagen, dass primär elektronische Geräte zu Unterhaltungs- und Kommunikationszwecken genutzt werden.

Während der Interviews wurden die Teilnehmer auch zu den Erfahrungen befragt, die sie mit den elektronischen Geräten im Hinblick auf ihre Bedienbarkeit haben. Die Ergebnisse hierzu fallen sehr unterschiedlich aus, was auf verschiedene Einschränkungen zurückzuführen ist. Eine Person kann das Radio und den Fernseher nicht selbst bedienen, sondern benötigt hierfür das Pflegepersonal. Eine weitere Person äußerte, dass die Geräte von der Bedienung eher nicht so gut auf ihre Bedürfnisse ausgelegt seien. Die 2 anderen Interviewpartner von den 4, die regelmäßig elektronische Geräte nutzen, können dies den Umständen entsprechend gut. Voraussetzung dafür sind entsprechende Hilfsmittel. Eine Person bedient ihren Fernseher und die Stereoanlage mittels eines Holzsticks über ihren Mund und den Computer mittels

Head-Tracking. Die zweite Person nutzt für die Bedienung des Computers einen Mundstick als Mausersatz und für den Fernseher und die Stereoanlage ein Blasrohr. Der Mundstick ist ihr zu langsam und zu häufig kaputt. Letzteres ist problematisch, da die Reparaturzeit 2 bis 3 Monate beträgt und sie solange den Computer nicht nutzen kann.

Die Interviewpartner wurden nicht nur zu ihren Erfahrungen mit elektronischen Geräten sondern auch konkret zu Smarthomes befragt. Zwei der interviewten Personen gaben an, den Begriff Smarthome zu kennen, 1 sagte, sie kenne ihn ein wenig, 1 weitere Person wusste nicht, was ein Smarthome ist und bei der 5. Person war im Nachhinein nicht mehr sicher feststellbar, ob sie „Smarthome“ mit „Smartphone“ verwechselt hatte. Genutzt hat ein Smarthome bisher nur 1 der interviewten Personen. Hierbei handelt es sich jene, die mittels Blasrohr ihren Fernseher sowie die Stereoanlange steuern kann. Hinzu kommt, dass sich über dieses Gerät auch Steckdosen schalten lassen, weshalb der Autor dies als Nutzung eines Smarthomes wertet.

Ideen für ein Smarthome

Wie schon im vorherigen Unterkapitel 3.1.3 beschrieben, war es sehr schwierig, Visionen von den Interviewpartnern zu erhalten. Folglich gibt es weniger Vorschläge für Anwendungsgebiete eines barrierefreien Smarthomes als erwartet. Einer der Interviewpartner will kein Smarthome und die anderen 4 nannten folgende Ideen:

- Steuerung der Heizkörper, Beleuchtung und Rollläden im Zimmer.
- Via Smarthome steuerbare Musikwiedergabe.
- Eine sprachgesteuerte Suchfunktion für Filme im Internet.
- Die Einstellpositionen des Betts sollen via Smarthome steuerbar sein.
- Eine Bedienung für den Fernseher, die mehr Funktionen unterstützt, als die bisherige Eingabemöglichkeit von einer der interviewten Personen.

Bedienung

Hinsichtlich der Eingabemöglichkeiten wurden den Teilnehmern der Interviews Fragen zu der Benutzeroberfläche und den Eingabemethoden gestellt.

Bezüglich Ersterer resultierten aus den Interviews nur wenige Vorschläge. 2 Interviewpartner nannten große Schaltflächen sowie eine große Schrift als wichtige Eigenschaft. Gegenüber einer baumartigen Menüstruktur, wie sie der Autor vorschlug, gab es keine Einwände. Des Weiteren hat er die Teilnehmer in den Interviews nach ihrer Meinung zu einer raumadaptiven Benutzeroberfläche gefragt. Diese würde sich an den Raum, indem sich das Gerät zur Steuerung des barrierefreien Smarthomes befindet, anpassen. Zum Beispiel könnte es die Steuerung für die Beleuchtung im aktuellen Raum bevorzugt darstellen, sodass weniger Bedienschritte für diese erforderlich sind. Darin erkannten die 4 Interviewpartner, die einen Nutzen in einem barrierefreien Smarthome sehen keinen Vorteil. 2 von ihnen sogar einen Nachteil, da sie eine

konstante Benutzeroberfläche bevorzugen. Eine Erklärung hierfür ist vermutlich, dass die Interviewten jeweils nur ein einzelnes Zimmer für sich haben und dadurch der Bedarf für eine raumadaptive Benutzeroberfläche nicht existiert.

Für die Bedienung des barrierefreien Smarthomes sind im Rahmen der Interviews folgende Eingabemöglichkeiten vorgeschlagen worden:

- Eine Kombination aus Sprachsteuerung und großen Schaltflächen.
- Eine Kombination aus Sprachsteuerung und Blasrohr sowie für die Steuerung vom Rollstuhl aus durch die Verwendung von dessen Joystick.
- Steuerung mittels Touch über ein fest installiertes Tablet.
- Steuerung via Touch mit einer Hand.
- Steuerung durch Kopfbewegungen.

Eine Sprachsteuerung finden 4 Interviewteilnehmer möglich oder gar praktisch. Einer sagte, dass es ihm via Sprachsteuerung zu langsam wäre. Von den 4 würden 2 sie mit einer weiteren Eingabemöglichkeit kombinieren. Als weitere Eingabemethode nannten sie das Face-Tracking sowie ein Touch-Display.

Direkt für das Face-Tracking haben sich nur 2 Teilnehmer ausgesprochen. Allerdings konnten es 2 weitere Interviewpartner ihrer Meinung nach nicht beurteilen, da sie es noch nie ausprobiert haben, aber stehen dieser Eingabemethode offen gegenüber. Lediglich eine interviewte Person sprach sich explizit dagegen aus, da sie diese Art der Eingabe schon als Mausersatz am Computer ausprobiert hat und sie dabei nicht gut fand, weil sie häufig die Maustastenfunktionen versehentlich betätigte.

Zu der Bedienung gehört zudem die Fragestellung, wie sich die Steuerung eines barrierefreien Smarthomes ein- und ausschalten lässt, da es womöglich nicht gewollt ist, dass eine Sprachsteuerung permanent aktiv ist. Hierzu schlugen die Interviewteilnehmer folgendes vor:

- Das Ein- und Ausschalten soll durch das Pflegepersonal erfolgen.
- Automatische Abschaltung nach 10 Minuten und das Einschalten über eine Eingabe mit den Fingern.
- Schaltfläche / Taste für das Ein- und Ausschalten, welche mittels einem Stick / Stift über den Mund betätigbar ist.

Hardware

Bei der bevorzugten Hardware ergibt sich ein geteiltes Bild. Von den 4 Interviewteilnehmern, die das barrierefreie Smarthome nutzen würden, findet jeweils die Hälfte ein kleineres Gerät in der Größe eines Handys / Smartphones beziehungsweise ein sehr großes Tablet gut. Bei den kleineren Geräten äußerte eine Person sogar den Wunsch nach einer Smartwatch. Des Weiteren sollte das Gerät tendenziell an einem bestimmten Ort im Raum fest installiert sein. Allerdings äußerte eine Person auch den Wunsch, dass das Gerät bei ihr und somit ortsungebunden sein sollte. Eine weitere Person, die zwar eine stationär Platzierung möchte, könnte sich zusätzlich auch noch eine Befestigung am Rollstuhl vorstellen.

Privatsphäre

Bezüglich der Privatsphäre sind die Ergebnisse der Interviews sehr homogen. Von den 5 Interviewpartnern hat sich nur 1 gegen ein mit dem Internet verbundenes barrierefreies Smarthome ausgesprochen. Genauso verhält es sich im Hinblick auf die Kamera und das Mikrofon. Sie wären für eine Steuerung via Face-Tracking beziehungsweise Sprache erforderlich. Eine interviewte Person gab an, dass sie sich dadurch beobachtet fühlen würde. Sie fügte jedoch ihre Aussage bei, dass falls ein Mikrofon zur Steuerung erforderlich sei, müsse das akzeptiert werden.

3.2. Definition der Anforderungen

Über die Anforderungen lassen sich die Ziele definieren, welche das barrierefreie Smarthome erfüllen soll. Die Ziele resultieren hauptsächlich aus den im Unterkapitel 3.1.4 ausgewerteten Interviews.

Die Anforderungsdefinition erfolgt mit User-Stories beziehungsweise Epics. Ob es sich um Erstere oder Letztere handelt, ist abhängig von der Detailtiefe. Bei Epics handelt es sich um große User-Stories, die zu einem späteren Zeitpunkt der Realisierung in mehrere kleinere und somit konkretere User-Stories aufgeteilt werden. Folglich finden Epics vor allem in der Anfangsphase einer Umsetzung Anwendung, wenn es darum geht, das System zu beschreiben, so wie es auch in diesem Unterkapitel der Fall ist. (vgl. [Wir11, S. 60])

Die Beschreibung der User-Stories folgt dem Muster von Mike Cohn (vgl. [Wir11, S. 59]): „*Als <Benutzerrolle> will ich <das Ziel>[, so dass <Grund für das Ziel>].*“ [Wir11, S. 59]

Bei der Benutzerrolle unterscheidet die Ausarbeitung zwischen Benutzern und Entwicklern. Dies soll der Übersichtlichkeit dienen. Zwar wäre es möglich, auch noch zwischen Benutzer und Administrator unterscheiden, jedoch wird es sehr stark von den jeweiligen Einschränkungen des Benutzers abhängen, ob er selbst der Administrator ist oder nicht, weshalb auf eine Diversifizierung verzichtet wird. Das Ziel beschreibt die eigentliche Anforderung. Der Grund für das Ziel ist optional und soll die Hintergründe des Ziels erläutern. (vgl. [Wir11, S. 59])

Die Folgende Auflistung definiert zunächst die funktionalen und anschließend die nichtfunktionalen Anforderungen:

- F1 Unterschiedliche Eingabemethoden
  Als Benutzer des barrierefreien Smarthomes möchte ich zwischen unterschiedlichen Eingabemethoden auswählen können, so dass ich die für mich am geeignetste benutzen kann.
  Begründung: Die Auswertung der Interviews im Unterkapitel 3.1.4 zeigt, dass nicht alle Benutzer mit Mobilitätsproblemen an ihren Händen, dieselben Einschränkungen beziehungsweise Probleme haben. Des Weiteren geht aus ihr nicht hervor, dass alle interviewten Personen eine Eingabemethode, wie zum Beispiel die Sprachsteuerung, favorisieren. Darüber hinaus ist zu berücksichtigen, dass die zukünftigen Benutzer des barrierefreien Smarthomes womöglich schon andere elektronische Geräte und in diesem Zusammenhang auch Eingabegeräte zur Herstellung der Barrierefreiheit benutzen, wodurch sie im Umgang mit diesen geübt sind. Wenn sie die schon gewohnte Eingabemethode auch zur Steuerung des barrierefreien Smarthomes benutzen können, stellt dies eine Erleichterung dar.

- F2 Unterstützung der Bedienungshilfen von Android
  Als Benutzer des barrierefreien Smarthomes möchte ich, dass dieses die Bedienungshilfen von Android unterstützt, so dass ich es vergleichbar mit anderen Apps benutzen kann und es für so viele Personen wie möglich nutzbar ist.
  Begründung: Das Betriebssystem Android besitzt schon selbst Funktionen zur Sicherstellung der Barrierefreiheit, welche in Abschnitt 2.3 beschrieben sind. Ebenso wie schon bei der Begründung zur Anforderung F1 gilt auch hier, dass eine Unterstützung von Eingabemöglichkeiten, mit welchen der Benutzer schon vertraut ist, die Benutzung erleichtert. Eine Unterstützung der Android Bedienungshilfen ist deshalb sinnvoll.

- F3 Steuerung via Face-Tracking
  Als Benutzer des barrierefreien Smarthomes möchte ich dieses mittels meines Gesichts bedienen können, so dass ich meine vorhandenen motorischen Fähigkeiten nutzen kann und auf möglichst wenig Hilfe durch andere Personen angewiesen bin.
  Begründung: Das Unterkapitel 2.2 erläutert, dass viele vom Hals ab querschnittsgelähmte Menschen lediglich noch ihren Kopf bewegen können. Dadurch ist die Gesichtsverfolgung zur Steuerung des barrierefreien Smarthome eine mögliche Eingabemethode. Zudem werden zumindest ähnliche Eingabemethoden schon genutzt, wie die Umfrage von Caltenco et al. zeigt (vgl. [CBJA12]). Hierdurch sind manche Personen aus der Zielgruppe des barrierefreien Smarthomes mit dieser Eingabemethode schon vertraut. Des Weiteren waren viele Interviewteilnehmer dem Face-Tracking gegenüber nicht abgeneigt, wie die Auswertung in 3.1.4 zeigt.

- F4 Sprachsteuerung
  Als Benutzer des barrierefreien Smarthomes möchte ich dieses mittels einer Sprachsteuerung bedienen, so dass ich meine vorhandenen sprachlichen Fähigkeiten nutzen kann sowie auf möglichst wenig Hilfe durch andere Personen angewiesen bin.

Begründung: Das Unterkapitel 2.2 beschreibt, dass es bei einer Querschnittlähmung vom Hals an abwärts passieren kann, dass die betroffene Person in der Beweglichkeit ihres Kopfes eingeschränkt ist. Für diese Personengruppe ist eine Steuerung des barrierefreien Smarthomes mittels Gesichtsverfolgung nicht möglich. Des Weiteren sprachen sich die interviewten Personen nicht ausschließlich für eine Bedienung über Face-Tracking, sondern teilweise auch für eine Sprachsteuerung aus, wie die Auswertung in Abschnitt 3.1.4 zeigt.

- F5 Steuerung mittels Taster oder Tastatur
  Als Benutzer des barrierefreien Smarthomes möchte ich dieses mit einem Taster oder einer Tastatur steuern können, so dass ich mit gar keiner oder nur wenig spezieller Hardware das barrierefreie Smarthome entsprechend meiner motorischen Fähigkeiten so bequem und optimal wie es geht nutzen kann.
  Begründung: Die Auswertung der Interviews verdeutlicht, dass es nicht für alle Interviewpartner unmöglich ist, mittels einer Hand oder beiden Händen eine Tastatur, Spezialtastatur oder einen Taster zu bedienen.

- F6 Konfigurierbare Anzahl an Buttons
  Als Benutzer des barrierefreien Smarthomes möchte ich sowohl die Button-Anzahl in horizontale als auch vertikaler Richtung, die auf der Benutzeroberfläche gleichzeitig dargestellt sind, konfigurieren können, so dass die Bedienung für mich nicht zu kompliziert ist.
  Begründung: Für eine konfigurierbare Anzahl an Buttons beziehungsweise Menüelemente sprechen mehrere Gründe. Zunächst einmal macht die Auswertung der Interviews deutlich, dass es keine Displaygröße gibt, welche alle interviewten Personen favorisieren. Eine Festlegung hinsichtlich der Displaygröße wäre auch nicht sinnvoll. Vielmehr bietet es sich an, eine größere Bandbreite von diesen zu unterstützen, indem der Benutzer die Anzahl der dargestellten Schaltflächen und Menüelemente an die Größe des Displays anpasst. Des Weiteren können die Personen aus der Zielgruppe unterschiedliche kognitive und motorische Fähigkeiten haben, wodurch zu viele Auswahlmöglichkeiten auf einmal sie überfordern könnten. Zuletzt ist die geeignete Anzahl an Buttons auch von der verwendeten Eingabemethode abhängig. Zum Beispiel ist es denkbar, dass das Face-Tracking nicht so präzise ist wie eine Tastatur, weshalb größere und im Gegenzug weniger Buttons eine Benutzung von Ersterer erleichtern. Bei einer Tastatur hingegen, würde die Benutzeroberfläche mit genau so vielen Buttons das Potenzial der Eingabemethode womöglich nicht ausnutzen.

- F7 Konfigurierbare Schriftgröße
  Als Benutzer des barrierefreien Smarthomes möchte ich die Schriftgröße für die Button-Beschriftung konfigurieren können, so dass ich die Benutzeroberfläche an meine eigene Fähigkeiten sowie die räumliche Situation anpassen kann.
  Begründung: In den Interviews wurde der Wunsch geäußert, dass die Schriftgröße einstellbar sein sollte. Dessen Umsetzung ist sinnvoll, da die späteren Benutzer des barrierefreien Smarthomes unterschiedlich gut sehen und lesen könnten und zudem die

räumliche Positionierung des Gerätes frei wählbar sein soll. Dadurch kann nicht von einer konstanten Entfernung zwischen den Benutzern sowie dem Gerät ausgegangen werden. Durch eine konfigurierbare Größe der Button-Beschriftung ist es möglich, dass beispielsweise ein Benutzer, welcher das barrierefreie Smarthome aus einer größeren Entfernung bedienen möchte, eine höhere Schriftgröße auswählt.

- F8 Ein- und Ausschalten
  Als Benutzer möchte ich das barrierefreie Smarthome selbstständig ein- und ausschalten können, so dass dieses nicht permanent eingeschaltet ist und ich dafür auch nicht auf die Unterstützung einer anderen Person angewiesen bin. Begründung: Die Steuerung des barrierefreien Smarthomes sollte nicht immer aktiv sein, da dies je nach Eingabemethode einen mehr oder wenigen hohen Ressourcenverbrauch an Rechenleistung darstellen würde und manche Benutzer es vielleicht auch nicht wollen. Zum Beispiel könnten sie sich von einer permanent aktiven Sprachsteuerung oder Kamera für das Face-Tracking beobachtet fühlen. Zusätzlich könnte es insbesondere bei den 2 zuvor genannten Eingabemöglichkeiten zu ungewollten Fehleingaben kommen, wenn sich der Benutzer beispielsweise mit einer anderen Person unterhält und die Sprachsteuerung dabei Gesprochenes irrtümlicherweise als Eingabe interpretiert.

- N1 Eignung für unterschiedlich Geräte
  Als Benutzer möchte ich das barrierefreie Smarthome sowohl mit einem Tablet als auch mit einem Smartphone steuern können, so dass ich eventuell vorhandene und die meiner jeweiligen Situation am besten geeignete Hardware nutzen kann.
  Begründung: Die im Unterkapitel 3.1.4 durchgeführte Auswertung der Interviews kommt zu dem Ergebnis, dass die interviewten Personen das barrierefreie Smarthome mittels unterschiedlich großen Geräten steuern möchte. Dem zu Folge bietet es sich an, die Anwendung so zu gestalten, dass sie sowohl auf Smartphones als auch auf Tablets ein gutes Benutzererlebnis bietet. Zusätzlich erhöht dies die Wahrscheinlichkeit, dass der Benutzer die Anwendung zur Steuerung des barrierefreien Smarthomes auf schon vorhandener Hardware ausführen kann. Ein weiterer Vorteil ist, dass deren Größe an den Einsatzort anpassbar ist. Falls ein Benutzer das barrierefreie Smarthome zum Beispiel von seinem Rollstuhl aus steuern können will, bietet sich hierfür aufgrund der kleineren Abmessungen ein Smartphone an. Hingegen wäre womöglich ein Tablet für einen Benutzer, der das barrierefreie Smarthome nur von einer bestimmten Position innerhalb des Gebäudes bedienen können will, besser geeignet, da es entweder mehr oder größere Schaltflächen darstellen kann.

- N2 Verzicht auf Hilfsmittel-Hardware
  Als Benutzer möchte ich ein barrierefreies Smarthome, das keine Hilfsmittel-Hardware erfordert, die es schon heute für Menschen mit einer hohen Querschnittlähmung ermöglichen, mit elektronischen Geräten zu interagieren, so dass ich gängige Hardware verwenden kann, welche billiger und leichter ersetzbar ist.
  Begründung: Hilfsmittel-Hardware, die es schon heute vom Hals ab querschnittsgelähmten Menschen ermöglicht, mit elektronischen Geräten zu interagieren, hat den Nachteil,

dass sie gegenüber vergleichbaren Eingabegeräten für unversehrte Menschen, erheblich teurer ist. Die hohen Preise ließen sich durch die Verwendung von Standard-Hardware umgehen. Gängige Hardware ist zudem vermutlich schneller reparier- oder ersetzbar und durch die geringeren Anschaffungskosten wäre es günstiger, Ersatz vor zu halten. Idealerweise reduziert sich der Bedarf an Hardware dadurch, dass für die vom barrierefreien Smarthome angebotenen Eingabemethoden keine zusätzliche erfordern.

- N3 Erweiterbares Menü
  Als Benutzer und Entwickler möchte ich das Menü des barrierefreien Smarthomes um weitere Menüelemente ergänzen können, so dass für das Hinzufügen von neuen Funktionalitäten keine Arbeiten am Quellcode der Benutzeroberfläche erforderlich sind.
  Begründung: Die Benutzer werden unterschiedliche Wünsche und Anforderungen bezüglich der Geräte haben, die sie über das barrierefreie Smarthome bedienen möchten. Ebenso werden mit der Zeit neue hinzukommen. Eine Benutzeroberfläche, welche so entwickelt ist, dass sie später von einem Benutzer ohne Programmierkenntnisse erweiterbar ist, bietet vielfältige, individuelle und vor allem kostengünstige Erweiterungsmöglichkeiten.

- N4 Unterstützung von Erweiterungen
  Als Entwickler möchte ich das barrierefreie Smarthome in Form eines Baukastensystems um Funktionalitäten erweitern können, ohne dass hierzu Arbeiten am schon vorhandenen Quellcode erforderlich sind.
  Begründung: Vergleichbar mit der Anforderung N3, werden die Benutzer des barrierefreien Smarthomes unterschiedliche Geräte mit diesem bedienen wollen. Um noch nicht unterstützte leicht in das barrierefreie Smarthome integrieren zu können, ist es praktisch, die gerätespezifischen Funktionalitäten in Form von Erweiterungen umzusetzen. Dadurch sind sie von Dritten programmierbar, ohne dass diese Kenntnis und Zugriff vom beziehungsweise auf den Quellcode des barrierefreien Smarthomes haben.

- N5 Erweiterbarkeit der Eingabemöglichkeiten
  Als Benutzer und Entwickler möchte ich, dass sich das barrierefreie Smarthome ohne großen Aufwand um neue Eingabemethoden erweitern lässt, so dass ich von diesen profitiere.
  Begründung: Es ist nicht ausgeschlossen, dass in den nächsten Jahren bessere Eingabemethoden für die Zielgruppe entwickelt werden. Damit von diesen auch das barrierefreie Smarthome möglichst schnell und ohne großen Aufwand profitiert, sollte dessen Architektur die Ergänzung der Eingabemethoden berücksichtigen.

4. Konzept

Dieses Kapitel beschreibt das Konzept für das barrierefreie Smarthome. Hierzu gibt es am Anfang im Unterkapitel 4.1 zunächst einen Überblick über das System und dessen Komponenten. Im Anschluss daran erläutert das Unterkapitel 4.2 den Aufbau der Benutzeroberfläche und die darin enthaltenen Navigationsmöglichkeiten. Das vorletzte Unterkapitel 4.3 konzentriert sich auf die angebotenen Eingabemethoden, bevor das Unterkapitel 4.4 die Vorstellung des Konzepts abschließt, indem es die Verarbeitung der Eingabeergebnisse beschreibt.

4.1. Systemübersicht

Die Abbildung 4.1 zeigt das barrierefreie Smarthome auf einer sehr abstrahierten Ebene. Die grünen Rechtecke stellen die Zielgeräte dar, welche in der Regel nicht auf der selben Hardware, wie die Anwendung zur Steuerung des barrierefreien Smarthomes, ausgeführt werden. Typische Zielgeräte wären beispielsweise Funksteckdosen, ein Fernseher sowie Funkthermostate.

Das blaue Rechteck in Abbildung 4.1 stellt das Gerät dar, welches die Anwendung für das barrierefreie Smarthome ausführt. Diese lässt sich in 2 Bereiche unterteilen. Das wären zum

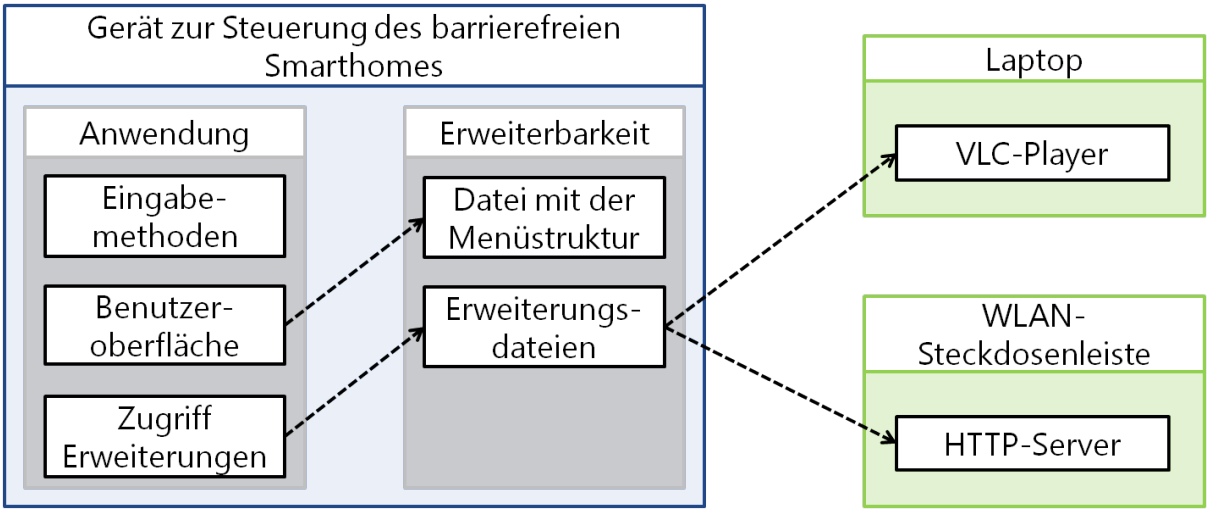

Abbildung 4.1.: Aufbau des barrierefreien Smarthomes.

einen die Anwendung im eigentlichen Sinne sowie die Erweiterungen. Erstere besteht aus folgenden Komponenten:

- Eingabemethoden: Die Komponente bietet dem Benutzer unterschiedliche Eingabemethoden an, wodurch er zwischen diesen wählen kann, wie in Anforderungen F1 gefordert. Eine detailliertere konzeptionelle Beschreibung der Eingabemethoden erfolgt in 4.3.
- Benutzeroberfläche: Die Anwendung generiert die Benutzeroberfläche aus einer Datei, welche zu der Erweiterbarkeit gehört. Dadurch ist es möglich, dass, wie in Anforderung N3 gefordert, die Benutzeroberfläche nachträglich um Menüelemente erweitert oder reduziert werden kann. Demzufolge definiert diese Komponente nicht die Benutzeroberfläche, sondern bietet die erforderlichen Funktionen, um sie dynamisch zur Laufzeit aus einer Datei generieren zu können. Des Weiteren bietet die Anwendung innerhalb der Komponente Benutzeroberfläche die entsprechende Funktionalität, welche zur Navigation innerhalb des Menüs erforderlich ist. Eine ausführliche Beschreibung der Benutzeroberfläche bietet das Unterkapitel 4.2.
- Handhabung der Erweiterungen: Diese Komponente erhält die Eingabebefehle von der Benutzeroberfläche und leitet sie an die passenden Erweiterungen weiter. Sie stellt somit eine Verbindung zwischen diesen und der Komponente Benutzeroberfläche dar. Welche Erweiterung ausgewählt werden soll und welchen Eingabebefehl sie erhält, resultiert aus der Definitionsdatei für die Benutzeroberfläche, in welcher für jeden Button festgelegt ist, ob und wenn ja, welche Aktion auszuführen ist.

Der zweite Bereich besteht aus den Erweiterungen sowie der Datei für die Definition der Menüstruktur. Letztere nutzt die Komponente Benutzeroberfläche um gleichnamige zu erzeugen. Die Erweiterungen bestehen aus mehreren Dateien. Das Konzept sieht je Zielgerät eine Erweiterung vor. Sprich, bezogen auf die zuvor genannten möglichen Zielgeräte, könnte es jeweils eine Erweiterung für die Funksteckdose, eine für den Fernseher und eine für die Thermostate geben. Dadurch, dass die Ansteuerung der Zielgeräte über die Erweiterungen und nicht die Anwendung für das barrierefreie Smarthome erfolgt, lässt sich die Anforderung N4, welche die Möglichkeit zum nachträglichen Hinzufügen von Funktionalitäten durch Dritte fordert, realisieren. Selbiges gilt auch für die Definitionsdatei der Menüstruktur bezogen auf die Anforderung N3. Die Kommunikation zwischen Erweiterung und entsprechendem Zielgerät ist durch das Konzept nicht vorgegeben, woraus eine höhere Flexibilität gegeben ist. Die Kommunikationsmöglichkeiten sind lediglich durch die Fähigkeiten der verwendeten Hardware sowie des Betriebssystems begrenzt.

4.2. Benutzeroberfläche

Dieses Unterkapitel beschreibt die Definition der Menüstruktur, welche die Benutzeroberfläche anzeigen soll. Des Weiteren erläutert es das Navigationskonzept innerhalb der Menüstruktur sowie die Individualisierbarkeit der Benutzeroberfläche.

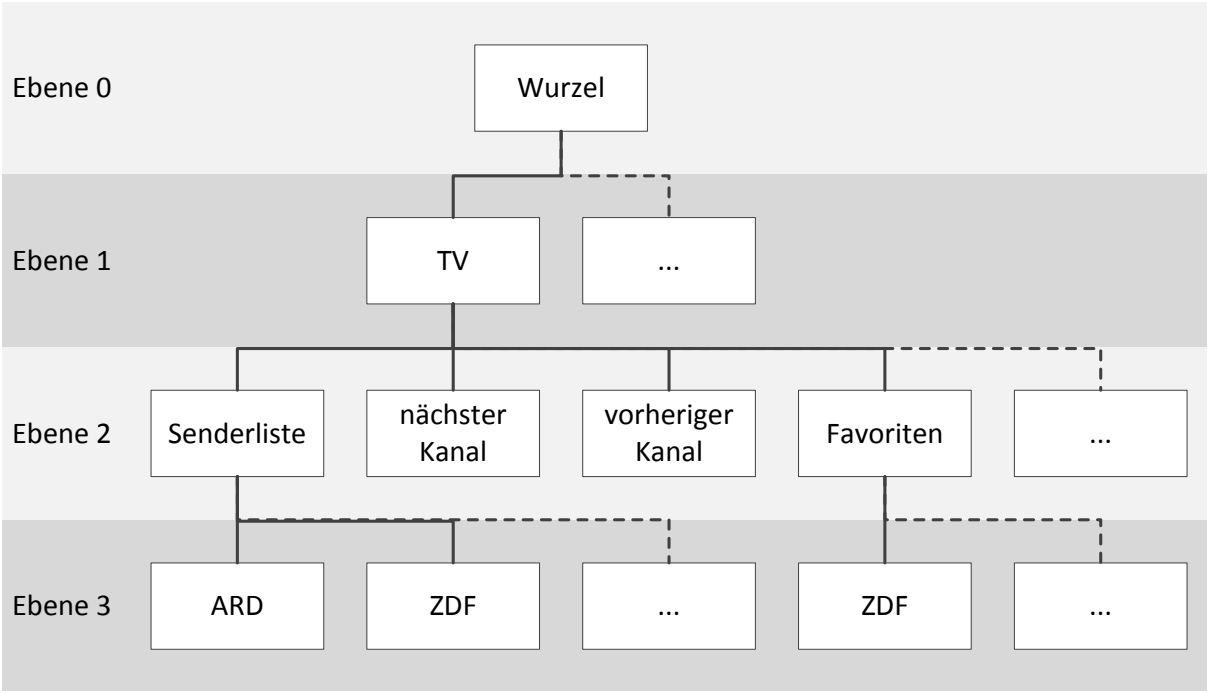

Abbildung 4.2.: Baumhierarchie als Menüstruktur

4.2.1. Menüstruktur

Die Struktur des Menüs entspricht einer Baumhierarchie. Die Abbildung 4.2 soll dies exemplarisch an einem Ausschnitt eines potenziellen Menüs für einen Fernseher veranschaulichen. Die Wurzel auf der Ebene 0 ist für den Benutzer nicht sichtbar. Die höchste für ihn sichtbare Ebene ist die darauffolgende Ebene 1. Sie ist das Hauptmenü. Jedes Rechteck in Abbildung 4.2 beziehungsweise jeder Knoten der Baumstruktur entspricht einem Menüelement. Die Benutzeroberfläche stellt diese als Buttons dar. Abbildung 4.4 zeigt den Entwurf der Benutzeroberfläche exemplarisch anhand der Menüelemente aus der Senderliste von Ebene 3. Aus Platzgründen sind auf dieser Ebene in Abbildung 4.2 nicht alle Menüelemente dargestellt. Sowohl die Definition der einzelnen Menüelemente als auch ihre Einordnung in die Baumstruktur erfolgt in der Definitionsdatei für die Menüstruktur. Ein Menüelement besteht in dieser aus folgenden Elementen:

- ID: Die ID dient zur eindeutigen Identifikation des jeweiligen Menüelements. Durch sie lassen sich in der Definitionsdatei einem Menüelement seine Untermenüelemente beziehungsweise Kindknoten zuordnen.
- Name: Der Name ist die Bezeichnung des jeweiligen Menüelements. Ihn gibt es in drei unterschiedlichen Versionen. Diese wären zum einen die Lang- und Kurzform. Sie dient dazu, die Anforderungen F6 und F7, die eine konfigurierbare Anzahl an gleichzeitig sichtbaren Buttons und Schriftgröße verlangen. Die Kurzversion des Namens lässt sich in Situationen verwenden, in welchen für den langen Namen nicht mehr ausreichend

Platz ist. Des Weiteren kommt diese auch der Forderung N1 nach der Unterstützung unterschiedlicher Displaygrößen zu Gute. Darüber hinaus gibt es als dritte Version des Namens noch eine aussprechbare Variante. Sie dient zur Umsetzung der Anforderung F2. Diese fordert die Unterstützung der Android Bedienungshilfen. Im Rahmen von dieser wird der aussprechbare Name für Google TalkBack verwendet, worüber blinde und sehbehinderte sich die Funktion des jeweiligen Menüelements mittels Sprachausgabe ermitteln können (vgl. [Goo16e]). Zusätzlich zu den 3 Versionen des Namens lassen sich noch Sprachbefehle für das Menüelement definieren. Diese dienen der Umsetzung der Anforderung F4, welche eine Sprachsteuerung fordert.

- Icon: Das Icon soll eine intuitive Bedienung gewährleisten, indem der Benutzer durch dieses nicht die Bezeichnung des Menüelements lesen muss. Denkbar wäre zum Beispiel, dass die Menüelemente in Abbildung 4.4 für die Fernsehsender das jeweilige Senderlogo als Icon verwenden.
- Erweiterung: Hier wird spezifiziert, ob und wenn ja, welche Erweiterung eine Aktion ausführen soll, wenn der Benutzer den Button betätigt. Sofern eine Aktion ausgeführt werden soll, wird der Dateipfad zu der entsprechenden Erweiterung angegeben sowie innerhalb von dieser, der Klassenpfad zu der auszuführenden Klasse. Zusätzlich ist es möglich Parameter in Form von Schlüssel-Wert-Paaren festzulegen, auf welche die Erweiterung Zugriff hat.

4.2.2. Navigation

Der Benutzer kann innerhalb der baumartigen Menüstruktur sowohl vertikal als auch horizontal navigieren. Mittels der vertikalen Navigation kann er zwischen den Ebenen beziehungsweise zwischen den Eltern- und Kindknoten wechseln. In dem in Abbildung 4.3 dargestellten Menü kann er beispielsweise über den Button „TV“ eine Ebene tiefer in das Untermenü des Fernsehers gelangen. Zurück zu der nächst höheren Ebene, gelangt er über den Button „eine Menüebene nach oben“. Die Ebene 0 ist für die Benutzer nicht erreichbar. Der Button „eine Menüebene nach oben“ gehört zu insgesamt 3 Navigations-Buttons. Sie dienen zur Navigation innerhalb des Menüs und sind nicht Teil der Definitionsdatei für die Menüstruktur, sondern der Anwendung. Die 2 noch nicht beschriebenen Navigations-Buttons dienen zur Navigation innerhalb der Kindknoten eines Elternknotens, also einem Untermenü. In Abbildung 4.2 stellen beispielsweise die TV-Sender in der Senderliste ein Untermenü dar. Die Navigation innerhalb eines Untermenüs ist erforderlich, da die Anzahl der gleichzeitig angezeigten Buttons konfigurierbar sein soll und die Größe eines Untermenüs unbegrenzt ist, wodurch es passieren kann, dass dessen Menüelemente nicht alle gleichzeitig darstellbar sind. In der skizzenhaft dargestellten Benutzeroberfläche in Abbildung 4.4 ist dies zum Beispiel der Fall. Sie stellt das Untermenü Senderliste dar, welches mehr Elemente beziehungsweise Programme beinhaltet, als auf einmal anzeigbar sind. Aus diesem Grund befindet sich in der unteren rechten Ecke der Benutzeroberfläche der Navigations-Button „nächste Menüelemente“. Über sie gelangt der Benutzer zu weiteren Elementen des Untermenüs. Abbildung 4.5 zeigt die Benutzeroberfläche wie sie

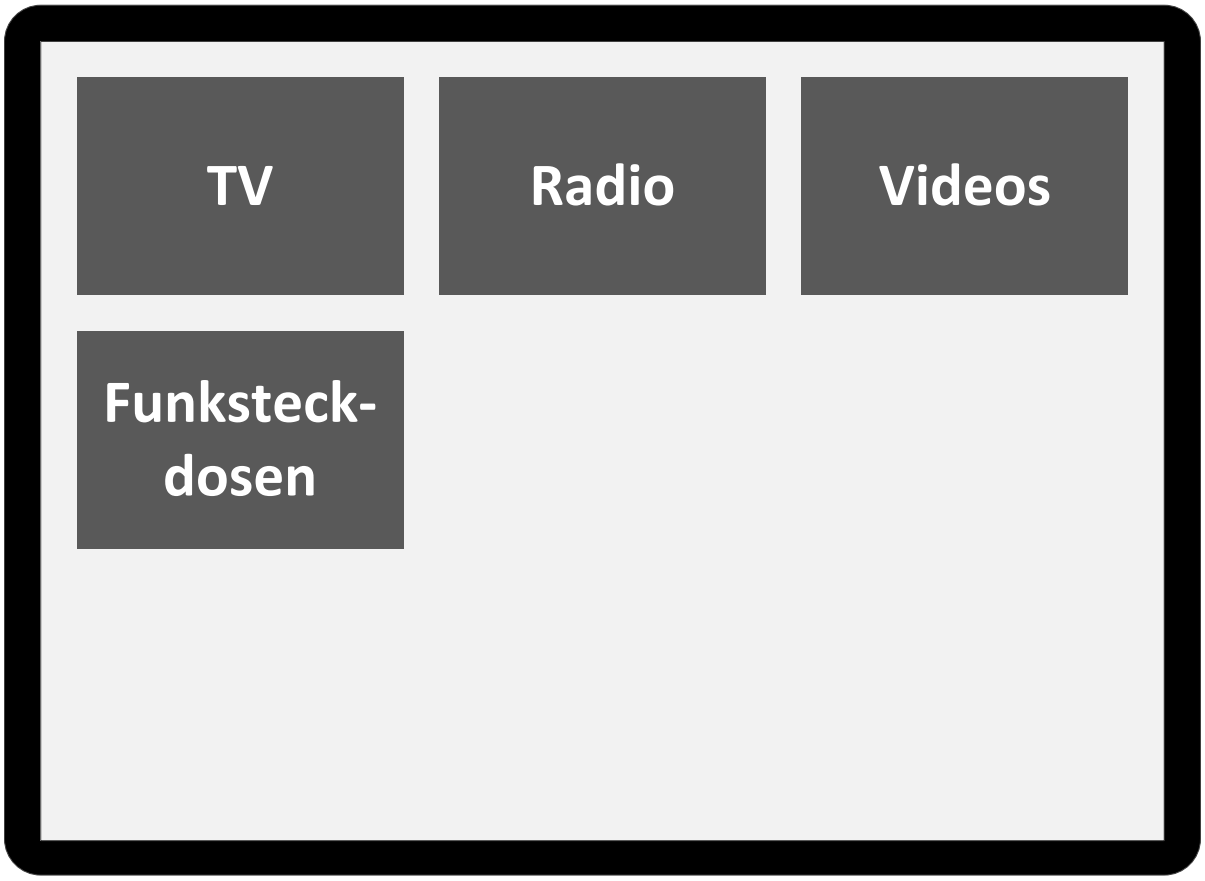

Abbildung 4.3.: Hauptmenü im Entwurf für die Benutzeroberfläche.

aussehen würde, nachdem der Benutzer in Abbildung 4.4 den Navigations-Button „nächste Menüelemente“ getätigt hat. Um zurück zu den vorherigen Elementen des Untermenüs zu gelangen, gibt es den Navigations-Button „vorherige Menüelemente“, wie er auch in Abbildung 4.5 zu sehen ist. Für die vertikale Navigation in dem Menü kann der Benutzer zudem einstellen, ob er, wenn er in ein Untermenü gelangt, in dem er zuvor schon einmal war, innerhalb von diesem an die vorherige Position kommen möchte oder an den Anfang.

Die Verwendung von Buttons zur Navigation hat 3 Vorteile. Erstens lässt sich über sie dem Benutzer signalisieren, in welche Richtungen er ausgehend von der aktuellen Position im Menü navigieren kann. Angenommen er befindet sich in Abbildung 4.2 gerade auf der Ebene 1, könnte er keine Menüebene mehr weiter nach oben in der Baumstruktur gehen. In dieser Situation ist es möglich, dem Benutzer den entsprechenden Navigations-Button nicht oder nur deaktiviert, sprich so, dass er sie nicht betätigen kann, darzustellen. Des Weiteren ist mittels Buttons möglich, mit jeder der in Abschnitt 4.3 beschriebenen Eingabemethoden durch das Menü zu navigieren. Um dies beispielhaft zu beschreiben, ist ein Vergleich zwischen den Eingabemethoden Sprachsteuerung und Face-Tracking hilfreich. Für Erste ließen sich entsprechende Sprachbefehle für die Navigation innerhalb des Menüs definieren. Diese wären bei der

Abbildung 4.4.: Beginn des Untermenüs Senderliste im Entwurf für die Benutzeroberfläche.

Gesichtsverfolgung nutzlos. Stattdessen bräuchte es Buttons zum Navigieren oder es müssten bestimmte Wertebereiche für die Positionen des Gesichtes zur Navigation festgelegt werden. Zum Beispiel, dass der Benutzer die nächsten Elemente eines Untermenüs auswählen kann, in dem er den Kopf über einen bestimmten Winkel hinaus nach rechts dreht. Diese Lösung würde dazu führen, dass die Benutzeroberfläche nicht mehr eingabemethodenübergreifend konsistent ist. Mittels des Navigations-Buttons hingegen, ist die Oberfläche sowie die Navigation für den Benutzer bei jeder Eingabemethode dieselbe.

4.2.3. Individualisierbarkeit

Sowohl die Anforderungen bezüglich der Größe der Hardware als jenen an die Eingabemethoden zeigen, dass es nicht die eine Benutzeroberfläche gibt, welche die Bedürfnisse aller Benutzer abdeckt. Aus diesem Grund bietet sie eine hohe Individualisierbarkeit hinsichtlich des jeweiligen Benutzers. Dabei handelt es sich um folgende Bereiche:

Abbildung 4.5.: Weitere Menüelemente im Untermenü Senderliste im Entwurf für die Benutzeroberfläche.

- Die Anzahl der Buttons ist sowohl in horizontaler als auch in vertikaler Richtung konfigurierbar, wie in Anforderung F6 verlangt. Bezogen auf die Abbildung 4.4 bedeutet dies, dass sich die Benutzeroberfläche in ihrer Konfiguration beispielsweise so ändern lässt, dass sie anstelle von 3 Zeilen mit jeweils 3 Buttons, 3 Zeilen mit jeweils 2 Buttons darstellt. Hierdurch lässt sich die Benutzeroberfläche sehr gut an die jeweilige Displaygröße anpassen. Die Notwendigkeit dazu resultiert aus der Anforderung N1, welche die Benutzbarkeit sowohl auf Tablets als auch Smartphones verlangt.
- Auf die selbe Anforderung ist auch die Konfigurierbarkeit der Schriftgröße für die Button-Texte zurückzuführen. Darüber hinaus verlangt diese jedoch auch explizit die Anforderung F7.
- Die Navigations-Buttons können entweder dauerhaft oder nur im betätigbaren Zustand sichtbar sein, wenn sie also aufgrund der aktuellen Position im Menü auch nutzbar sind. Dies hat den Vorteil, dass der Benutzer die Navigationsschaltflächen in deaktiviertem Zustand ausblenden kann, wodurch sich mehr Menüelemente gleichzeitig darstellen lassen.

- Des Weiteren kann der Benutzer einstellen, dass bei der vertikalen Navigation im Menü je Untermenü die zuletzt angezeigten Menüelemente gespeichert werden. Dies hat den Vorteil, dass der Benutzer bei einer hohen Anzahl an Elementen in einem Untermenü, durch dieses womöglich nicht erneut sehr weit „durchblättern“ muss, um bei dem benötigten Menüelement anzukommen, wenn er dieses zuvor kurz verlassen musste, um die Funktionalität eines anderen Untermenüs zu nutzen.

4.3. Eingabemöglichkeiten

Die Anwendung zur Steuerung des barrierefreien Smarthomes unterstützt mehrere Eingabemöglichkeiten, wie es die Anforderung F1 verlangt. Aus dem Abschnitt 2.2 sowie der Auswertung der Interviews in 3.1.4 geht hervor, dass es sowohl aufgrund der physischen sowie geistigen Fähigkeiten als auch wegen der unterschiedlichen persönlichen Präferenzen, nicht die eine perfekte Eingabemethode zur Bedienung des barrierefreien Smarthomes für die Benutzer aus der Zielgruppe gibt.

Aus diesem Grund lässt sich die Anwendung für das barrierefreie Smarthome neben den herkömmlichen Eingabemethoden mittels Maus, Tastatur und Touch auch via Sprachsteuerung, Face-Tracking sowie Scanning bedienen. Letztere drei Eingabemethoden sind speziell auf die Bedürfnisse der Zielgruppe abgestimmt und sollen ihr die eigenständige Benutzung ermöglichen.

4.3.1. Face-Tracking

Die Steuerung mittels Gesichtsverfolgung wird mit Hilfe der Mobile Vision API von Google realisiert. Sie ermöglicht die Erkennung und Verfolgung von Gesichtern in Bildern beziehungsweise einem Video (vgl. [Goo16b]). Die Verwendung der Mobile Vision API zur Steuerung des barrierefreien Smarthomes mittels Gesichtsverfolgung eignet sich aus mehreren Gründen. Da sie auf Tablets und Smartphones, deren Betriebssystem Android oder iOS ist, verwendbar ist und deren vorhandene Frontkamera nutzen kann, fällt kein Bedarf an zusätzlicher Hardware oder gar solcher, die speziell auf die Bedürfnisse der Zielgruppe abgestimmt ist, an (vgl. [Goo16b]). Hierdurch lässt sich sehr gut die Anforderung N2 realisieren. Darüber hinaus lässt sich die Anforderung N1, welche die Unterstützung von Hardware in unterschiedlicher Größe fordert, mittels der Mobile Vision API, was das Face-Tracking betrifft, ebenfalls sehr gut realisieren, da sie sowohl für Smartphones als auch Tablets verfügbar ist. Des Weiteren entstehen durch ihre Verwendung keine Kosten (vgl. [Goo16c]).

Zu einem erkannten Gesicht liefert die Mobile Vision API mehrere Informationen (vgl. [Goo16b]). Unter anderem handelt es sich dabei um folgende für das Face-Tracking benötigte:

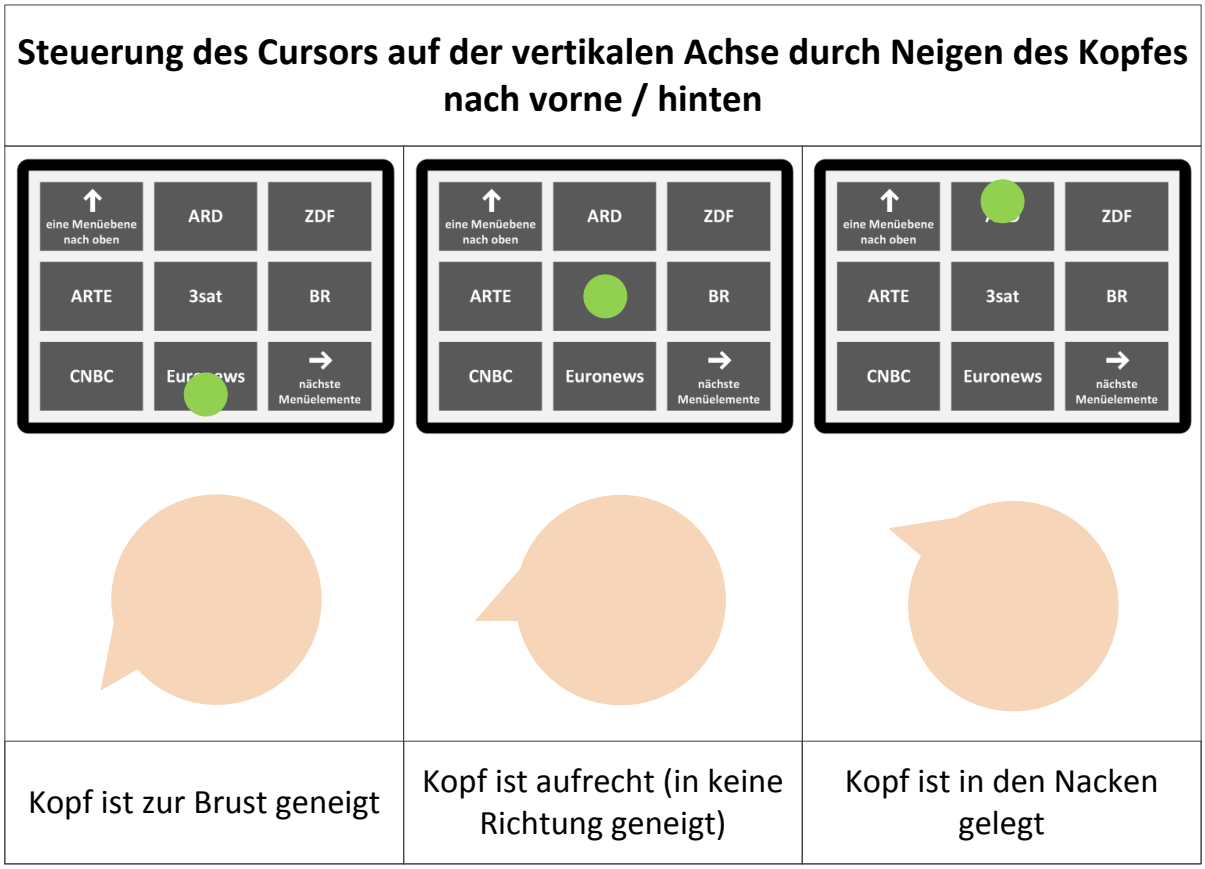

Abbildung 4.6.: Steuerung des Cursors auf der vertikalen Achse durch Neigen des Kopfes.

- Die Position als Koordinaten in Form von Pixelwerten von diversen Merkmalen im Gesicht, wie zum Beispiel den Augen sowie der Nasenwurzel. (vgl. [Goo16b] u. [Goo16f])
- Rotation des Gesichts in Grad. Sprich, ob die Person frontal auf die Kamera schaut oder nach links / rechts an dieser vorbei sieht. (vgl. [Goo16b])
- Öffnungswahrscheinlichkeit für das linke und rechte Auge (vgl. [Goo16d]). Jedoch ist diese bisher (Stand 09.12.2016) nur in der API Version für Android verfügbar (vgl. [Goo16b]).

Die Neigung und Rotation von dem Kopf des Benutzers werden auf die Position eines Kreises abgebildet, welcher über den Buttons in der Benutzeroberfläche angezeigt wird. Er ist somit eine Art Cursor. In Abbildung 4.6 ist er grün dargestellt. Indem der Benutzer seinen Kopf in Richtung Brust oder „leicht“ in den Nacken legt, ändert er die vertikale Position des Cursors. Die Abbildung 4.6 veranschaulicht das. Realisiert wird die vertikale Steuerung, indem das Intervall mit den möglichen Werten der Kopfneigung auf jenes mit den möglichen Positionen des Cursors auf der vertikalen Achse ausgehend von 0 abgebildet wird. Dafür sind folgende Variablen von Bedeutung:

- Eingabewert e_v: Bei dem Eingabewert handelt es sich um die ermittelte Kopfneigung.
- Eingabewert für die minimale vertikale Cursor-Position e_{minv}: Dies ist der kleinste Eingabewert für die Kopfneigung.
- Eingabewert für die maximale vertikale Cursor-Position e_{maxv}: Dies ist der größte Eingabewert für die Kopfneigung.
- Maximale vertikale Cursor-Position p_{maxv}: Das ist die maximale Position des Cursors auf der vertikalen Achse.
- Vertikale Cursor-Position p_v: Hierbei handelt es sich um die aus der Kopfneigung berechnete Position des Cursors auf der vertikalen Achse.

Die mathematische Formel in vereinfachter Form hierfür lautet wie folgt:

$$p_v = \frac{e_v - e_{minv}}{e_{maxv} - e_{minv}} \cdot p_{maxv}$$

Bei der Formel handelt es sich um eine vereinfachte Form, weil sie davon ausgeht, dass die Eingabewerte größer oder gleich dem minimalen Eingabewert und kleiner oder gleich dem maximalen Eingabewert sind. In der Praxis ist dies jedoch nicht der Fall, da aus Komfortgründen nicht die aus körperlicher Sicht maximal möglichen Werte für die Kopfneigung verwendet werden.

Da in der Mobile Vision API die Bestimmung der Kopfneigung noch nicht implementiert ist, muss diese über die vertikale Positionsveränderung eines Merkmals im Gesicht, zum Beispiel der Nasenwurzel, bestimmt werden (vgl. [Goo16b]). Dies ist möglich, da sich beim Neigen des Kopfes auch die Position der Nasenwurzel oder anderer Merkmale in der Vertikalen verändert.

Über das Drehen seines Kopfes nach links oder rechts kann der Benutzer den Cursor auf der horizontalen Achse bewegen. Die in Abbildung 4.7 gezeigten Beispiele veranschaulichen das.

Die Rotation des Kopfes lässt sich direkt über die Mobile Vision API ermitteln, da diese in ihr schon implementiert ist (vgl. [Goo16b]). Die Mobile Vision API gibt die Drehung des Kopfes in Grad zurück. Wenn der Benutzer seinen Kopf nach links dreht, handelt es sich um einen positiven und bei einer Drehung nach rechts, um einen negativen Wert (vgl. [Goo16b]). Auch hier muss das Intervall der Eingabewerte auf jenes mit den möglichen Positionen des Cursors auf der horizontalen Achse ausgehend von 0 abgebildet werden. Dafür sind folgende Variablen von Bedeutung:

- Eingabewert e_h: Drehung des Kopfes in Grad.
- Eingabewert für die minimale horizontale Cursor-Position e_{minh}: Dies ist der kleinste Eingabewert für die Drehung des Kopfes.
- Eingabewert für die maximale horizontale Cursor-Position e_{maxh}: Dies ist der größte Eingabewert für die Drehung des Kopfes.

- Maximale horizontale Cursor-Position p_{maxh}: Das ist die maximale Position des Cursors auf der horizontalen Achse.
- Horizontale Cursor-Position p_h: Hierbei handelt es sich um die aus der Kopfdrehung berechnete Position des Cursors auf der horizontalen Achse.

Die dazugehörige Formel in vereinfachter Form lautet wie folgt:

$$p_h = \frac{e_h - e_{minh}}{e_{maxh} - e_{minh}} \cdot p_{maxh}$$

Die Eingabewerte für die minimale und maximale horizontale Position können nicht größer als ungefähr +18 Grad beziehungsweise kleiner als -18 Grad sein. Ursächlich hierfür ist, dass andernfalls die Öffnungswahrscheinlichkeit für das linke und rechte Auge nicht mehr feststellbar ist. (vgl. [Goo16b])

Die Augenöffnungswahrscheinlichkeit dient zur Erkennung des Blinzelns. Mittels diesem kann der Benutzer den Button betätigen, den er mit dem Cursor ausgewählt hat. Gegen Verweilen als Auswahlmethode spricht, dass Dybdal et al. in ihrer Studie zu dem Ergebnis kamen, dass

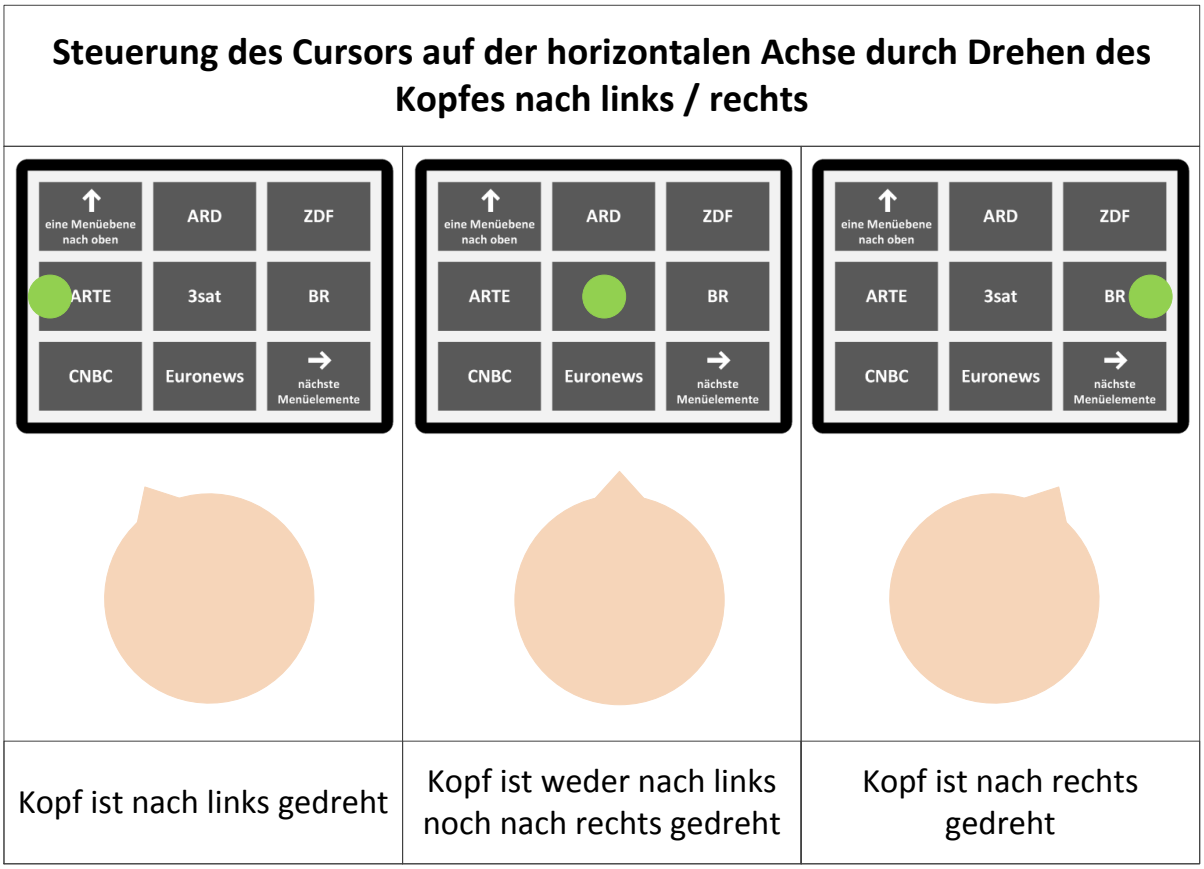

Abbildung 4.7.: Steuerung des Cursors auf der horizontalen Achse durch Drehen des Kopfes.

sich Gesten zur Bestätigung einer Auswahl auf kleinen Displays besser eignen als Verweilen (vgl. [DAH12]). Des Weiteren möchten die potenziellen Benutzer mit der Anwendung zur Steuerung des barrierefreien Smarthomes nicht permanent mit diesem interagieren, sondern zwischendurch andere Tätigkeiten durchführen, weshalb eine Betätigung der Buttons durch Verweilen auch deshalb als weniger geeignet erscheint. Das Blinzeln lässt sich in 3 Arten unterscheiden:

- Blinzeln mit dem linken Auge.
- Blinzeln mit dem rechten Auge.
- Gleichzeitiges Blinzeln mit beiden Augen.

Des Weiteren wäre es noch möglich, die Dauer des Blinzelns auszuwerten, sodass beispielsweise zwischen schnellem und langsamen Blinzeln unterschieden werden kann. Dieser Ansatz wird im Rahmen der Abschlussarbeit jedoch nicht weiter verfolgt, da ohne die Berücksichtigung der Geschwindigkeit beim Blinzeln schon 3 Eingabebefehle möglich sind. Das Konzept für das Face-Tracking sieht vor, dass mittels Blinzeln ein Button betätigbar ist. Zusätzlich soll der Benutzer das Face-Tracking pausieren und fortsetzen können. Jedoch besteht schon bei diesen 2 verschiedenen Eingabebefehlen das Problem, dass nicht alle Menschen mit nur einem Auge blinzeln können. Es ist aber auch möglich, mittels allen 3 Blinzelarten ein und dieselbe Funktion auszuführen, zum Beispiel, dass der Button betätigt wird, unabhängig davon, ob mit dem linken, rechten oder beiden Auge(n) geblinzelt wurde. Das kann für Benutzer, die Schwierigkeiten beim Blinzeln haben, die Betätigung einer Schaltfläche erleichtern.

4.3.2. Scanning

Diese Eingabemethode fokussiert die Buttons nacheinander in einer definierten Reihenfolge. Der Fortschritt der Fokussierung kann wahlweise manuell oder automatisch erfolgen. Bei letzterer ist festlegbar, wie lange ein Button fokussiert bleiben soll. Je kürzer dies ist, desto schneller ist der Fortschritt. Optional ist einem Eingabebefehl die Funktionalität zuweisbar, die automatische Fokussierung zu pausieren oder fortzusetzen. Wenn der Fortschritt manuell erfolgt, muss der Benutzer für jede Fokussierung eine Eingabe vornehmen. Unabhängig davon, ob der Fortschritt der Fokussierung automatisch oder manuell erfolgt, kann der Benutzer über eine Eingabe den zu diesem Zeitpunkt fokussierten Button betätigen. Erfolgt die Fokussierung des jeweils nächsten Buttons automatisch, ist auch die Rede von 1-Button-Scanning, da der Benutzer in diesem Fall nur 1 Taster beziehungsweise einen Eingabebefehl zur Bedienung benötigt. Führt er den Fortschritt hingegen selbst durch, sind 2 erforderlich, weshalb diese Eingabemöglichkeit 2-Button-Scanning genannt wird. Der Benutzer soll das Scanning sowohl über eine Tastatur als auch mittels Blinzeln bedienen können. Für Erstere ist theoretisch die Verwendung einer normalen Tastatur möglich. Jedoch ist das Scanning für Benutzer, welche eine herkömmliche Tastatur noch bedienen können, unpraktisch. Vielmehr ist es für Personen gedacht, die eine Tastatur nur noch sehr eingeschränkt oder gar nicht nutzen können. Für Erstere bietet sich die Benutzung eines wie in Abbildung 4.8 dargestellten Button-Switch an.

Abbildung 4.8.: Button Switch, der sich für das 1- und 2-Button-Scanning eignet.

Bei diesem handelt es sich um eine spezielle Tastatur, welche aus lediglich 2, aber dafür sehr großen, Tasten besteht. Dadurch können ihn Personen nutzen, die eine oder beide Hände noch eingeschränkt bewegen können. Beim 1-Button-Scanning kann der Benutzer über eine der beiden Tasten die fokussierte Schaltfläche betätigen und optional mit der zweiten den Fortschritt pausieren oder fortsetzen. Bei der Eingabemethode 2-Button-Scanning dient die zweite Taste zur Fokussierung des nächsten Buttons. Analog dazu soll auch das Scanning via Blinzeln funktionieren. Die Erkennung von diesem erfolgt über die Mobile Vision API von Google, welche auch das im vorherigen Unterkapitel beschriebene Face-Tracking nutzt. Sofern der Benutzer sowohl mit einem einzelnen als auch mit beiden Augen blinzeln kann, stehen drei verschiedene Eingabebefehle zur Verfügung. Für das 2-Button-Scanning muss der Benutzer 2 Blinzelarten beherrschen, vergleichbar mit den 2 Tasten beim Bluetooth Switch. Beim 1-Button-Scanning ist es hingegen ausreichend, eine der Blinzelarten zu können, da das Pausieren und Fortsetzen optional ist.

4.3.3. Sprachsteuerung

Bei dieser Eingabemethode kann der Benutzer die Schaltflächen via Sprache bedienen. Diese Arbeit verfolgt die Sprachsteuerung konzeptionell nur soweit, dass das in 4.2.1 beschriebene Konzept für die Definition der Menüstruktur die Zuweisung von Sprachbefehlen in den einzelnen Menüelementen vorsieht. Der Grund dafür, dass das Konzept die Sprachsteuerung nur am Rande unterstützt ist der, dass Google derzeit an einer Sprachsteuerung für das Betriebssystem Android arbeitet, um Benutzern, welche ihre Hände nicht zur Bedienung verwenden können, diese zu ermöglichen (vgl. [Goo16m]). Bei der Sprachsteuerung handelt es sich um die App Voice Access, welche sich zum Erstellungszeitpunkt dieser Arbeit noch in der Entwicklung befindet (vgl. [Goo16m]), aber zu Testzwecken schon verwendbar ist. Die Spracheingabe mittels Voice Access funktioniert, indem der Benutzer entweder den Button-Text des Buttons, den er betätigen möchte, oder die Zahl sagt, welche die Sprachsteuerung an jedem klickbaren Element der Benutzeroberfläche einblendet (vgl. [Goo16l]). Das Betätigen einer Schaltfläche über die zwei zuvor genannten Möglichkeiten funktioniert schon gut, weshalb die mittels Voice Access mögliche Sprachsteuerung für die im Kapitel 6 beschriebenen Benutzbarkeitstests ausreichend ist. Eine eigene Umsetzung der Sprachsteuerung zum jetzigen Zeitpunkt wäre in den Augen des Autors nicht sinnvoll gewesen, da es besser ist, zunächst abzuwarten, wie gut die App Voice Access nach ihrer Fertigstellung funktionieren wird. Wenn ihre Funktionsweise zufriedenstellend ist, wäre es sowohl für die Benutzung als auch die Entwicklung beziehungsweise Pflege der Anwendung zur Steuerung des barrierefreien Smarthomes einfacher, die vom Betriebssystem zur Verfügung gestellte Sprachsteuerung zu verwenden, als eine eigene zu entwickeln. Nachteilig an der Sprachsteuerung mittels Voice Access ist , dass sie für die Spracherkennung auf eine Internetverbindung angewiesen ist (vgl. [Goo16i]).

4.3.4. Vergleich der Eingabemethoden

Die Bedienung des Smarthomes via Touch, Maus oder Tastatur ist für die in Abschnitt 3.1.1 beschriebene Zielgruppe nur schwer oder gar nicht möglich. Aus diesem Grund soll sich die Anwendung für das barrierefreie Smarthome durch eine der in den vorherigen 3 Unterkapiteln beschriebenen Eingabemethoden barrierefrei bedienen lassen. Auf den ersten Blick erscheint es vielleicht als unnötig, dass zur Herstellung der Barrierefreiheit für Benutzer, welche ihre Hände nur noch eingeschränkt oder gar nicht mehr verwenden können, mehrere Eingabemethoden entwickelt werden sollen. Jedoch haben diese Benutzer trotz der Gemeinsamkeit, dass sie ihre Hände nur noch eingeschränkt oder gar nicht mehr verwenden können, unterschiedliche Fähigkeiten und Bedürfnisse. Das ist zum einen darauf zurück zu führen, dass sich ihre Behinderung beziehungsweise Krankheit unterschiedlich auswirkt oder sie an mehreren leiden. Zum anderen haben sie unterschiedliche Wünsche und Bedürfnisse hinsichtlich der Bedienung, wie auch die Auswertung der Interviews in 3.1.4 im Bereich der Eingabemethoden ergab. Die Interviewteilnehmer wurden nach ihrer Meinung zu der Sprachsteuerung sowie dem Face-Tracking gefragt. Die Tabelle 4.1 soll den zuerst genannten Aspekt untermauern,

Tabelle 4.1.: Benötigte Fähigkeiten je Eingabemethode

Eingabemethode	erforderliche Fähigkeit(en)
Touch	eingeschränkte Motorik an einer Hand
Maus	uneingeschränkte Motorik an einer Hand
Tastatur	uneingeschränkte Motorik an einer Hand
1-Button-Scanning via Taster	1 Taster bedienbar
1-Button-Scanning via Blinzeln	1 Blinzelart beherrschen
2-Button-Scanning via Taster	2 Taster bedienbar
2-Button-Scanning via Blinzeln	2 Blinzelarten beherrschen
Face-Tracking	Kopf bewegen u. 1 Blinzelart beherrschen
Sprachsteuerung	sehr verständliche Aussprache

indem sie die Eingabemethoden für das barrierefreie Smarthome sowie die jeweils für sie benötigten Fähigkeiten auflistet.

Die Tabelle 4.1 zeigt, dass sich das Face-Tracking für Personen eignet, die ihren Kopf bewegen und blinzeln können. Jedoch kann die Querschnittlähmung auch so hoch sein, dass eine betroffene Person ihren Kopf nicht mehr bewegen kann, wie in Abschnitt 2.2.2 näher beschrieben. Für sie stellt die Sprachsteuerung dann eine Alternative dar. Letztere könnte grundsätzlich auch von den Personen genutzt werden, die ihren Kopf noch bewegen können. Jedoch bringt auch die Sprachsteuerung Einschränkungen mit sich. Zum einen können sie Umgebungsgeräusche stören und zum anderen gibt es Benutzer, die neben der Querschnittlähmung zusätzlich noch Sprachprobleme haben, was die Sprachsteuerung zumindest erschwert oder gar unmöglich macht, weshalb diese vom Face-Tracking profitieren können. Darüber hinaus ist es möglich, dass ein Benutzer weder seinen Kopf bewegen noch für die Sprachsteuerung ausreichend gut sprechen kann. Für diesen Benutzer eignet sich dann das Scanning mittels Blinzeln. Das Scanning ist in Kombination mit einer speziellen Tastatur mit großen Tasten darüber hinaus für Benutzer geeignet, die ihre Hände zwar noch bewegen können, aber denen es an der Präzision zur Bedienung einer herkömmlichen Tastatur oder Maus fehlt. Für die selbe Benutzergruppe wäre es teilweise auch möglich die App zur Steuerung des barrierefreien Smarthomes mittels Touch zu bedienen, da sich die Anzeigegröße der Buttons einstellen und somit an die motorischen Fähigkeiten der Hand des Benutzers anpassen lässt. Für die Bedienung einer Maus oder Tastatur muss die Funktionsfähigkeit der Hand hingegen besser erhalten sein.

4.4. Eingabeverarbeitung

Die Anforderungen F1 sowie N5 verlangen zum einen die Unterstützung mehrerer Eingabemethoden und zum anderen, dass sich weitere zu einem späteren Zeitpunkt problemlos hinzufügen lassen. Deshalb ist es sinnvoll, die Realisierung der Eingabemethoden Scanning

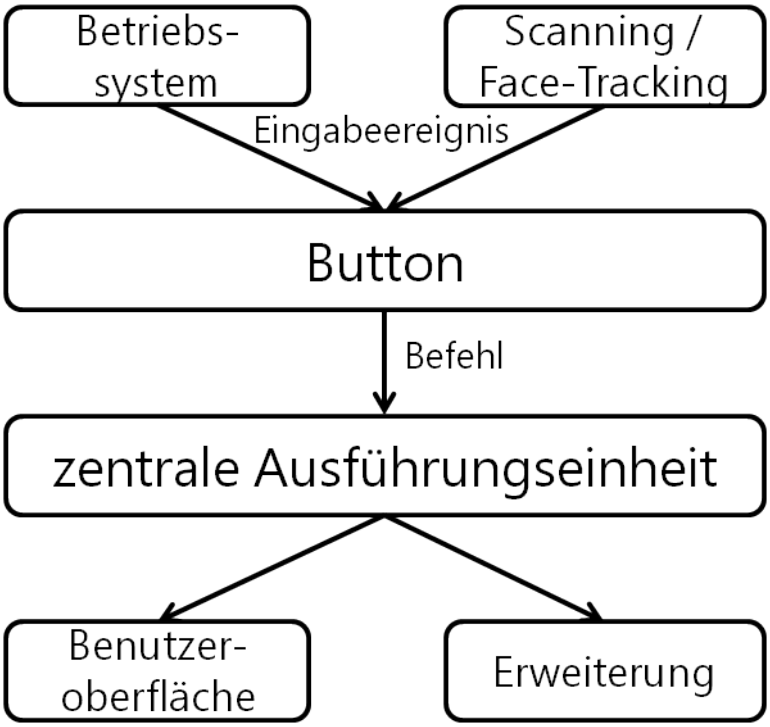

Abbildung 4.9.: Konzeptionelle Eingabeverarbeitung

sowie Face-Tracking, welche nicht vom Betriebssystem aus angeboten werden, von der Benutzeroberfläche zu trennen. Dadurch ist es zu einem späteren Zeitpunkt mit wenig Aufwand möglich, eine weitere Eingabemethode hinzuzufügen. Die Abbildung 4.9 zeigt den Ablauf, welchen ein Eingabeereignis in der Anwendung zur Steuerung des barrierefreien Smarthomes nimmt und dadurch auch, wie die 2 zuvor genannten Eingabemethoden gekapselt sind.

Auf der zweiten Ebene befindet sich die Benutzeroberfläche. Sie besteht aus den Buttons. Wenn der Benutzer die Eingabe via Touch, Tastatur, Maus oder der Sprachsteuerung vornimmt, wird diese vom Betriebssystem verarbeitet und an den entsprechenden Button weiter geleitet. Die Eingaben, die der Benutzer mittels Scanning oder Face-Tracking tätigt, kann das Betriebssystem dagegen nicht weiterleiten, da diese Eingabemethoden nicht Bestandteil von diesem sind. Stattdessen sind sie Teil der Anwendung zur Steuerung des barrierefreien Smarthomes und befinden sich in der Abbildung 4.9 auf der Ebene 1. Aus der Perspektive eines Buttons verhalten sich die beiden Eingabemethoden jedoch vergleichbar wie die anderen, da sie wie das Betriebssystem das Eingabeereignis an ihn weitergeben.

Da das Konzept für die Benutzeroberfläche vorsieht, dass der Benutzer diese selbst um Menüelemente beziehungsweise Buttons erweitern kann, ist es für die Anwendung zur Steuerung des barrierefreien Smarthomes gar nicht möglich, zu wissen, an welches Ziel sich der Eingabebefehl richtet. Bei den nicht bekannten Zielen handelt es sich um die im Unterkapitel 4.1 beschriebenen Erweiterungen. Für solche Situationen, in denen es also unklar ist, was genau

passieren soll, wenn der Benutzer eine Button tätigt, eignet sich das Befehls-Entwurfsmuster (vgl. [GR04, S. 173]). Der in Abbildung 4.9 dargestellte Ablauf der Eingabeverarbeitung entspricht diesem. Wenn ein Button ein Eingabeereignis erhalten hat, leitet er einen Befehl an eine zentrale Ausführungseinheit auf Ebene 3 weiter. Der Befehl beinhaltet die Funktionalität, welche als Folge der Button-Betätigung auszuführen ist. Entsprechend den im Unterkapitel 4.2.1 beschriebenen Menüelementen besteht die auszuführende Funktionalität aus einem Navigationsschritt in der Menüstruktur und / oder dem Aufruf einer Erweiterung. Durch die zentrale Ausführung der Befehle ist die Logik für die Steuerung des barrierefreien Smarthomes von deren Benutzeroberfläche getrennt.

5. Umsetzung Prototyp

Dieses Kapitel behandelt die prototypische Implementierung der Anwendung zur Steuerung des barrierefreien Smarthomes. Hierzu begründet es zunächst, welche Anforderungen in Form des Prototyps realisiert und was für Technologien dafür verwendet werden. Anschließend stellt es dessen Architektur vor und beschreibt die Implementation ihrer einzelnen Komponenten.

5.1. Begründung für die prototypische Umsetzung ausgewählter Anforderungen

Die Realisierung des Prototyps umfasst einen Großteil der in Abschnitt 3.2 definierten Anforderungen. Das Hauptaugenmerk der Realisierung liegt dabei auf den barrierefreien Eingabemethoden Face-Tracking (Anforderung F3), Sprachsteuerung (Anforderung F4) sowie 1- und 2-Button-Scanning (Anforderung F5), um diese im Rahmen der in Kapitel 6 beschriebenen Benutzbarkeitstests mit den herkömmlichen Eingabemethoden Touch, Maus und Tastatur vergleichen zu können.

Lediglich zwei Anforderungen werden gar nicht oder nur teilweise implementiert. Dabei handelt es sich zum einen um die Anforderung F8, welche fordert, dass der Benutzer das barrierefreie Smarthome selbstständig ein- und ausschalten können soll. Im Verlauf der Interviews und Kozeptionierung hat sich herausgestellt, dass diese Funktion besondere Herausforderungen mit sich bringt, welche im Rahmen dieser Arbeit nicht umsetzbar sind. Darüber hinaus liegt der Schwerpunkt, wie schon weiter oben beschrieben, auf dem Vergleich der verschiedenen Eingabemethoden zur Steuerung des barrierefreien Smarthomes, weshalb das Ein- und Ausschalten von diesem zunächst als zweitrangig betrachtet werden kann. Zumal ein Großteil der Personen aus der in Abschnitt 3.1.1 beschriebenen Zielgruppe die meiste Zeit eine Hilfsperson in ihrer Nähe hat, welche das Ein- und Ausschalten zunächst übernehmen kann. Zum anderen wird die in der Anforderung F5 verlangte Sprachsteuerung nicht implementiert, sondern mit Hilfe der App Voice Access von Google, welche sich derzeit noch in der Entwicklung befindet (Stand 08.12.2016), in den Benutzbarkeitstests getestet (vgl. [Goo16m]). Zwar könnte eine eigene Implementierung womöglich besser funktionieren, jedoch würde es sich bei dieser dann um eine Sprachsteuerung handeln, welche nur innerhalb der App zur Steuerung des barrierefreien Smarthomes verwendbar ist. Langfristig könnte das zu Schwierigkeiten mit der Voice Access App von Google führen oder mit dieser konkurrieren, weshalb es aus der Sicht des Autors wenig sinnvoll gewesen wäre, in eine eigene Sprachsteuerung zum jetzigen Zeitpunkt zu

investieren. Zumal die Funktionalität der App Voice Access zum Vergleich der Eingabemethode Sprachsteuerung mit den anderen Eingabemethoden grundsätzlich ausreichend ist und sie später sogar eine Möglichkeit zur Umsetzung des Ein- und Ausschaltens der Steuerung des barrierefreien Smarthomes sein könnte, da sich mit ihr das gesamte Android Betriebssystem bedienen lassen können soll (vgl. [Goo16m]).

In den in Abschnitt 3.1 beschriebenen Interviews äußerten die Teilnehmer mehrere Wünsche hinsichtlich Geräten, welche sie mit der App zur Steuerung des barrierefreien Smarthomes bedienen können möchten. Dazu gehörten unter anderem die Rollläden, Heizung sowie die Einstellung der Liegeposition im Bett. Jedoch müsste für die Ansteuerung von diesen zunächst entsprechende Hardware angeschafft werden, was mit Kosten verbunden gewesen wäre. Darüber hinaus eignen sich diese Geräte nicht für Demonstrationszwecke und Benutzbarkeitstests, da sie entweder fest in einem Gebäude installiert oder nur schwer zu transportieren sind. Die Auswertung der Interviews in Abschnitt 3.1.4 zeigt aber auch, dass die Teilnehmer sehr viel Zeit mit dem Konsum von Fernsehen und Radio verbringen. Darüber hinaus äußerte eine interviewte Person den Wunsch, an ihrem Fernseher mehr Funktionen eigenständig bedienen zu können. Eine weitere gab als mögliches Anwendungsszenario an, mittels einer Sprachsteuerung nach Filmen im Internet suchen zu können. Deshalb bietet es sich an, einen Fernseher als ein Zielgerät für die Anwendung zur Steuerung des barrierefreien Smarthomes zu benutzen, denn zum einen würde er vermutlich von einem Großteil der Personen aus der Zielgruppe häufig genutzt und zum anderen lässt er sich leicht transportieren und es muss keine Hardware gekauft werden, da für die Benutzbarkeitstests auch ein Laptop als „Fernseher“ ausreicht. In Abhängigkeit der genutzten TV-Software wäre es zudem sogar möglich, eine Onlinefilmsuche inklusive Streaming zu realisieren. Zusätzlich zu der Ansteuerung eines Fernsehers soll jedoch noch eine eher klassische Smarthome-Funktion realisiert werden. Hierzu bieten sich Funksteckdosen als zweites Zielgerät an, da sie sehr flexibel sind. Zum Beispiel lässt sich mit ihnen ohne großen Aufwand die Steuerung einer Beleuchtung aufbauen, da sich herkömmliche Lampen über sie ein- und ausschalten lassen. Des Weiteren sind sie transportabel, was für die Benutzbarkeitstests von Vorteil ist.

5.2. Verwendete Systeme und Technologien

Die App zur Steuerung des barrierefreien Smarthomes wird für das Betriebssystem Android prototypisch umgesetzt. Sie setzt den API-Level 21 oder höher beim Android Software Development Kit voraus. Optimiert ist sie auf den API-Level 23. Die Entwicklung und das Testen der App zur Steuerung des barrierefreien Smarthomes erfolgt auf dem Tablet Pixel C, auf welchem Android 7.0 installiert ist. Selbiges findet auch Verwendung für die in Kapitel 6 durchgeführten Benutzerstudien. Die Erweiterungen und in die App eingebundene Bibliotheken basieren auf der Java Version 6.

Als Smart-TV Zielgerät sollte ursprünglich das Programm Kodi TV, installiert auf einem Computer, dienen. Dabei handelt es sich um ein umfangreiches Media Center Programm, welches

nicht nur das Fernsehen, Filmeschauen und Musikhören ermöglicht, sondern zusätzlich durch Addons erweiterbar ist (vgl. [Kod16a]). Hierdurch wäre es möglich, nicht nur lineares Fernsehen und Radio, sondern auch Videoplattformen, wie von einer der interviewten Personen gewünscht, zu unterstützen. Jedoch ist Kodi TV auf die Bedienung via Fernbedienung ausgelegt (vgl. [Kod16c]). Dies ist vermutlich ursächlich dafür, dass die angebotene API nur die Navigationsmöglichkeiten bietet, welche mittels einer Fernbedienung möglich sind, wahrscheinlich um die Entwicklung von Apps als Ersatz für diese zu unterstützen (vgl. [Kod16b]). Bei den Navigationsmöglichkeiten handelt es sich um die Pfeiltasten sowie die Auswahlbestätigung (vgl. [Kod16b]). Es ist jedoch nicht möglich, mittels einer Anfrage, einen bestimmten Fernseh- beziehungsweise Radiosender auszuwählen, unabhängig davon, wie dieser empfangen wird (vgl. [Kod16b]). Dadurch müsste der Benutzer von der App zur Steuerung des barrierefreien Smarthomes in dieser jedes Mal eine Eingabe vornehmen, um in der Benutzeroberfläche von Kodi TV einen „Schritt“ zu navigieren oder eine Auswahl zu bestätigen. Zum Einstellen eines Senders wäre eine Folge von Eingaben in der App erforderlich. Darüber hinaus wäre es unbequem, wenn der Benutzer mit dem Blick ständig zwischen zwei Benutzeroberflächen wechseln müsste. Denn die App zur Steuerung des barrierefreien Smarthomes wäre im Endeffekt eine virtuelle Tastatur mit welcher der Benutzer in der Oberfläche von Kodi TV navigiert. Aus den genannten Nachteilen von Kodi TV in Kombination mit dem Konzept, wurde nach andere Optionen gesucht. Bei dieser Suche stellte sich der VLC-Player als eine Alternative heraus. Zwar besitzt dieser eine Benutzeroberfläche, die für Fernseher weniger geeignet und zudem nicht über Addons erweiterbar ist, was zur Konsequenz hat, dass Videoplattformen nicht ohne weiteres integrierbar sind (vgl. [Vid16a]), aber seine API ermöglicht es, mittels einer Anfrage und damit auch einer Benutzereingabe in der App zur Steuerung des barrierefreien Smarthomes, einen Fernseh- oder Radiosender auszuwählen (vgl. [Vid16b]). Dadurch ist es möglich in der Benutzeroberfläche der App, eine Senderauswahl in Form mehrerer Buttons darzustellen, wo jeder einen Sender repräsentiert und der Benutzer den gewünschten durch Betätigen des entsprechenden Buttons auswählen kann.

Für die Funksteckdosen wird eine WLAN-Steckdosenleiste der Wöhlke EDV-Beratung GmbH verwendet (vgl. [Wöh16a]). Sie besitzt drei Steckdosen, welche sich unter anderem mittels HTTP-Anfragen einzeln an- und ausschalten lassen (vgl. [Wöh16b]). Für die prototypische Realisierung ist sie ideal, da sie leicht zu transportieren und vielfältig einsetzbar ist. Beispielsweise ist es denkbar, mit ihr im Rahmen von Benutzbarkeitstests oder zu Demonstrationszwecken nicht nur Leuchtmittel, sondern auch Klimaanlagen beziehungsweise elektrische Heizungen ein- und auszuschalten.

5.3. Architektur

Dieses Unterkapitel beschreibt die Architektur des Prototypen, ausgehend von einem UML-Komponentendiagramm. Im Anschluss daran gibt es in Form eines UML-Klassendiagramms einen Überblick über die Eingabemethoden und wie diese hinsichtlich der Architektur in den

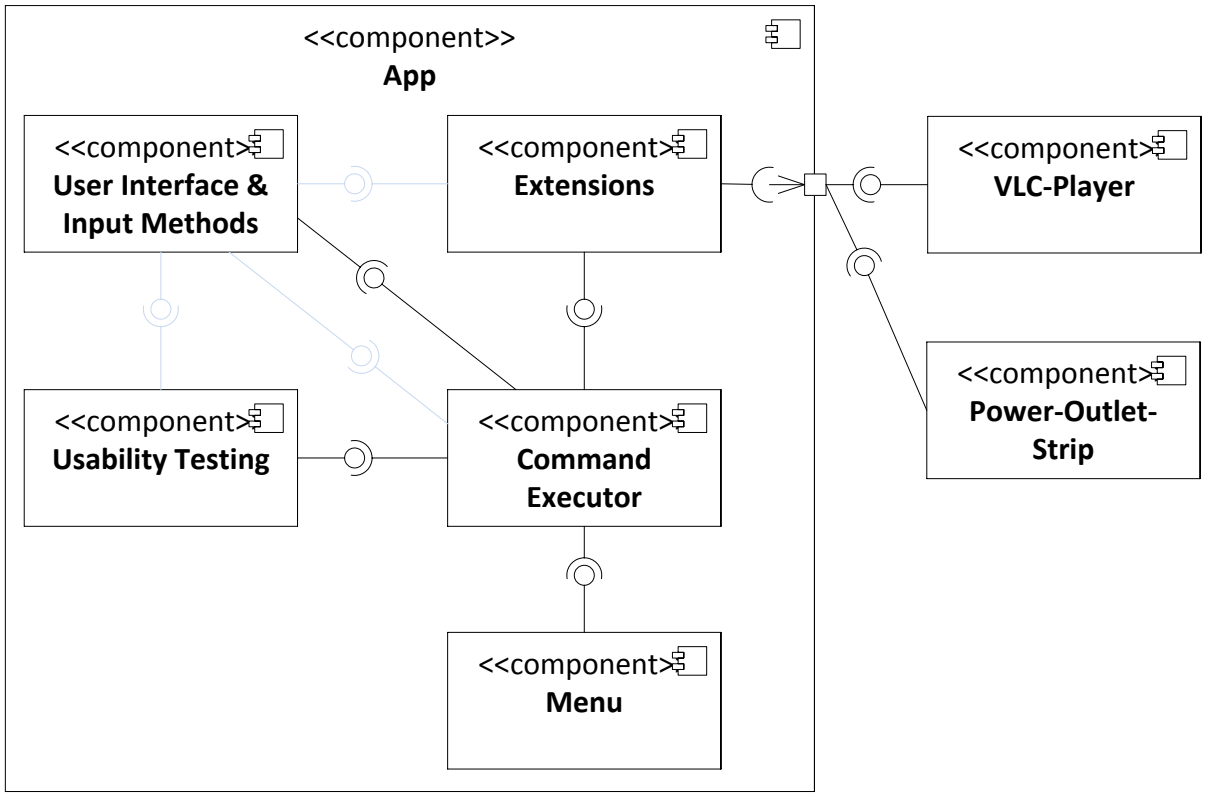

Abbildung 5.1.: UML-Komponentendiagramm für die App zur Steuerung des barrierefreien Smarthomes.

Prototypen integriert sind. Das Unterkapitel schließt im Anschluss daran mit einer abstrakten Beschreibung der Abhandlung einer Benutzereingabe mit Hilfe eines UML-Sequenzdiagramms ab. Die in diesem Abschnitt beschriebene Architektur ist nicht vollständig, beinhaltet also nicht alle Klassen, sondern gibt einen Überblick über die wichtigsten Komponenten des Prototyps.

5.3.1. Komponenten

Das UML-Komponentendiagramm in Abbildung 5.1 zeigt die prototypische Realisierung des barrierefreien Smarthomes unterteilt in Module. Grundsätzlich kann zwischen zwei Bereichen unterschieden werden. Die Komponente App fasst alle Komponenten zusammen, welche Bestandteil der Steuerung des barrierefreien Smarthomes sind und auf einem Gerät mit dem Betriebssystem Android ausgeführt werden. Bei den zwei Komponenten, die sich außerhalb davon befinden, handelt es sich um die Zielgeräte, welche über die App bedienbar sind. Diese wären zum einen die Komponente VLC-Player, welche auf einem Computer ausgeführt wird und den Smart-TV darstellen soll und zum anderen die Komponente Power-Outlet-Strip. Beide Zielgeräte behandelt der Abschnitt 5.4.6 noch ausführlicher.

Die Komponente User Interface & Input Methods umfasst die Benutzeroberfläche sowie die unterstützten Eingabemethoden. Zu ihr gehört die Darstellung der Menüelemente in Form von Buttons und Meldungen an den Benutzer. Nicht Bestandteil der Komponente User Interface & Input Methods ist die Menüstruktur.

Die Menüstruktur ist in einer eigenen Komponente namens Menu realisiert. Ihre Aufgabe ist es zum einen, die Menüstruktur beim Start der App aus einer XML-Datei zu generieren und zum anderen, die Navigation in ihr. Für Letzteres verfolgt sie die aktuelle Position innerhalb der Menüstruktur und bietet eine Schnittstelle um in dieser zu navigieren. Das Ergebnis eines Navigationsbefehls sind die im Anschluss an diesen darzustellenden Menüelemente beziehungsweise Buttons. Hierdurch ist die komplette Menüstruktur sowie die Navigation in der Komponente Menu gekapselt.

Die Schnittstelle der Komponente Menu nutzt die Komponente Command Executor. Diese führt einen Befehl aus, wenn der Benutzer einen Button tätigt. Den entsprechenden Befehl erhält sie hierfür von der Komponente User Interface & Input Methods. Für die Ausführung eines Befehls nutzt die Komponente Command Executor Schnittstellen der Komponenten Menu und Extensions. Letztere nutzt er, um die Zielgeräte anzusteuern. Die Schnittstelle der Komponente Menu nutzt er, um Navigationsbefehle in der Menüstruktur auszuführen und um die als nächstes anzuzeigenden Menüelemente zu erhalten. Durch die Anwendung des Befehl-Entwurfmusters in Form der Komponente Command Executor findet die Ausführung aller Benutzereingaben an einer zentralen Stelle statt, was auch für die Protokollierung während der im Abschnitt 6.2 beschriebenen qualitativen Benutzbarkeitstests von Vorteil ist.

Für diese gibt es eigens die Komponente Usability Testing. Sie bietet eine Schnittstelle zur Protokollierung der Benutzereingaben in einer CSV-Datei. Über sie kann die Komponente Command Executor während den qualitativen Benutzbarkeitstests die Eingaben der Probanden festhalten.

Die Komponente Extensions fasst sämtliche Erweiterungen zusammen, welche zur Ansteuerung der Zielgeräte dienen. Für jede Art von Zielgerät wird es dazu in der Regel jeweils eine JAR-Datei geben, weshalb die Komponente Extensions sozusagen aus mehreren Unterkomponenten besteht. Im Rahmen der Implementierung des Prototyps gibt es jeweils eine Erweiterung zur Ansteuerung des VLC-Players und der WLAN-Steckdosenleiste.

Die Komponente User Interface & Input Methods besitzt zudem noch drei Rückrufschnittstellen (engl. callback interfaces). Sie sind in dem in Abbildung 5.1 dargestellten UML-Komponentendiagramm der Übersichtlichkeit wegen in Hellblau hervorgehoben. Eine der Rückrufschnittstellen nutzt die Komponente Usability Testing um den Benutzer die zu lösende Aufgaben und das Ende des Tests in der Benutzeroberfläche anzeigen zu können. Auf die zweite Rückrufschnittstelle greift die Komponente Command Executor zu, um im Anschluss an die Ausführung eines Navigationsbefehls die neu anzuzeigenden Menüelemente beziehungsweise Buttons an die Benutzeroberfläche übergeben zu können. Die dritte Rückrufschnittstelle nutzt die Komponente Extensions. Über sie können die Erweiterungen dem Benutzer Fehlermeldungen oder andersweitige Informationen anzeigen.

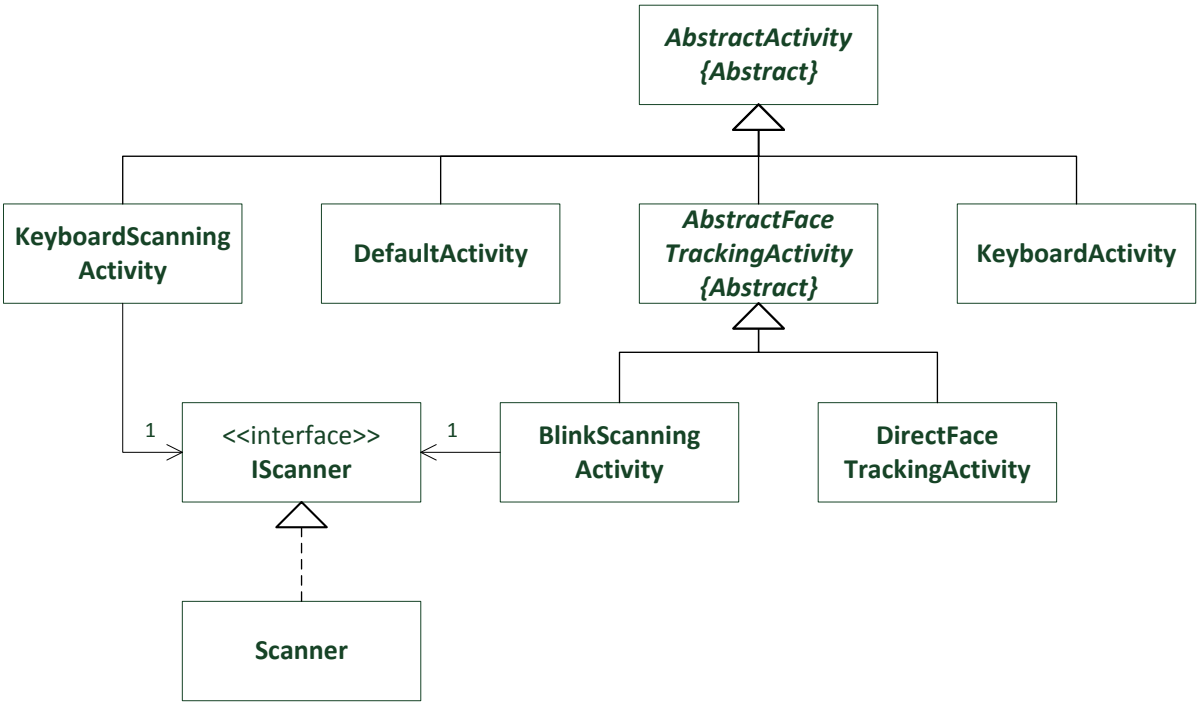

Abbildung 5.2.: UML-Klassendiagramm mit den wichtigen Klassen der Eingabemethoden.

5.3.2. Eingabemethoden

Das UML-Klassendiagramm in Abbildung 5.2 zeigt eine Übersicht über die Klassen der Eingabemethoden und wie diese zueinander in Beziehung stehen. Sämtliche in diesem Diagramm gezeigten Klassen gehören zu der im vorherigen Abschnitt unter anderem beschriebenen Komponente User Interface & Input Methods. Das UML-Klassendiagramm besteht jedoch nur aus den wichtigsten Klassen je Eingabemethoden. Hilfsklassen wurden der Übersichtlichkeit wegen weg gelassen.

Die Klasse `AbstractActivity` erweitert die `Activity` Klasse von Android. Erstere besitzt die Funktionalität, welche alle Eingabemethoden benötigen. Hierbei handelt es sich in erster Linie um Funktionen zur Darstellung der Menüelemente sowie um die drei Rückruf-Funktionen für die Komponenten Usability Testing, Command Executor und Extensions.

Die `DefaultActivity` erweitert die Klasse `AbstractActivity`. Sie besitzt keine spezifischen Funktionen, da sie die Eingabemethoden nutzen, welche das Betriebssystem Android schon unterstützt. Dies wären Touch, Maus sowie die Sprachsteuerung mittels der App Voice Access.

Die Klasse `KeyboardActivity` beinhaltet die Implementation für die Eingabemethode Tastatur. Diese ist nicht in der `DefaultActivity` realisiert, da in den Einstellungen festlegbar ist, welcher der angezeigten Buttons standardmäßig fokussiert ist. Ohne diese Option wäre eine eigene

Klasse nicht nötig, da die Eingabe mittels Tastatur das Betriebssystem Android ebenfalls schon unterstützt.

Die abstrakte Klasse `AbstractFaceTrackingActivity` generalisiert die erforderlichen Funktionalitäten des Face-Trackings, die sowohl für die Eingabemethode Face-Tracking als auch die beiden Eingabemethoden 1- und 2-Button-Scanning mittels Blinzeln erforderlich sind. Erstere ist vollends in der Klasse `DirectFaceTrackingActivity` realisiert. Für das Face-Tracking sind jedoch im erheblichen Umfang noch weitere Klassen erforderlich, welche der Abschnitt 5.4.4 detaillierter beschreibt. Das 1- und 2-Button-Scanning mittels Blinzeln ist in der Klasse `BlinkScanningActivity` implementiert, da es für die Blinzelerkennung Funktionen des Face-Trackings benötigt.

Für die Eingabemethoden 1- und 2-Button-Scanning mittels einer Tastatur oder Tasters ist die Klasse `KeyboardScanningActivity` zuständig. Da das 1-Button-Scanning mittels Blinzelns und jenes mit Tastatur, für die automatische Fokussierung des nächsten Buttons sowie die Möglichkeit zur Pausierung und Fortsetzung von dieser, die selbe Funktionalität benötigen, aber es keine Mehrfachvererbung in Java gibt, ist dieser Teil der Implementierung in die Klasse `Scanner`, welche die Schnittstelle `IScanner` implementiert, ausgelagert.

5.3.3. Ablauf einer Eingabe

Das UML-Sequenzdiagramm in Abbildung 5.3 zeigt den Ablauf einer Benutzereingabe und soll damit das Zusammenspiel der zuvor beschriebenen Komponenten und Klassen verdeutlichen. Wenn der Benutzer mittels einer der zur Verfügung stehenden Eingabemethoden einen Button betätigt, wird dessen Methode `performClick`, wenn es sich um keine betriebssystemseitig unterstützte Eingabemethode handelt, durch eine der von der Klasse `AbstractActivity` erbende Klasse aufgerufen. Im anderen Fall übernimmt das Android Betriebssystem die Verarbeitung des Eingabeereignisses. In beiden Fällen finden im Vorfeld noch weitere Methodenaufrufe statt, die für das Verständnis des Ablaufs jedoch unbedeutend sind und in Folge dessen in dem UML-Sequenzdiagramm in Abbildung 5.3 nicht enthalten sind.

In Folge des Aufrufs der Methode `performClick` beziehungsweise der Eingabeverarbeitung durch das Android Betriebssystem wird die Methode `onClick` des jeweiligen `View.OnClickListeners` aufgerufen. Jeder der betätigbaren Buttons besitzt einen. Dieser greift daraufhin auf eine Instanz der Klasse `CommandExecutor` zu, welche als Singleton implementiert und Kern der Komponente Command Executor ist, um mittels des Aufrufs der Methode `executeCommand` eine Instanz einer, die Klasse `AbstractCommand` erweiternde Klasse, auszuführen. Eine solche Instanz besitzt jeder Button. Sie beinhaltet die jeweilige Funktionalität, welche ausgeführt werden soll, wenn der Benutzer den entsprechenden Button betätigt. Eine genauere Beschreibung davon findet in Abschnitt 5.4.2 statt.

Der `CommandExecutor` ruft die Methode `executeCommand` an einer Instanz der Klasse `AbstractCommand` auf und führt, sofern qualitative Benutzbarkeitstests durchgeführt werden, eine Protokollierung der Eingabe durch. Diese ist in dem UML-Sequenzdiagramm nicht

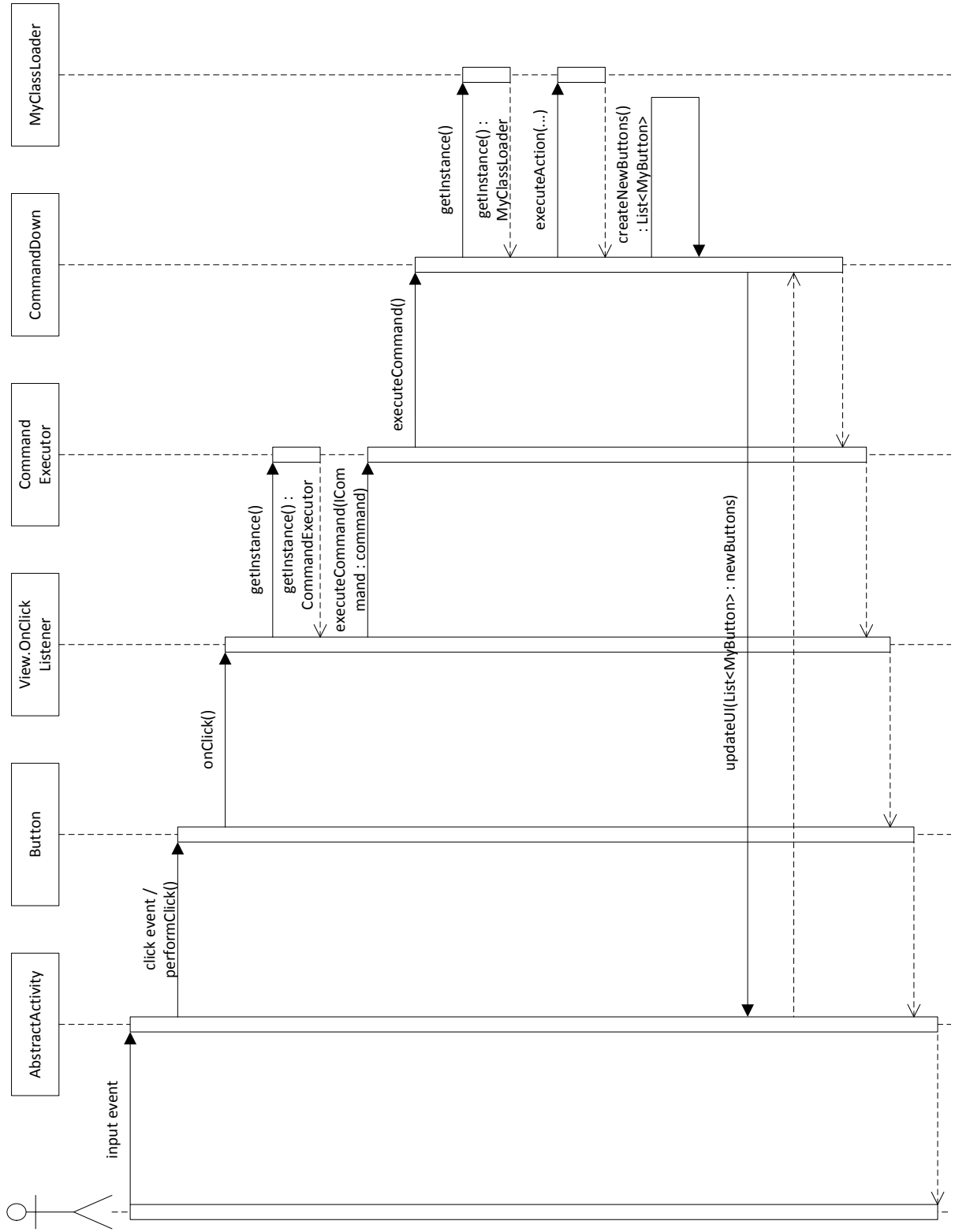

Abbildung 5.3.: UML-Sequenzdiagramm, das die Verarbeitung einer Eingabe in der App zur Steuerung des barrierefreien Smarthomes zeigt.

dargestellt. Wenn das Menüelement, welches der betätigte Button repräsentiert, den Aufruf einer Erweiterung beinhaltet, beispielsweise die Auswahl eines Fernsehsenders, wird in der Methode `executeCommand` der Klasse `AbstractCommand` zunächst die Methode `executeAction` in der Klasse `MyClassLoader` aufgerufen. Diese greift anschließend auf eine der Erweiterungen aus der Komponente Extensions zu, welche daraufhin die Kommunikation mit dem entsprechenden Zielgerät vornimmt. Diese ist nicht mehr in dem UML-Sequenzdiagramm in Abbildung 5.3 enthalten. Des Weiteren findet dieser Schritt nebenläufig statt, wodurch die Ausführung der Methode `executeCommand` in der Klasse `AbstractCommand` nicht verzögert oder schlimmstenfalls blockiert wird. Wenn es sich bei dem betätigten Button um einen Navigationsbutton handelt, mit dem also in der Menüstruktur nach oben, unten, links oder rechts navigiert werden kann, instanziiert die Methode `executeCommand` über den Aufruf der Methode `createButtons` anschließend die neuen Buttons. Hierfür muss Letztere auf die Schnittstelle der Komponente Menu zugreifen. Dieser Zugriff sowie die erforderlichen Schritte zur Instanziierung der Buttons sind in dem UML-Sequenzdiagramm in Abbildung 5.3 nicht mehr dargestellt. Nach der Instanziierung der neuen Buttons sind diese noch an die Benutzeroberfläche zu übergeben, damit sie für den Benutzer sichtbar werden. Hierfür ruft die Methode `executeCommand` die Methode `updateUI` der Klasse `AbstractActivity` auf und übergibt dieser die neuen Buttons. Bei der Methode `updateUI` handelt es sich um eine der drei Rückrufschnittstellen der Komponente User Interface & Input Methods.

5.4. Implementierung

Dieses Unterkapitel befasst sich mit der Implementierung des Prototyps. Dazu startet es mit der Benutzeroberfläche sowie der Menüstruktur. Anschließend beschreibt es, wie die Komponente Command Executor sowie die Eingabemethoden realisiert wurden. Den Abschluss dieses Unterkapitels bildet die Implementierung der Erweiterungen.

5.4.1. Menüstruktur und Benutzeroberfläche

Dieses Unterkapitel behandelt die Implementierung der Menüstruktur sowie der Benutzeroberfläche. Zunächst beschreibt es jedoch, wie sich die Menüstruktur in einer XML-Datei definieren lässt.

Definition der Menüstruktur

Die Definition der Menüstruktur findet in einer XML-Datei statt, welche beim Start der App mittels JDOM geparst wird um die entsprechende Java-Objekte zu erzeugen (vgl. [JDO16]). Das Parsen der XML-Datei ist Bestandteil der Komponente Menu ebenso die entsprechenden Klassen, deren Instanzen die XML-Elemente zur Laufzeit repräsentieren.

Der Quellcodeausschnitt 5.1 zeigt eine Übersicht der XML-Datei, welche die Menüstruktur des Prototyps beinhaltet. Sie besteht aus 4 Elementen, welche jeweils eine Liste mit weiteren Elementen beinhaltet.

```
<?xml version="1.0" encoding="UTF-8"?>

<menu xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

      <supported_languages>
            ....
      </supported_languages>

      <shared_attributes>
            ...
      </shared_attributes>

      <menu_items>
            ...
      </menu_items>

      <menu_hierarchy>
            ...
      </menu_hierarchy>

</menu>
```

Quellcodeausschnitt 5.1: Aufbau der Menüstruktur in der XML-Datei.

Das Element `supported_languages` beinhaltet die Sprachen, welche die definierte Benutzeroberfläche unterstützt. Eine Mehrsprachigkeit ist vorgesehen, aber im Prototyp nicht vollständig umgesetzt, weshalb diesem Element bis jetzt keine weitere Bedeutung zukommt.

Das Element `shared_attributes` kann beliebig viele `shared_attribute` Elemente beinhalten. Ein Shared-Attribute besteht aus einem eindeutigen Schlüssel als Attribut, sowie einem Datentyp und einem Wert als Unterelement. Bei dem Datentyp und dem Wert handelt es sich um Strings. Der Quellcodeausschnitt 5.2 zeigt exemplarisch ein Shared-Attribute, welches die IP-Adresse des Zielgeräts mit VLC-Player beinhaltet. Auf die Shared-Attributes können alle Erweiterungen zugreifen. Sie stellen somit auch eine Möglichkeit zum Datenaustausch zwischen ihnen her. Des Weiteren lassen sie sich als Session-Attribute nutzen, welche nur zur Laufzeit der App existieren. In diesem Fall sind sie dann nicht Bestandteil der XML-Datei, sondern werden erst zur Laufzeit erzeugt. Ihr Schlüssel muss trotzdem eindeutig sein. Die Shared-Attributes lassen sich aber auch zur Definition von globalen Parametern verwenden. Ein globaler Parameter ist beispielsweise die exemplarisch gezeigte IP-Adresse des Zielgeräts mit dem VLC-Player. Diese steht der Erweiterung zur Ansteuerung des VLC-Players somit immer zur Verfügung, weshalb sie nicht in jedem Menüelement, das einen Befehl an diesen beinhaltet, als Parameter definiert sein muss.

```
<shared_attributes>
```

```
2       <shared_attribute key="/vlc/ip">
3             <type>String</type>
4             <value>192.168.137.1</value>
5       </shared_attribute>
6       ...
7 </shared_attributes>
```

Quellcodeausschnitt 5.2: Shared-Attribute in der XML-Datei.

Das Element `menu_items` beinhaltet sämtliche Menüelemente, die in der Menüstruktur enthalten sind. Ein Menüelement wird repräsentiert durch ein Element `menu_item`. Der Quellcodeausschnitt 5.3 zeigt die Definition des Menüelements zur Auswahl des Fernsehsenders ZDF. Das Attribut `id` dient zur eindeutigen Identifizierung eines Menüelements. Diese ist erforderlich, um weiter unten in der XML-Datei die Menühierarchie zu definieren. Das Unterelement `names` beinhaltet mindestens ein Element `name`. Dadurch, dass es von diesem mehrere geben kann, lässt sich eine Mehrsprachigkeit realisieren. Die jeweilige Sprache eines `name` Elements definiert das Attribut `language`. Des Weiteren besitzt jedes `name` Element sowohl eine Langversion als auch eine Kurzversion als Bezeichnung. Hierbei handelt es sich um die Bezeichnung, die auch auf den Button zu sehen ist. Der Prototyp lässt sich zu einem späteren Zeitpunkt so erweitern, dass in Abhängigkeit des zur Verfügung stehenden Platzes auf dem Button entweder die Lang- oder Kurzversion zu sehen ist. Darüber hinaus gibt es eine aussprechbare Version der Bezeichnung. Diese wird zur Sprachwiedergabe mittels TalkBack genutzt, um auch die Barrierefreiheit für blinde und sehbehinderte Nutzer zu gewährleisten (vgl. [Goo16e]). Die `speech_recognition_values` stellen das vierte Unterelement da. Es beinhaltet mindestens eine Bezeichnung, über welche der Button mittels Sprachsteuerung zu einem fortgeschritteneren Entwicklungszeitpunkt betätigbar ist. Der Prototyp verwendet diese Einträge jedoch noch nicht. Das Element `icon_path` ist optional. Über es lässt sich eine Grafik verlinken, die auf dem entsprechenden Button angezeigt wird. Über die Unterelemente `action_class` und `parameters` des Elements `menu_item` ist eine Funktionalität festlegbar, welche beim Betätigen des Buttons ausgeführt wird. Ersteres Unterelement legt die zu verwendende Erweiterung fest. Das Element `source_file_path` beinhaltet den Pfad zu der JAR-Datei, welche die Erweiterung darstellt. Das Element `class_path` enthält den Pfad zu der aufzurufenden Klasse innerhalb der JAR-Datei. In dem Element `parameters` lassen sich Parameter in Form von Schlüssel-Wert-Paaren definieren, die beim Aufruf der Erweiterung an diese übergeben werden. Bei der Definition eines Menüelements zur Auswahl eines Fernsehsenders, ist dessen Position innerhalb der Playlist des VLC-Players als Parameter definiert, sowie eine Bezeichnung für die Aktion welche in der über das Element `class_path` festgelegten Klasse auszuführen ist, um den Fernsehsender auszuwählen.

```
1 <menu_item id="tv_zdf">
2       <names>
3             <name language="Deutsch">
4                   <long_name>ZDF</long_name>
5                   <short_name>ZDF</short_name>
6                   <speakable_name>ZDF</speakable_name>
7                   <speech_recognition_values>
```

```
8                       <speech_recognition_value>ZDF</speech_recognition_value>
9                 </speech_recognition_values>
10          </name>
11    </names>
12    <icon_path>/bfs/menu/icons/channels/tv/zdf.png</icon_path>
13    <action_class>
14          <source_file_path>/bfs/vlc/VlcActionClasses.jar</source_file_path>
15          <class_path>de.ableitner.vlcactions.VlcActionClass</class_path>
16    </action_class>
17    <parameters>
18          <parameter>
19                <name>methodName</name>
20                <value>playTvChannel</value>
21          </parameter>
22          <parameter>
23                <name>itemId</name>
24                <value>27</value>
25          </parameter>
26    </parameters>
27 </menu_item>
```

Quellcodeausschnitt 5.3: Menülement zur Auswahl des Fernsehsenders ZDF in der XML-Datei.

Der vierte und letzte Teil der XMl-Datei zur Definition der Benutzeroberfläche ist das Element `menu_hierarchy`. Es definiert die Struktur des Menüs. Dazu besitzt es ein Unterelement `root_menu`, welches die IDs derjenigen Menüelemente beinhaltet, die Teil des Hauptmenüs, also der obersten, für den Benutzer sichtbaren Menüebene, sind. Die IDs befinden sich jeweils als Attribut in einem Element `sub_menu_item`. Neben dem Hauptmenü gibt es noch Untermenüs. Je Untermenü gibt es ein Element `sub_menu`, welches ein Unterelement von dem Element `root_menu` ist. Der Quellcodeausschnitt 5.4 zeigt exemplarisch die Definition eines Untermenüs anhand der Favoriten-Sender des Fernsehers. Jedes `sub_menu` Element besitzt die ID des Vorgängerknotens als Attribut. Die enthaltenen Menüelemente sind anschließend, wie schon beim Hauptmenü, als Unterelemente definiert. Mit Hilfe des Elements `menu_hierarchy` ist es möglich, im Anschluss an das Parsen der XML-Datei, die einzelnen Menüelemente als baumartige Menühierarchie zu verknüpfen.

```
1  <menu_hierarchy>
2        ...
3        <sub_menu id_of_parent_menu_item="tv_favorites">
4              <sub_menu_item id_of_sub_menu_item="tv_ard"/>
5              <sub_menu_item id_of_sub_menu_item="tv_zdf"/>
6              <sub_menu_item id_of_sub_menu_item="tv_swrBW"/>
7              <sub_menu_item id_of_sub_menu_item="tv_rtl"/>
8              <sub_menu_item id_of_sub_menu_item="tv_sat1"/>
9              <sub_menu_item id_of_sub_menu_item="tv_pro7"/>
10             <sub_menu_item id_of_sub_menu_item="tv_vox"/>
11       </sub_menu>
12       ...
```

```
13 </menu_hierarchy>
```

Quellcodeausschnitt 5.4: Untermenü Favoriten des Menüs für den Fernseher in der XML-Datei.

Implementierung der Menüstruktur

Die Klasse `Menu` beinhaltet die Logik für die Navigation in der Menüstruktur. Sie ist Bestandteil der gleichnamigen Komponente Menu und bildet die Schnittstelle von dieser, welche die Komponente Command Executor nutzt, um die Navigationsbefehle durchzuführen. Der Quellcodeausschnitt 5.5 stellt die Signaturen der vier wichtigsten Methoden dieser Schnittstelle dar. Für jede Navigationsrichtung gibt es eine. Die Methoden `up` und `down` dienen zur vertikalen Navigation um zwischen den in Abbildung 4.2 dargestellten Ebenen zu wechseln. Zur horizontalen Navigation innerhalb eines Untermenüs gibt es die Methoden `left` und `right`. Alle 4 Methoden geben als Ergebnis die Menüelemente zurück, welche die Benutzeroberfläche im Anschluss an den Navigationsschritt darstellt. Da die maximale Anzahl an Buttons konfigurierbar ist, wird diese über den Parameter `countOfMenuItemsToDisplay` übergeben. Dieser limitiert die Anzahl der zurückgegebenen Menüelemente und ermöglicht es zudem, zur Laufzeit festzuhalten, welche gegenwärtig angezeigt werden, was für die horizontale Navigation erforderlich ist. Die Methoden `up` und `down` besitzen zudem noch den booleschen Parameter `usePreviousPosition`. Ist dieser wahr, geben die beiden Methoden die im anzuzeigenden Untermenü jeweils zuletzt sichtbaren Menüelemente zurück. Anderenfalls handelt es sich um die ersten Menüelemente des Untermenüs. Der Parameter `menuItem` in der Methode down übergibt die Referenz auf das Menüelement, dessen Untermenü ausgewählt wurde.

```
1  public List<MenuItem> up(int countOfMenuItemsToDisplay, boolean
       usePreviousPosition){
2     // implementation
3  }
4
5  public List<MenuItem> down(int countOfMenuItemsToDisplay, boolean
       usePreviousPosition, MenuItem menuItem){
6     // implementation
7  }
8
9  public List<MenuItem> left(int countOfMenuItemsToDisplay){
10    // implementation
11 }
12
13 public List<MenuItem> right(int countOfMenuItemsToDisplay){
14    // implementation
15 }
```

Quellcodeausschnitt 5.5: Bestimmung des als nächstes zu betätigenden Buttons.

Implementierung Benutzeroberfläche

Ein Hauptteil der Benutzeroberfläche ist in der Klasse `AbstractActivity` implementiert. Sie gehört zu der Komponente User Interface & Input Methods. Die Funktionalität dieser Klasse beschränkt sich auf das Einfügen der Buttons, welche sie von der Komponente Command Executor erhält, auf deren Implementation der folgende Abschnitt eingeht, sowie die Anzeige von Fehlermeldungen und Debug-Daten. Darüber hinaus enthält die Klasse `AbstractActivity` noch Funktionalität für die in Kapitel 6 beschriebenen Benutzbarkeitstests. Die Klasse `AbstractActivity` besitzt deshalb so wenig Komplexität, da die Ausführung der Navigationsschnitte in der Menüstruktur die Komponenten Menu und Command Executor übernehmen. Aus diesem Grund muss sie lediglich die Benutzeroberfläche aktualisieren, indem sie die vorhandenen Buttons durch die neu erhaltenen ersetzt.

Wie in Anforderung F6 gefordert, ist die Anzahl der gleichzeitig sichtbaren Buttons sowohl in der horizontalen als auch der vertikalen Ausdehnung konfigurierbar. Aus diesem Grund muss sich die Größe eines Buttons dynamisch an den zur Verfügung stehenden Platz anpassen. Dies wird dadurch erreicht, dass keine statischen Abmessungen hinterlegt sind, sondern die Buttons die Layout-Eigenschaften besitzen, das Elternlayout sowohl vertikal als auch horizontal maximal auszufüllen. Bei diesem handelt es sich um ein Linear Layout mit horizontaler Ausrichtung. Für jede Button-Zeile gibt es eines. Damit sich die Buttons innerhalb von diesem die zur Verfügung stehende Fläche gleichmäßig teilen, sind sie gleich gewichtet. Da es mehrere Zeilen mit Buttons geben können soll, befinden sich die Linear Layouts, die jeweils eine Button-Zeile beinhalten, wiederum selbst in einem Linear Layout, das jedoch vertikal ausgerichtet ist. Auch hier wird die gleichmäßige Verteilung des zur Verfügung stehenden Platzes durch eine Gewichtung erreicht. Die Abbildung 5.4 zeigt anhand eines Screenshots den zuvor beschriebenen Aufbau der Benutzeroberfläche.

Da die Anzahl der anzuzeigenden Menüelemente nicht immer der maximalen Anzahl an Buttons entspricht, Letztere aber dennoch eine einheitliche Größe besitzen sollen, füllen die freien Flächen sogenannte Platzhalter-Buttons auf, die weder betätigbar noch sichtbar sind.

Der Quellcodeausschnitt 5.6 beinhaltet die Methode `updateUI` der Klasse `AbstractActivity`. In ihr findet die Aktualisierung der Benutzeroberfläche statt. Hierzu werden zunächst die vorhandenen Linear Layouts, welche jeweils eine Button-Zeile bilden, entfernt. Anschließend wird das Linear Layout für die oberste Button-Zeile in einer eigens dafür zuständigen Methode sowie ein Zähler initialisiert, der die Anzahl der je Linear Layout schon eingefügten Buttons zählt. Im Anschluss daran iteriert eine for-each-Schleife über die Liste mit den neuen Buttons. Innerhalb von dieser werden die Buttons der Reihe nach in ein horizontales Linear Layout eingefügt und der Zähler inkrementiert. Entspricht dieser der maximal zulässigen Anzahl an Buttons in horizontaler Richtung, wird das horizontale Linear Layout in das vertikale Linear Layout eingefügt, ein neues initialisiert sowie der Zähler zurückgesetzt. Im letzten if-Ausdruck wird in jedem Button noch eine Referenz auf dessen linken und rechten Nachbarn gesetzt. Dadurch ist es möglich, dass der Benutzer bei der Eingabemethode Tastatur, wenn er möchte, mit den beiden Pfeiltasten links und rechts nicht nur innerhalb einer Button-Zeile sondern

über diese hinweg, navigieren kann. Sprich er kann, aber muss nicht, die Pfeiltasten aufwärts oder abwärts benutzen, um die Button-Zeile zu wechseln.

```
1  public void updateUI(List<MyButton> newButtons) {
2     this.linearLayoutForButtons.removeAllViews();
3     int buttonsOfOneHorizontalLinearLayout = 0;
4     LinearLayout horizontalLinearLayout = this.createLinearLayout();
5     int id = 0;
6     Button previousButton = null;
7     for(Button button : newButtons){
8        horizontalLinearLayout.addView(button);
9        buttonsOfOneHorizontalLinearLayout++;
10       if(buttonsOfOneHorizontalLinearLayout ==
             Settings.getInstance().menuGetNumberOfHorizontalButtons()){
11          this.linearLayoutForButtons.addView(horizontalLinearLayout);
12          buttonsOfOneHorizontalLinearLayout = 0;
13          horizontalLinearLayout = this.createLinearLayout();
14       }
15       if(previousButton != null){
16          previousButton.setNextFocusRightId(button.getId());
17          button.setNextFocusLeftId(previousButton.getId());
18       }
19       previousButton = button;
20    }
21 }
```

Quellcodeausschnitt 5.6: Einfügen neuer Buttons zur Aktualisierung der Benutzeroberfläche.

Die Instanziierung der Buttons übernimmt die Klasse `ButtonFactory`. Sie besitzt statische Methoden, über deren Aufruf jeweils ein bestimmter Button erstellbar ist. Grundsätzlich kann zwischen 3 unterschiedlichen Arten von Buttons unterschieden werden. Dabei handelt es sich um den Platzhalter-Button, einen Button mit Beschriftung sowie einen mit Grafik und Beschriftung. Bei ersteren 2 handelt es sich um die Klasse `MyButton`, welche von der Android Klasse `Button` erbt. Die Klasse `MyButton` kann im Vergleich zu dieser zusätzlich eine Refernz auf eine Instanz einer Klasse, welche die Schnittstelle `ICommand` implementiert, besitzen. Diese ist erforderlich zur Ausführung der Funktionalität des jeweiligen Buttons in der Komponente Command Executor. Der Platzhalter-Button besitzt eine solche Referenz nicht, da er keine Funktionalität enthält. Aus diesem Grund werden seine Sichtbar- und Klickbarkeit schon bei der Erzeugung in der Klasse `ButtonFactory` auf false gesetzt. Für beide Buttons mit Beschriftung ist die Schriftgröße von dieser einstellbar, wie in Anforderung F7 verlangt. Für die Buttons mit Grafiken gibt es nochmals eine eigene Klasse `MyIconButton`, welche von `MyButton` erbt. In ihr ist die Methode `onDraw`, welche das Android Betriebssystem zum Zeichnen des

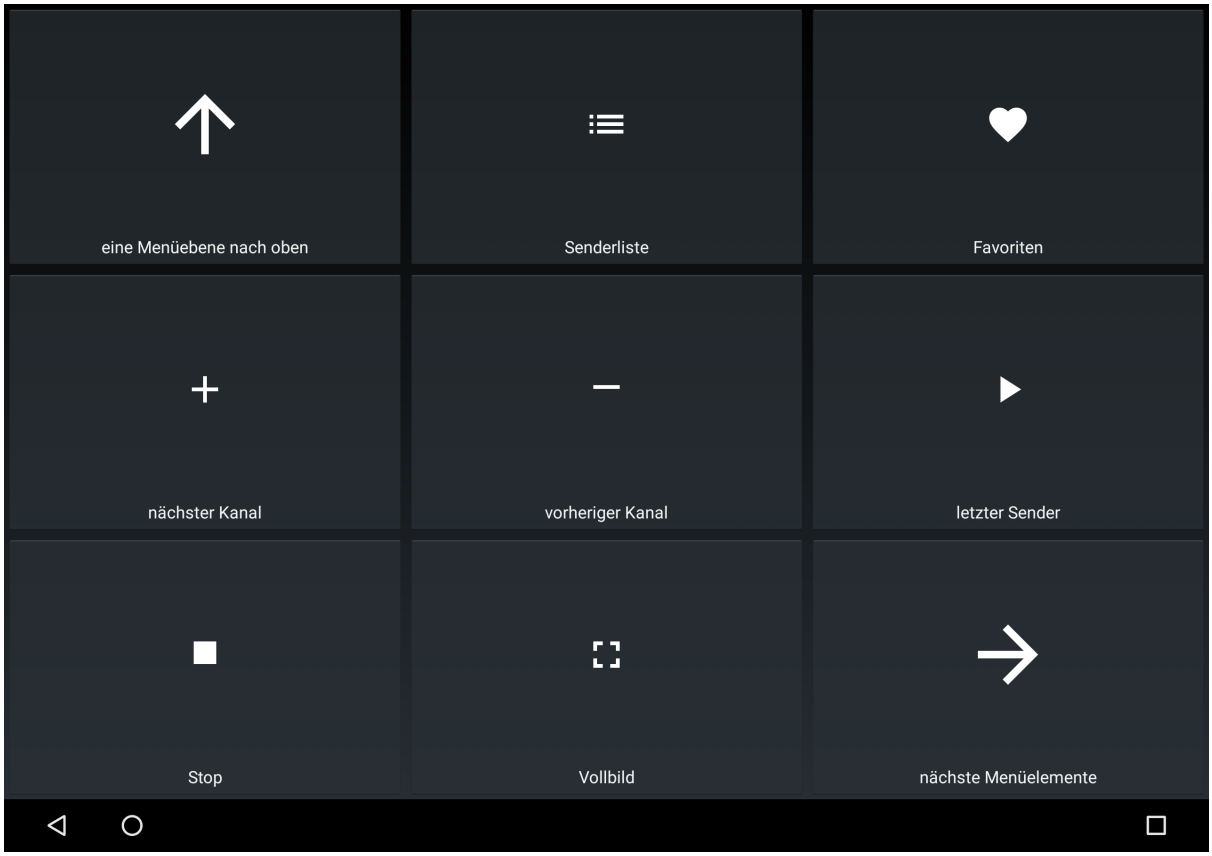

Abbildung 5.4.: Screenshot von der Benutzeroberfläche. (Die Icons auf den Buttons stammen von: https://material.io/icons/ (16.12.2016))

Buttons auf der Benutzeroberfläche aufruft, überschrieben um die Grafik einzufügen. Bei dieser handelt es sich um jene, die in der in Abschnitt 5.4.1 beschriebenen XML-Datei zu Definition einer Benutzeroberfläche durch Angeben eines Dateipfads zu dieser im Element `icon_path` festlegbar ist. Da die Anzahl der Buttons sowie die Schriftgröße konfigurierbar sind und die Button-Beschriftungen unterschiedlich lang sind, müssen die Grafiken jeweils so skaliert werden, dass sie sich in den noch freien Bereich eines Buttons einfügen. Diese Skalierung findet in der überschriebenen `onDraw` Methode statt. Im Anschluss daran ruft diese noch die `onDraw` Methode der Oberklasse auf.

Insbesondere bei einer hohen Anzahl an Buttons und der Verwendung von Grafiken mit einer großen Dateigröße ist es zunächst zu einer für den Benutzer wahrnehmbaren Verzögerung beim Aktualisieren der Benutzeroberfläche gekommen. Dies lies sich durch puffern der Grafiken nach ihrer erstmaligen Verwendung reduzieren. Hierzu gibt es eigens die Klasse `IconLoader`. Sie puffert die Grafiken in einer `HashMap`. Als Schlüssel dient deren jeweiliger Dateipfad. Die Anzahl der maximal gleichzeitig gepufferten Grafiken ist über eine Konstante festgelegt. Wenn die maximale Anzahl an gepufferten Grafiken erreicht ist, aber eine neue zu puffern ist, entfernt

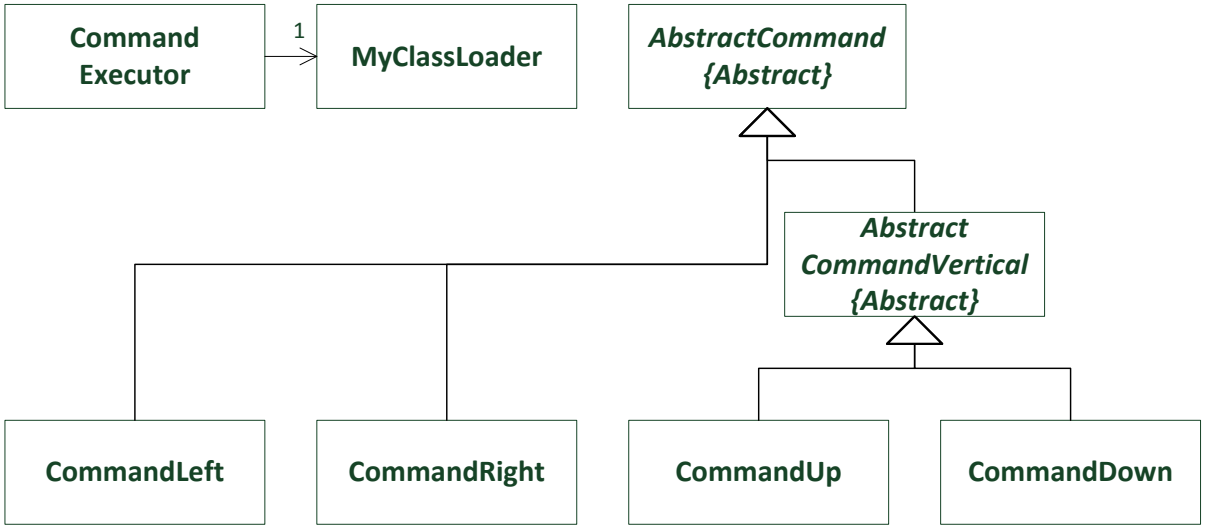

Abbildung 5.5.: Klassendiagramm mit den Klassen der Komponente Command Executor.

die Klasse `IconLoader` eine aus dem Puffer. Die Auswahl von dieser Grafik erfolgt nach dem Least-recently-used-Prinzip.

5.4.2. Komponente Command Executor

Das UML-Klassendiagramm in Abbildung 5.5 zeigt die Klassen der Komponente Command Executor. Sie bilden, abgesehen von der Klasse `MyClassLoader` zusammen das Befehls-Entwurfsmuster (vgl. [GR04, S. 273-277]). Insgesamt gibt es 4 Befehle. Für jede Navigationsmöglichkeit innerhalb der Menüstruktur einen. Jeder Befehl hat seine eigene Klasse. Die Gemeinsamkeiten von diesen sind in den abstrakten Oberklassen `AbstractCommand` sowie `AbstractCommandVertical` zusammengefasst. Im vorherigen Abschnitt 5.4.1 wurde erwähnt, dass jeder Button mit Ausnahme der Platzhalter-Buttons, eine Referenz auf eine Instanz einer Klasse besitzt, welche die Schnittstelle `ICommand` implementiert. Dabei handelt es sich jeweils um eine der 4 Klassen `CommandUp`, `CommandDown`, `CommandLeft` und `CommandRight`. Das UML-Sequenzdiagramm in Abbildung 5.3 stellt den Ablauf dar, wenn der Benutzer einen Button betätigt. Teil dieses Ablaufs ist auch die Komponente Command Executor, denn die zu ihr gehörende Klasse `CommandExecutor` führt die von den Buttons referenzierten Befehle aus. Innerhalb der Klasse ist dafür die Methode `executeCommand` verantwortlich, welche im Normalfall lediglich die gleichnamige Methode `executeCommand` der Schnittstelle `ICommand` aufruft. Normalfall deshalb, weil sie im Rahmen der in Abschnitt 6.2 beschriebenen qualitativen Benutzbarkeitstests auch noch die Protokollierung der Benutzereingabe durchführt.

Die Methode `executeCommand` der Klasse `CommandDown` führt zunächst den Aufruf einer Erweiterung aus, wenn in der XML-Datei für das entsprechende Menüelement eine spezifiziert ist, was nicht immer der Fall ist. Manchmal betätigt der Benutzer nur deshalb einen Button mit

einer Referenz auf eine Instanz der Klasse `CommandDown`, um in ein Untermenü zu gelangen, das sich eine Ebene tiefer in der Menüstruktur befindet. Ist eine Erweiterung hinterlegt, ruft die Methode `executeCommand` die Methode `executeAction` der als Singleton realisierten Klasse `MyClassLoader` auf. Diese bekommt unter anderem als Parameter den Verzeichnispfad zu der Erweiterung beziehungsweise der JAR-Datei sowie einen Klassenpfad zu einer Klasse in dieser übergeben. Dadurch lässt sich diese Klasse mittels Java Reflections instanziieren. Dieser Vorgang ist im Quellcodeausschnitt 5.7 zu sehen, welchem die Fehlerbehandlung zum Zweck der Übersichtlichkeit fehlt. Daneben wird noch eine Reihe weiterer Parameter übergeben. Diese sind zum einen die Shared-Attributes sowie Parameter, jeweils in Form einer `HashMap`, eine Referenz auf die Benutzeroberfläche als Rückrufschnittstelle und ein Kontext-Objekt. Letzteres beinhaltet androidspezifische Funktionen, welche in den Erweiterungen benötigt werden. Im Quellcodeausschnitt 5.7 wird in Anschluss an die Instanziierung der Klasse eine Methode `action` ausgeführt. Ihr Aufruf erfolgt in einem eigenen Thread um sicher zu stellen, dass eine Erweiterung die Ausführung der App zu Steuerung des barrierefreien Smarthomes nicht blockieren kann.

```
1  public void executeAction(String dexFilePath, String actionClassPath,
       ConcurrentHashMap<String, SharedAttribute> sharedAttributes, final
       ConcurrentHashMap<String, String> parameters, ICallbackUserInterface
       callbackUserInterface, Context context){
2      String fullPath = Environment.getExternalStorageDirectory() +
           dexFilePath;
3      String optimizedDirectoryPath =
           context.getCodeCacheDir().getAbsolutePath();
4
5      DexClassLoader dexClassLoader = new DexClassLoader(fullPath,
           optimizedDirectoryPath, null, this.getClass().getClassLoader());
6      Class<Object> actionClass =
           (Class<Object>)dexClassLoader.loadClass(actionClassPath);
7
8      Constructor<Object> actionClassConstructor =
           actionClass.getConstructor(ConcurrentHashMap.class,
           ICallbackUserInterface.class, Context.class);
9
10     final Object actionClassInstance = (Object)
           actionClassConstructor.newInstance(sharedAttributes,
           callbackUserInterface, context);
11
12     final Method methodAction = actionClass.getMethod("action",
           ConcurrentHashMap.class);
13     Runnable runnable = new Runnable() {
14        @Override
15        public void run() {
```

```
16            methodAction.invoke(actionClassInstance, parameters);
17        }
18    };
19    Thread thread = new Thread(runnable);
20    thread.start();
21 }
```

Quellcodeausschnitt 5.7: Aufruf der Erweiterungen mittels Java Reflections.

Danach generiert die Methode `executeCommand` eine Liste mit neuen Buttons. Dieser Schritt findet unabhängig davon statt, um welche Unterklasse es sich von der Klasse `AbstractCommand` handelt. Die einzige Ausnahme stellt die Implementierung der Methode `executeCommand` in der Klasse `CommandDown` dar. Befindet sich der betätigte Button schon auf der untersten Ebene der Menüstruktur, kann es sprichwörtlich nicht mehr weiter runter gehen. In diesem Fall bleibt die Aktualisierung der Benutzeroberfläche aus. In allen anderen Fällen ermittelt sie zunächst, wie viele Menüelemente als nächstes in der Benutzeroberfläche anzeigbar sind. Das ist von der konfigurierbaren maximalen Anzahl an Buttons sowie von den anzuzeigenden Navigations-Buttons abhängig. Letztere müssen nicht immer zu sehen sein. Befinden sich im Hauptmenü (oberste für den Benutzer sichtbare Menüebene) beispielsweise genauso viele oder weniger Menüelemente als die eingestellte maximale Anzahl an Buttons, müssen keine Navigationsbuttons angezeigt werden. Im Sinne einer kontinuierlichen Benutzeroberfläche kann der Benutzer jedoch einstellen, dass diese immer sichtbar, aber in vergleichbaren Fällen wie dem beschriebenen, nicht betätigbar sind. Nachdem die Anzahl der darstellbaren Menüelemente bekannt ist, führt sie den Navigationsbefehl über die Schnittstelle der Komponente Menu aus. Als Rückgabe erhält sie von dieser die anzuzeigenden Menüelemente. Mit Hilfe der Klasse `ButtonFactory` erzeugt sie für jedes Menüelement einen Button. Sofern die Anzahl der erhaltenen Menüelemente kleiner ist, als die der anzeigbaren Buttons, erzeugt sie der Differenz entsprechend zusätzliche Platzhalter-Buttons.

Am Ende übergibt die Methode sämtliche erstellten Buttons in Form einer Liste an die Benutzeroberfläche. Hierfür ruft sie die Methode `updateUI` der Klasse `AbstractMenuActivity` auf, welche eine der drei Rückrufschnittstellen der Komponente User Interface & Input Methods darstellt. Der Quellcodeausschnitt 5.6 in Abschnitt 5.4.1 zeigt die Methode `updateUI`. Aus ihm geht hervor, dass die Methode `updateUI` die übergebenen Buttons entsprechend der Reihenfolge in der Liste von links nach rechts in die jeweilige Button-Zeile und Letztere von oben nach unten in die Benutzeroberfläche einfügt. Deshalb übergibt die Methode `executeCommand` die Buttons schon in der erforderlichen Reihenfolge.

5.4.3. Eingabemethoden Touch, Maus, Tastatur und Sprachsteuerung

Aus der in Abschnitt 5.3 beschriebenen Architektur geht hervor, dass die Eingabemethoden Touch, Maus sowie die Sprachsteuerung die Klasse `DefaultActivity` nutzen. Diese erbt von der abstrakten Klasse `AbstractActivity` und erweitert diese um keine Funktionalität. Der

Grund hierfür ist, dass die Eingabemethoden Touch, Maus und Sprachsteuerung schon das Android Betriebssystems ausreichend unterstützt, weshalb keine eigene Implementierung für sie erforderlich ist. Das gleiche gilt grundsätzlich auch für das Eingabemittel Tastatur. Allerdings ist bei dieser Eingabemethode standardmäßig der oberste linke Button fokussiert, was zur Folge hat, dass der Benutzer durchschnittlich die Pfeiltasten häufiger betätigen muss, um den gewünschten Button auszuwählen, wie wenn zu Beginn ein Button in der Mitte fokussiert ist. Da manche Benutzer aus Gewohnheit jedoch vielleicht Ersteres bevorzugen, können sie einstellen, ob zu Beginn der Button oben links, unten rechts oder in der Mitte fokussiert ist. Sollte die Anzahl der vertikalen und / oder horizontalen Buttons ein Vielfaches von 2 sein, wird ausgehend von der Mitte der obere bzw. linkere Button fokussiert, sofern der Benutzer eingestellt hat, dass zu Beginn der mittlere fokussiert sein soll. Die Fokussierung findet in der Methode `updateUI` der Klasse `KeyboardActivity` statt, da sie jedes Mal aufgerufen wird, wenn neue Buttons in die Benutzeroberfläche einzufügen sind. Für Letzteres ruft sie zunächst die Methode `updateUI` in der Oberklasse, also der `AbstractActivity` auf. Im Anschluss daran fokussiert sie entsprechend der Konfiguration entweder den ersten oder letzten Button aus der Liste über die Methode `requestFocus` der Klasse Button oder berechnet zur Fokussierung des mittleren Buttons dessen Index in der Liste.

5.4.4. Face-Tracking

Das UML-Klassendiagramm in Abbildung 5.6 zeigt die wichtigsten Klassen und Methoden des Face-Trackings. Die Klassen `AbstractActivity` und `DirectFaceTrackingActivity` initialisieren das Face-Tracking und beinhalten einen Teil der face-tracking-spezifischen Funktionalitäten für die Benutzeroberfläche. Das Face-Tracking selbst ist in der Mobile Vision API von Google realisiert (vgl. [Goo16b]). Die Klasse `FaceTracker` verarbeitet die Ergebnisse von dieser, indem sie diese in Benutzereingaben übersetzt. Bei diesen handelt es sich zum einen um die vertikale und horizontale Bewegung des Cursors. Für Erstere ist die Klasse `NoseYTracker` und für Zweitere die Klasse `EulerYTracker` verantwortlich. Beide implementieren die Schnittstelle `ITracker`, über welche die Klasse `FaceTracker` auf sie zugreift. Dadurch lassen sich ohne Anpassungen an dieser die Tracker auswechseln. Zum anderen wertet die Klasse `FaceTracker` für beide Augen des Benutzers deren Öffnungswahrscheinlichkeit aus, um ein Blinzeln zu erkennen und in Folge dessen eine Aktion auszuführen. Die Erkennung des Blinzelns findet entweder in der Klasse `OneBlinkTracker` oder `TwoBlinksTracker statt`.

Die Eingabemethoden 1- und 2-Button-Scanning nutzen ebenfalls das Face-Tracking. Wie in Abbildung 5.2 in Abschnitt 5.3.2 zu erkennen ist, erbt die Klasse `BlinkScanningActivity` ebenfalls von der Klasse `AbstractFaceTrackingActivity`. Da Erstere sowie die Klasse `DirectFaceTrackingActivty` jedoch nicht ein und dieselben Funktionalitäten benötigen, ist die Instanziierung der Klasse `FaceTracker` in der Klasse `AbstractFaceTrackingActivity` generalisiert. Hierzu definiert diese die in dem UML-Klassendiagramm in Abbildung 5.6 zu sehenden abstrakten Methoden, welche ihre Unterklassen implementieren müssen. Je nach

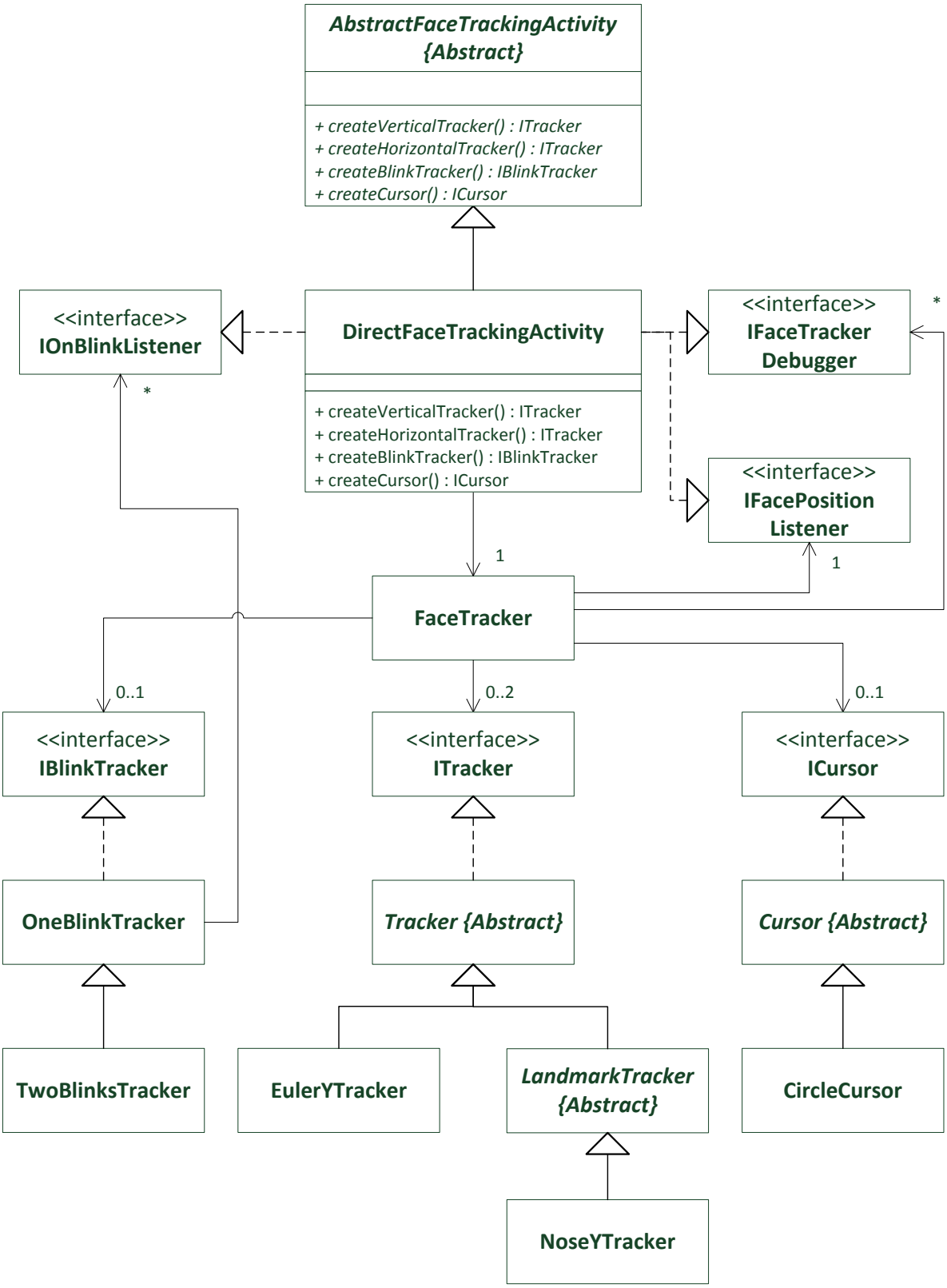

Abbildung 5.6.: Klassendiagramm mit den Klassen des Face-Trackings.

Eingabemethode und deren Einstellungen, wird in diesen Methoden ein Tracker / Beobachter / Cursor erzeugt und der Instanz der Klasse `FaceTracker` als Referenz übergeben.

Die Methode `onUpdate` ist die wichtigste in der Klasse `FaceTracker`. Die Mobile Vision API ruft sie auf, wenn sich an dem Gesicht des Benutzers etwas verändert und übergibt die Ergebnisse als Parameter. Im Anschluss daran leitet die Methode `onUpdate` diese zur Auswertung an die Tracker weiter. Der Quellcodeausschnitt 5.8 zeigt, dass hierfür zunächst alle registrierten Debugger über die Ergebnisse informiert werden. Als Debugger können sich alle Klasse registrieren, welche die Schnittstelle `IFaceTrackerDebugger` implementieren. Auf die Klasse `DirectFaceTrackingActivity` trifft dies zu. Hierdurch kann sich der Benutzer oder Entwickler bei Bedarf die Ergebnisse des Face-Trackings am unteren Rand der Benutzeroberfläche anzeigen lassen. Im Anschluss daran überprüft die Methode `onUpdate`, ob sich die Eingabemethode Face-Tracking gerade im Standby befindet. Standby bedeutet in diesem Fall, dass der Cursor sich durch Kopfbewegungen nicht bewegen und sich mittels Blinzeln auch kein Button betätigen lässt. Dies soll Fehleingaben vermeiden, während der Benutzer sich zwar im Blickfeld der Kamera befindet, aber die App zur Steuerung des barrierefreien Smarthomes gerade nicht bedienen möchte. Wie in dem Konzept in Abschnitt 4.3.1 beschrieben, kann zwischen 3 Arten beim Blinzeln unterschieden werden. In den Einstellungen kann der Benutzer einstellen, ob die Funktion Standby aktiv sein soll und wenn ja, mit welchem Blinzeltyp er diese aktivieren sowie deaktivieren möchte. Befindet sich das Face-Tracking nicht im Standby, leitet die Methode `onUpdate` die Ergebnisse an den horizontalen und vertikalen Tracker weiter. Hierzu ruft sie jeweils die Methode `getPixelValue` der Schnittstelle `ITracker` auf. Diese berechnet anhand der von der Mobile Vision API erhaltenen Ergebnisse die neue horizontale / vertikale Position des Cursors aus. Da es nicht zwingend erforderlich ist, sowohl einen horizontalen als auch vertikalen Tracker zu benutzen, prüft die Methode `onUpdate` im Vorfeld des Aufrufs der Methode `getPixelValue`, ob eine Referenz auf einen Tracker gesetzt ist. Wenn die zukünftige Position des Cursors berechnet ist und eine Referenz auf ihn existiert, informiert die Methode `onUpdate` diesen über seine neue Position. Selbiges gilt auch für den Gesichts-Positions-Beobachter. Unabhängig davon, ob sich die Eingabemethode Face-Tracking im Standby befindet, muss der Tracker für das Blinzeln mit den Daten aus der Mobile Vision API versorgt werden, weil sich hierüber der Standby wieder deaktivieren lässt.

```
1  public void onUpdate(FaceDetector.Detections<Face> detectionResults, Face
       face) {
2      for(IFaceTrackerDebugger faceTrackerDebugger :
           this.faceTrackerDebuggers){
3          faceTrackerDebugger.onUpdate(face);
4      }
5      if(this.standby == false){
6          int xCursorPositionInPixels = -1;
7          int yCursorPositionInPixels = -1;
8          if(this.horizontalTracker != null){
```

```
9           xCursorPositionInPixels =
                this.horizontalTracker.getPixelValue(face);
10      }
11      if(this.verticalTracker != null){
12          yCursorPositionInPixels =
                this.verticalTracker.getPixelValue(face);
13      }
14      if(this.cursor != null){
15          this.cursor.onPositionUpdate(xCursorPositionInPixels,
                yCursorPositionInPixels);
16      }
17      if(facePositionListener != null){
18          this.facePositionListener.onFacePositionUpdate(xCursorPositionInPixels,
                yCursorPositionInPixels);
19      }
20    }
21    if(this.blinkTracker != null){
22      this.blinkTracker.onUpdate(face);
23    }
24  }
```

Quellcodeausschnitt 5.8: Verarbeitung eines Updates der Mobile Vision API von Google.

Die folgenden 2 Abschnitte beschreiben, wie die Verarbeitung der Daten innerhalb der Tracker zum einen für die Steuerung des Cursors und zum anderen zur Erkennung des Blinzelns erfolgt. Im Anschluss daran folgt ein dritter Abschnitt, der sich mit den Anpassungen der Benutzeroberfläche an das Face-Tracking befasst.

Steuerung des Cursors

Der Quellcodeausschnitt 5.9 zeigt die wichtigsten Zeilen der Klasse Tracker. Die 4 Attribute `minPixelValue`, `maxPixelValue`, `inputValueForMinPixelValue` und `inputValueForMaxPixelValue` definieren 2 Intervalle. Die ersten 2 Attribute definieren das Intervall für die horizontale / vertikale Achse des Cursors, auf welcher der Benutzer diesen bewegen kann. Die Werte dafür werden während der Initialisierung des Face-Trackings gesetzt. Betrachtet man den Bereich der Benutzeroberfläche mit den Buttons wie in Abbildung 5.7 dargestellt als Koordinatensystem, welches die Displayauflösung als Grundlage besitzt, wird das Attribut `minPixelValue` mit dem kleinsten noch in der Benutzeroberfläche befindlichen Punkt der Y- / X-Achse initialisiert. Der Wert 0 ist nicht standardmäßig verwendbar, da ein Teil der verfügbaren Displayauflösung manchmal für eine Benachrichtigungs- und / oder Navigationsleiste des Android Betriebssystems verwendet wird. Bei dem Wert für das Attribut `maxPixelValue` handelt es sich folglich um den größten in der Benutzeroberfläche befindlichen Punkt der Y- / X-Achse. Bildlich betrachtet handelt es sich dabei um den unteren / rechten Rand der Benutzeroberfläche der App zur Steuerung des barrierefreien Smarthomes.

Die Attribute `inputValueForMinPixelValue` und `inputValueForMaxPixelValue` bilden das zweite Intervall. Es beinhaltet die möglichen Eingabewerte, also im Fall der prototypischen Realisierung, zum einen wie weit der Benutzer seinen Kopf nach links und rechts drehen muss, um den Cursor an den linken bzw. rechten Rand der Benutzeroberfläche zu bewegen und zum anderen in welchem Bereich des Kamerabildes er seine Nase auf- und absenken kann, um den Cursor auf der vertikalen Achse zu bewegen. Die Werte für diese Intervall legt der Benutzer in den Einstellungen selbst fest. Grundsätzlich ist es so, je größere das Intervall ist desto höher ist die Präzision bei der Bedienung des Cursors, was im Gegenzug jedoch auch eine entsprechende Beweglichkeit erfordert. Benutzer die ihren Kopf nur noch in einem geringen Umfang drehen können, stellen ein entsprechend kleines Intervall für die Eingabewerte ein und müssen aufgrund der dadurch nachlassenden Präzision gegebenenfalls die Anzahl der Buttons je Zeile verringern.

```
1  private int minPixelValue;
2  private int maxPixelValue;
3
4  private float inputValueForMinPixelValue;
5  private float inputValueForMaxPixelValue;
6
7  private IFilter filter;
8
9
10 protected abstract float getInputValue(Face face);
11
12 public int getPixelValue(Face face){
13    int pixelValue =
          this.convertInputValueToPixelValueLinear(this.getInputValue(face));
14    if(pixelValue != -1 && this.filter != null){
15       pixelValue = this.filter.filterValue(pixelValue);
16    }
17    return pixelValue;
18 }
19
20 protected int convertInputValueToPixelValueLinear(float inputValue){
21    float pixelValue = -1;
22    float tmp = (inputValue - this.inputValueForMinPixelValue) /
          (this.inputValueForMaxPixelValue - this.inputValueForMinPixelValue);
23    tmp = Math.max(0, tmp);
24    tmp = Math.min(1, tmp);
25    pixelValue = ((this.maxPixelValue - this.minPixelValue) * tmp) +
          this.minPixelValue;
26    return Math.round(pixelValue);
27 }
```

Quellcodeausschnitt 5.9: Abbildung der Gesichtsposition auf die des Cursors.

Die Klasse `FaceTracker` ruft den vertikalen sowie den horizontalen Tracker über die Methode `getPixelValue` auf, welche auch im Quellcodeausschnitt 5.9 in einer vereinfachten Form zu sehen ist. Diese bekommt als Parameter eine Instanz der Klasse `Face` übergeben, welche die von der Google Mobile Vision API berechneten Werte beinhaltet. Die abstrakte Methode `getInputValue` liest aus diesem Objekt anschließend den benötigten Wert aus. Die Klasse Tracker definiert sie lediglich. Ihre Implementierung erfolgt erst in den Unterklassen, welche jeweils ein Gesichtsmerkmal verfolgen. Hierdurch kann der Benutzer in den Einstellungen zwischen verschiedenen Gesichtsmerkmalen wählen, die zur Eingabe verfolgt werden sollen. Da die Unterklassen dafür alle dieselbe Methode `getInputValue` implementieren, lässt sich die Klasse `FaceTracker` mit diesen beliebig konfigurieren, ohne dafür Anpassungen vornehmen zu müssen. Im Rahmen der prototypischen Realisierung wird der Cursor durch Drehen des Kopfes horizontal bewegt. Den hierfür benötigten Tracker stellt die Klasse `EulerYTracker` dar. Sie liest den gleichnamigen Euler-Y-Winkel aus der Instanz der Klasse `Face` aus (vgl. [Goo16b]). Ursprünglich war es geplant, den Euler-X-Winkel, welcher die Neigung des Kopfes nach vorne beziehungsweise hinten beschreibt, zu verwenden, um den Cursor in vertikaler Richtung zu bewegen (vgl. [Goo16b]). Jedoch ist dieser in der Mobile Vision API von Google bisher noch nicht implementiert, sondern nur in der Dokumentation für die Zukunft schon einmal vorgemerkt (vgl. [Goo16b]). Aus diesem Grund wird stattdessen die Y-Koordinate der Nasenwurzel genutzt. Sie eignet sich dafür besonders gut, da sie zu jenen Gesichtsmerkmalen gehört, die mittels Googles Mobile Vision API bis zu einem Euler-Y-Winkel > -36 beziehungsweise < 36 erkennbar sind und damit bei einer Drehung des Kopfes am längsten verfolgbar sind (vgl. [Goo16b]). Das Auslesen der Y-Koordinate von der Nasenwurzel findet in der Klasse `NoseYTracker` statt.

Im Anschluss daran leitet die Methode `getPixelValue` den Eingabewert an die Methode `convertInputValueToPixelValueLinear` weiter, die ebenfalls in dem Quellcodeauschnitt 5.9 zu sehen ist. In ihr findet die Abbildung des Intervalls für den Eingabewert auf jenes der horizontalen / vertikalen Achse des Cursors statt. Dies entspricht grundsätzlich der im Abschnitt 4.3.1 im Rahmen des Konzepts beschriebenen beiden Formeln. Jedoch wird in den Zeilen 23 und 24 der Multiplikationsfaktor `tmp` auf einen Wertebereich >= 0 und <= 1 limitiert. Das ist nötig, da anders als im Konzept, in der Realität der Eingabewert auch außerhalb des durch die Attribute `inputValueForMinPixelValue` und `inputValueForMaxPixelValue` definierten Intervalls liegen kann. Des Weiteren muss auf die berechnete Position auf der vertikalen / horizontalen Achse des Cursors noch das Attribut `minPixelValue` addiert werden, weil der linke / obere Rand der Benutzeroberfläche mit den Buttons nicht zwingend bei 0 beginnt. Zum Abschluss rundet die Methode `convertInputValueToPixelValueLinear` das Ergebnis noch auf eine Ganzzahl, da die Position des Cursors in Pixel bestimmt wird.

Zurück in der Methode `getPixelValue` kann das Ergebnis anschließend noch mittels eines Filters geglättet werden. Dies ist im Prototyp mittels eines Tiefpassfilters möglich. Der Benutzer kann diesen in den Einstellungen sowohl für den vertikalen als auch den horizontalen Tracker

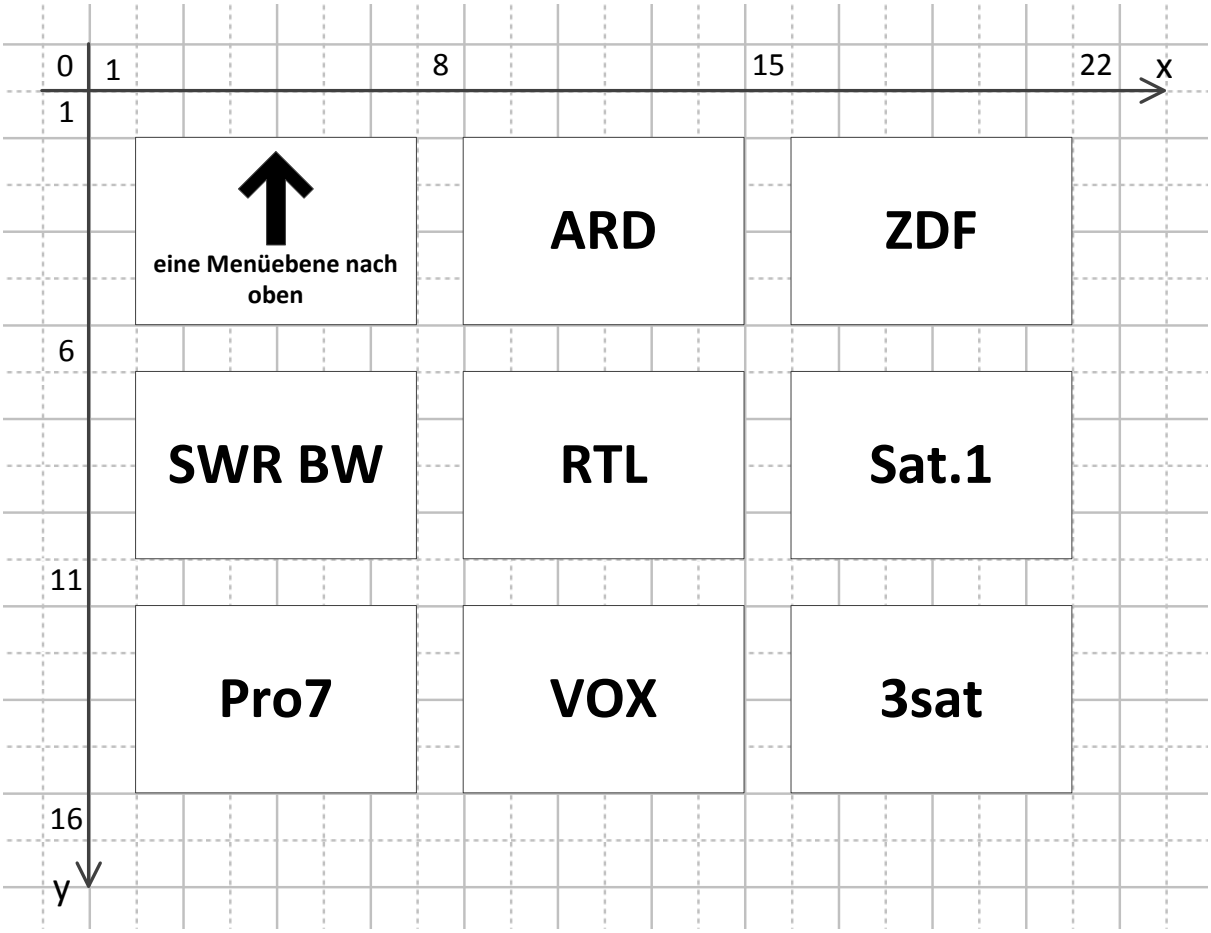

Abbildung 5.7.: Die Benutzeroberfläche in einem Koordinatensystem.

aktivieren und darüber hinaus den Glättungsfaktor festlegen. Die Nutzung des Tiefpassfilters erfolgt über die Schnittstelle `IFilter`. Dadurch ist es möglich, zu einem späteren Zeitpunkt weitere Filter hinzuzufügen und sie gegebenenfalls auch zu kombinieren.

Die Klasse `NoseYTracker` unterscheidet sich von der Klasse `EulerYTracker` dadurch, dass sie nicht nur den Eingabewert aus der Instanz der Klasse `Face` abruft, sondern diese teilweise mittels linearer Regression vorhersagt. Dies ist erforderlich, da zumindest auf dem Pixel C, die Mobile Vision API nicht in jedem Frame alle Gesichtsmerkmale, zu welchen die Nasenwurzel zählt, erkennt. Dies tritt insbesondere dann auf, wenn der Benutzer seinen Kopf bewegt. Der Euler-Y-Winkel für die Kopfdrehung ist hiervon nicht betroffen. Für den Cursor hatte dies ursprünglich zur Folge, dass seine Bewegungen in vertikaler Richtung weniger flüssig waren, da er solange auf seiner letzten Position verblieb, bis die Nasenwurzel wieder erkannt wurde. Durch die Verwendung einer linearen Regression hat sich dieses Problem reduziert. Wenn die Nasenwurzel in einem Frame nicht erkannt wurde, sagt die Klasse `NoseYTracker` deren Y-Koordinate mittels der linearen Regression basierend auf den 4 zuletzt erkannten Y-Koordinaten voraus. Eine weitere Möglichkeit wäre gewesen, neben der Nasenwurzel noch

von weiteren Gesichtsmerkmalen die jeweilige Y-Koordinate zu verwenden und so einzelne fehlende Gesichtsmerkmale kompensieren zu können. Allerdings zeigte sich beim Betrachten entsprechender Log-Ausgaben, dass in den Fällen, wo die Nasenwurzel nicht erkannt wurde, dies meistens auch auf die anderen Gesichtsmerkmale zutraf, weshalb dieser Ansatz nicht weiter verfolgt wurde.

Die im Quellcodeausschnitt 5.8 enthaltene Methode `onUpdate` der Klasse Face-Tracker aktualisiert nach der Berechnung der neuen Cursorposition diesen, indem sie dessen `onPositionUpdate` Methode aufruft, was zur Folge hat, dass der Cursor auf der errechneten Position neu gezeichnet wird. Der Cursor ist als Overlay realisiert. Dadurch ist es möglich, ihn über die Buttons zu zeichnen. Des Weiteren benachrichtigt die Methode `onUpdate` die `DirectFaceTrackingActivity` über die neue Position des Cursors, damit diese jenen Button fokussieren kann, über dem er sich befindet. Dazu implementiert die Klasse `DirectFaceTrackingActivity` die Schnittstelle `IFacePositionListener`.

Blinzelerkennung

Das Erkennen des Blinzelns erfolgt über die Augenöffnungswahrscheinlichkeit. Für diese gibt die Mobile Vision API von Google je Auge einen Wert zwischen 0 und 1 an (vgl. [Goo16d]). 0 bedeutet, dass das Auge geschlossen ist und 1, dass es geöffnet ist (vgl. [Goo16d]). Allerdings handelt es sich bei 0 und 1 um Extremwerte, die in der Praxis nur selten vorkommen. Vielmehr befinden sich die Werte für die Augenöffnungswahrscheinlichkeit im Bereich zwischen 0 und 1. Des Weiteren sind die Werte nach den Erfahrungen des Autors dieser Arbeit von der Größe und dem Aussehen des Auges abhängig. Augen die klein und / oder von Natur aus nicht so weit geöffnet sind, haben im geöffneten Zustand eine geringere Augenöffnungswahrscheinlichkeit, als Augen, die größer und / oder weiter geöffnet sind. Aus diesem Grund kann in den Einstellungen von der App zur Steuerung des barrierefreien Smarthomes sowohl für das linke als auch das rechte Auge jeweils ein Wertebereich angegeben werden, in dem sie das Auge als geschlossen und einen in dem sie es als geöffnet wertet. Bei richtiger Konfiguration stellt diese Einstellungsmöglichkeit eine Bandsperre da, durch welche nur die Extremwerte, also jene die nahe bei 0 oder 1 liegen, berücksichtigt werden. Darüber hinaus lässt sich in den Einstellungen jeweils für das linke und rechte Auge festlegen, wie viele Millisekunden dieses mindestens geschlossen sein muss, damit es als Blinzeln gilt. Dies soll vermeiden, dass unbewusstes Blinzeln als solches von der App erkannt wird. Genauso gibt es jedoch auch eine Maximaldauer, die ein Auge geschlossen sein darf, um es noch als Blinzeln zu werten. Diese Funktionalität ist vergleichbar mit den Maustasten, auf welche ein Benutzer im Falle einer noch rechtzeitig bemerkten Fehleingabe länger als üblich gedrückt halten kann, um beim anschließenden Loslassen kein Klickereignis auszulösen.

Die Blinzelerkennung ist in den Klassen `OneBlinkTracker` und `TwoBlinksTracker` implementiert. Erstere kann nur erkennen, ob mit dem linken oder dem rechten Augen geblinzelt wurde. Zweitere ist zusätzlich in der Lage, zu erkennen, ob der Benutzer mit beiden Augen oder nur einem geblinzelt hat. Beim Blinzeln mit beiden Augen gibt es jedoch das Problem, dass die Updaterate der Mobile Vision API so hoch ist, dass die Benutzer es nur selten schaffen,

beide Augen zwischen ein und den selben Updates zu öffnen. Aus diesem Grund muss nach dem Öffnen des ersten Auges eine Zeit lang abgewartet werden, ob sich auch noch das andere Auge öffnet. Wenn ja, dann gilt dies als Blinzeln mit beiden Augen und wenn nein, als Blinzeln mit dem linken beziehungsweise rechten Auge. Wie lange abgewartet werden soll, lässt sich ebenfalls in den Einstellungen festlegen. Das Abwarten führt in der Realität dazu, dass die, dem Blinzelereignis zugeordnete Aktion, merklich verzögert ausgeführt wird. Welche der beiden Klassen verwendet wird, hängt davon ab, welchen Blinzelarten in den Einstellungen eine Funktionalität zugeordnet ist. Soll beim Blinzeln mit beiden Augen nichts passieren, wird die Klasse `OneBlinkTracker` verwendet, um keine Verzögerung bei der Eingabe zu haben. Ist dem Blinzeln mit beiden Augen hingegen eine Funktionalität zugeteilt, ist die Verwendung der Klasse `TwoBlinksTracker notwendig`. Allerdings kann der Benutzer bei der Klasse `OneBlinkTracker` ebenfalls mit beiden Augen blinzeln, wenn zwischen dem Blinzeln mit dem linken und rechten Auge nicht unterschieden werden muss. Dies führt dann dazu, dass zunächst zwei getrennte Blinzelereignisse erkannt werden. Das spätere ist jedoch ignorierbar, indem der Benutzer in den Einstellungen einen zeitlichen Mindestabstand zwischen dem Blinzeln festlegt. Dieser ist empfehlenswert, da hierdurch vermeidbar ist, dass der Benutzer bei mehrmaligen, unmittelbar hintereinander stattfindenden Blinzeln für den Fall, dass das Blinzeln nicht immer erkannt wird, versehentlich mehrere Buttons ungewollt betätigt.

Wenn die beiden Klassen `OneBlinkTracker` und `TwoBlinksTracker` ein Blinzeln erkennen, rufen sie die Methode `onBlink` in Ersterer auf. Diese benachrichtigt dann die bei ihr registrierten Beobachter darüber, dass ein Blinzeln stattgefunden hat und um welche Art des Blinzelns es sich dabei handelt, sprich ob der Benutzer mit dem linken / rechten Auge oder beiden Augen geblinzelt hat. Um sich als Beobachter registrieren zu können, muss die entsprechende Klasse die Schnittstelle `IOnBlinkListener` implementieren. Bei der Klasse `DirectFaceTrackingActivity` ist dies der Fall, weshalb sie sich als Beobachter registrieren lässt und dadurch von der Klasse `OneBlinkTracker` oder `TwoBlinksTracker` über das Blinzeln in Kenntnis gesetzt wird.

Anpassungen an der Benutzeroberfläche für das Face-Tracking

Die Klasse `DirectFaceTrackingActivity` besitzt die Methode `onBlink`, da sie die Schnittstelle `IOnBlinkListener` implementiert. Die Methode wird von einer der beiden Klassen `OneBlinkTracker` oder `TwoBlinkTracker` aufgerufen, wenn sie ein Blinzeln des Benutzers erkannt haben. Sie prüft zunächst, ob und welche Funktionalität der jeweiligen Blinzelart zugeordnet ist. Diese wäre entweder das Ein- oder Ausschalten des Standbys oder das Betätigen eines Buttons.

Ist Letzteres der Fall, muss die Methode `onBlink` zuerst anhand des Cursors Position herausfinden, welcher Button zu betätigen ist. Hierzu besitzt die Klasse `DirectFaceTrackingActivity` eine Hash-Map vom Typ `TreeMap`, welche die Schnittstelle `NavigableMap` implementiert. Sie erlaubt es, auf die Einträge zuzugreifen, ohne den konkreten Schlüssel zu kennen. Hierzu sucht sie für einen Schlüssel in der Hash-Map bei Benutzung der Methode `floorKey` den nächst kleineren oder größeren beziehungsweise gleichgroßen Schlüssel. Dadurch lässt sich indirekt

Tabelle 5.1.: Exemplarische Werte einer Hash-Map mit jenen der Button-Zeilen.

Schlüssel	Wert
1	NULL
2	Referenz auf die Hash-Map der 1. Button-Zeile (von oben)
6	NULL
7	Referenz auf die Hash-Map der 2. Button-Zeile (von oben)
11	NULL
12	Referenz auf die Hash-Map der 3. Button-Zeile (von oben)
16	NULL

ein Schlüsselbereich für einen Eintrag definieren. Die Klasse `DirectFaceTrackingActivity` nutzt dies, um den Displaybereich, in dem sich ein Button befindet, als Schlüssel für diesen zu verwenden. Dadurch lässt sich anhand der Position des Cursors dann der Button finden, auf dem sich dieser gerade befindet. Für jede Button-Zeile gibt es eine Hash-Map. In dieser befindet sich für jeden Button sowie dem Abstand zwischen diesen oder dem Displayrand ein Eintrag. Als Schlüssel dient dabei die horizontale Position von deren oberen linken Ecke. Die Bildschirmauflösung des Android-Geräts wird dafür als ein Koordinatensystem betrachtet, das seinen Ursprung in der Ecke oben links hat. Die Abbildung 5.7 veranschaulicht dies. Jedes Karo des Koordinatensystems stellt ein Pixel dar. Folglich handelt es sich bei den Schlüsseln um Pixelwerte. Gehört der Schlüssel zu einem Button, so verweist er auf diesen. Gehört er zu einem Abstand, ist der dazugehörige Wert ein NULL-Zeiger. Dadurch ist anhand der horizontalen Position des Cursors, die ebenfalls in Pixeln angegeben ist, feststellbar, ob und wenn ja, auf welchem Button einer Button-Zeile er sich gerade befindet, indem in der Hash-Map nach dem Eintrag mit dem gleichen oder nächst kleineren Schlüssel gesucht wird. Oftmals wird es jedoch so sein, dass auf der Benutzeroberfläche, genauso wie in Abbildung 5.7, mehr als eine Button-Zeile sichtbar ist. Aus diesem Grund befinden sich die Hash-Maps der Button-Zeilen selbst in einer Hash-Map vom Typ `TreeMap`. Als Schlüssel dienen für diese aber die vertikalen Positionen der oberen linken Ecken der Button-Zeilen sowie den Abständen zwischen diesen oder dem Displayrand. Durch die Verschachtelung der Hash-Maps ist, wenn ein Button in Folge eines Blinzelns zu betätigen ist, anhand der vertikalen Position des Cursors ermittelbar, ob und wenn ja, auf welcher Button-Zeile er sich gerade befindet. Ist er auf einer, gibt die Hash-Map, welche jene für die Button-Zeilen beinhaltet, die Referenz auf eine von diesen zurück. In dieser Hash-Map kann anschließend mittels der horizontalen Position des Cursors ermittelt werden, ob und wenn ja, auf welchem Button er sich befindet. Befindet sich der Cursor auf einem, gibt die Hash-Map, der ihn beinhaltenden Button-Zeile, eine Referenz auf ihn zurück. Die Tabellen 5.1 und 5.2 stellen exemplarisch 2 dieser Hash-Maps dar, basierend auf der in Abbildung 5.7 zu sehenden Benutzeroberfläche. Die Tabelle 5.1 beinhaltet die Referenzen auf die Hash-Maps der 3 Button-Zeilen sowie 4 NULL-Zeiger für die Abstände zwischen diesen. In Tabelle 5.2 befinden sich die Schlüssel-Wert-Paare, wie sie sich auch in der Hash-Map der obersten Button-Zeile befinden würden.

Tabelle 5.2.: Exemplarische Werte einer Hash-Map für eine Button-Zeile.

Schlüssel	Wert
1	NULL
2	Referenz auf den Button „eine Menüebene nach oben“
8	NULL
9	Referenz auf den Button „ARD“
15	NULL
16	Referenz auf den Button „ZDF“
22	NULL

Über die aus den Hash-Maps erhaltene Referenz auf den Button lässt sich dieser im Anschluss betätigen. Dazu wird dessen Methode `performClick` aufgerufen, welche wiederum die `onClick` Methode seines Klick-Beobachters aufruft, was dann zu der im Abschnitt 5.4.2 beschriebenen Ausführung eines Befehls in der Komponente Command Executor führt. Durch den Aufruf der Methode `performClick` ist es für den Benutzer jedoch nicht direkt ersichtlich, dass er den Button betätigt hat, da ein visuelles Feedback ausbleibt. Die Studie von Kangas et al. zeigte, dass ein vibrotaktiles Feedback bei der Bedienung eines mobiles Endgeräts mittels Blickgesten hilfreich ist (vgl. [KAR+14]). Dies trifft auf die Eingabemethode Face-Tracking vermutlich ebenfalls zu. Jedoch ist ein visuelles Feedback für die Benutzergruppe des barrierefreien Smarthomes leichter wahrnehmbar, weshalb darüber die Bestätigung erfolgen soll. Der Quellcodeausschnitt 5.10 zeigt die Realisierung des visuellen Feedbacks. Hierzu wird der Status des Buttons zunächst über die Methode `setPressed` auf gedrückt gesetzt. Die Methode `invalidate` veranlasst, dass er neu gezeichnet wird. Anschließend wird ein Runnable-Objekt erzeugt, das die Methode `performClick` aufruft, den Status des Buttons zurücksetzt und ihn erneut zeichnet, jedoch mit einer Verzögerung von 100 Millisekunden. In diesen ist das Layout des Buttons sichtbar, welches der Benutzer sehen würde, wenn er ihn mittels Touch betätigt.

```
1  button.setPressed(true);
2  button.invalidate();
3  button.postDelayed(new Runnable(){
4     @Override
5     public void run(){
6        button.performClick();
7        button.setPressed(false);
8        button.invalidate();
9     }
10 }, 100);
```

Quellcodeausschnitt 5.10: Animation des Button-Klicks beim Face-Tracking.

Damit die Hash-Maps die derzeitig sichtbaren Buttons beinhalten, wird nach jedem Aufruf der Methode `updateUI` in der Klasse `AbstractActivity` die Hash-Map mit den Button-Zeilen geleert und anschließend neu befüllt.

Die Hash-Maps mit den Buttons haben noch eine weitere Funktion. Damit der Benutzer mittels des Face-Trackings ein vergleichbares Bedienerlebnis mit jenem bei den Eingabemethode Tastatur und 1- sowie 2-Button-Scanning hat, fokussiert die Klasse `DirectFaceTrackingActivity` den Button, auf dem sich der Cursor befindet. Ist dieser auf keinem Button, wird auch keiner fokussiert. Welcher Button zu fokussieren ist, ermittelt sie ebenfalls über die Hash-Maps. Das Vorgehen hierzu ist das selbe, wie bei der Betätigung eines Buttons. Um die Positionsänderungen des Cursors zu erhalten, implementiert die Klasse `DirectFaceTrackingActivity` die Schnittstelle `IFacePositionListener`, mit Hilfe derer sie sich bei der Instanz der Klasse `FaceTracker` als Beobachter registrieren kann, welche sie anschließend mittels der in der Schnittstelle definierten Methode `onFacePositionUpdate` über alle Positionsänderungen des Cursors informiert.

5.4.5. Scanning

Bei der Eingabemethode Scanning kann der Benutzer wählen, ob die Buttons von links nach rechts und oben nach unten in einem bestimmten Zeitabstand automatisch fokussiert werden oder er den Fortschritt in Form der Fokussierung selbst steuern möchte. Im Falle von Ersteren lässt sich der Zeitabstand in den Einstellungen festlegen. Des Weiteren kann der Benutzer den automatischen Fortschritt pausieren und im Anschluss daran auch wieder fortsetzen. Wie schon in Abschnitt 5.3.2 erwähnt, beinhaltet die Klasse `Scanner` die Funktionalität für den manuellen sowie automatischen Fortschritt bei der Fokussierung. Hierzu besitzt sie eine Liste, die alle angezeigten Buttons enthält. Beim automatischen Fortschritt iteriert ein eigens dafür zuständiger Thread über diese und fokussiert der Reihe nach die Buttons in der konfigurierten Geschwindigkeit. Zudem besitzt die Klasse `Scanner` eine Referenz auf den zum jeweiligen Zeitpunkt fokussierten Button. Für das Scanning mit dem automatischen Fortschritt ist ein Eingabeereignis erforderlich, um den fokussierten Button zu betätigen sowie optional ein weiteres, um das Scanning zu pausieren und im Anschluss daran wieder fortsetzen zu können. Für das 1-Button-Scanning kann der Benutzer zwei Tasten in den Einstellungen als Eingabeereignisse auswählen. Zum Beispiel die Tastencodes der beiden Tasten des in Abbildung 4.8 gezeigten Button Switch, wobei die zweite Taste nicht zwingend erforderlich ist, da das Anhalten und Pausieren des Scannings optional ist. Vergleichbar verhält es sich beim Scanning mittels Blinzeln. Hier hat der Benutzer die 3 Blinzelarten linkes Auge, rechtes Auge, beide Augen als Eingabeereignis zur Auswahl. In den Einstellungen kann er diesen jeweils eine der beiden Funktionalitäten „Button betätigen“ und „automatischen Fortschritt pausieren / fortsetzen“ zuordnen. Den Blinzelarten, deren Auftreten kein Ereignis auslösen soll, weißt der Benutzer keine Funktionalität zu. Bei der Eingabemethode 2-Button-Scanning beziehungsweise dem Scanning ohne automatischen Fortschritt, sind 2 Eingabeereignisse erforderlich. Eines um den jeweils nächsten Button zu fokussieren und eines um den fokussierten Button zu

betätigen. Das heißt, dass beim 2-Button-Scanning, 2 Tasten beziehungsweise Blinzelarten erforderlich sind. Die Eingabeverarbeitung des 1- und 2-Button-Scannings ist in der Klasse `KeyboardActivity` realisiert. Sie wertet die Eingabeereignisse aus und ruft entsprechend der ihnen zugeordneten Funktionalität die passende Methode in der Klasse Scanner auf. Die Klasse `BlinkScanningActivity` implementiert dagegen die Eingabeverarbeitung für das 1- und 2-Button-Scanning via Blinzeln. Dazu erbt sie von der Klasse `AbstractFaceTrackingActivity`. Wie in Abschnitt 5.4.4 beschrieben, beinhaltet diese die Initialisierung des Face-Trackings. Die erbende Klasse muss dafür die in dem UML-Klassendiagramm in Abbildung 5.6 enthaltenen abstrakten Methoden der Klasse `AbstractFaceTrackingActivity` erweitern. Da nur die Blinzelerkennung erforderlich ist, muss sie lediglich in der Methode `createBlinkTracker` - je nach ausgewählten Blinzelarten als Eingabeereignisse - ein Objekt der Klasse `OneBlinkTracker` oder `TwoBlinksTracker erzeugen`. In den anderen zu implementierenden Methoden genügt es, eine NULL-Referenz zurück zu geben. Während des Scanning wertet die Instanz der Klasse `FaceTracker` in der Methode `onUpdate`, welche auch im Quellcodeausschnitt 5.8 zu sehen ist, nur die Augenöffnungswahrscheinlichkeiten aus und leitet die Blinzelereignisse an die Klasse `BlinkScanningActivity` weiter, welche dann wie die Klasse `KeyboardScanningActivity` die entsprechenden Methoden der Klasse `Scanner` aufruft.

5.4.6. Erweiterungen

Die Erweiterungen lassen sich in Form von JAR-Dateien in die App zur Steuerung des barrierefreien Smarthomes einbinden. Die Klassen in diesen Erweiterungen, die direkt über die Klasse `MyClassLoader` der Komponente Command Executor aufrufbar sein sollen, müssen dafür einen Konstruktor besitzen, welcher als Parameter eine Hash-Map für die Shared-Attributes, eine Rückrufschnittstelle vom Typ `ICallbackUserInterace` für die `AbstractActivity` sowie ein Kontext-Objekt vom Typ `Context` besitzt. Des Weiteren benötigen sie eine Methode namens `action`, welche ebenfalls eine Hash-Map als Parameter für die in dem jeweiligen `menu_item` der XML-Datei definierten Parameter in Form von Schlüssel-Wert-Paaren besitzt, da diese von der Klasse `MyClassLoader` aufgerufen wird, um die entsprechende Funktionalität auszuführen.

VLC-Player

Im Rahmen des Prototyps dient der VLC-Player als Fernseher und Radio. Die Sender für beide befinden sich in einer gemeinsamen Wiedergabeliste. Im Rahmen der Benutzbarkeitstests handelte es sich dabei um im Vorfeld aufgezeichnete Ausschnitte aus Fernseh- und Radiosendungen. Dadurch konnten möglich Probleme aufgrund mangelnder Internetverbindung sowie schlechtem Fernseh- u. Radioempfangs vermieden werden.

Die Kommunikation zwischen der Erweiterung zur Steuerung des VLC-Players erfolgt über dessen Web-Schnittstelle (vgl. [Vid16b]). Sie ermöglicht es, die wichtigsten Funktionen mit Steuerbefehlen mittels HTTP zu senden. Für jeden dieser Steuerbefehle ist ein eigener HTTP-GET-Parameter festgelegt, an den bei Bedarf weitere Parameter anhängbar sind

(vgl. [Vid16b]). Die URL einer HTTP-Anfrage zur Steuerung des VLC-Players hat folgenden Aufbau: `http://:[Passwort]@[IP-Adresse des VLC-Players]:[Portnummer des VLC-Players]/requests/status.xml?[Steuerbefehl als HTTP-GET-Parameter]` (vgl. [Vid16b])

Unter sicherheitstechnischen Aspekten ist es suboptimal, dass das Passwort unverschlüsselt übertragen wird. Da es sich hierbei jedoch ausschließlich um eine prototypische Realisierung handelt, die zudem in keinem öffentlichen Netzwerk verwendet wird, ist die Verschlüsselung des Passworts vernachlässigbar.

Die Erweiterung besteht aus den 2 Klassen `AndroidHttpClient`, die als HTTP-Client dient und zum anderen um die Klasse `VlcActionClass`. Letztere stellt die Schnittstelle der Erweiterung dar, welche die Komponente Command Executor beziehungsweise in dieser die Klasse `MyClassLoader` verwendet. In dem Quellcodeausschnitt 5.3 in Abschnitt 5.4.1 ist erkennbar, dass bei den Menüelementen zur Steuerung des Fernsehers und Radios immer ein Methodenname als Parameter definiert ist. Dieser dient dazu, innerhalb der Methode `action` der Klasse `VlcActionClass`, zu unterscheiden, welcher Steuerbefehl an den VLC-Player zu senden ist. Auf diesen Parameter könnte auch verzichtet werden, wenn es innerhalb der Erweiterung für jeden Steuerbefehl eine eigene Klasse mit passender Implementierung der Methode `action` geben würde. Im Fall des VLC-Players wäre diese Alternative jedoch deutlich aufwendiger zu implementieren gewesen.

Im Folgenden werden die Steuerbefehle sowie deren Verwendungszweck, welche die Erweiterung zur Ansteuerung des barrierefreien Smarthomes nutzt, beschrieben:

- `command=pl_stop`
  Stoppt die Wiedergabe eines Fernseh- / Radiosenders. (vgl. [Vid16b])
- `command=pl_play&id=<id>`
  Startet die Wiedergabe eines Fernseh- / Radiosenders. Bei dem Parameter `id` handelt es sich um die ID des Fernseh- / Radiosenders in der Wiedergabeliste. Wenn der Benutzer die Wiedergabe im Fernseher- beziehungsweise Radiomenü startet, ist die Erweiterung so implementiert, dass der zuletzt geschaute Sender wiedergegeben wird. Dazu gibt es sowohl für das Radio als auch den Fernseher ein Shared-Attribute, das die ID des zuletzt wiedergegebenen Senders beinhaltet. Dieses wird bei jedem Senderwechsel aktualisiert. Da es im Prototyp nicht möglich ist, zur Laufzeit erstellte oder veränderte Shared-Attributes in der XML-Datei zur Definition der Benutzeroberfläche zu speichern, handelt es sich nach einem Neustart der App zu Steuerung des barrierefreien Smarthomes bei den zuletzt wiedergegebenen Sendern immer um die in der XML-Datei definierten. (vgl. [Vid16b])
- `command=pl_next` und `?command=pl_previous`
  Mittels ihnen kann durch die Einträge einer Wiedergabeliste gezappt werden. Die Erweiterung zur Steuerung des VLC-Players nutzt die beiden Befehle, damit der Benutzer zum nächsten oder vorherigen Fernseh- / Radiosender wechseln kann. Da sich in der Wiedergabeliste die Sender vom Fernsehen und Radio befinden, könnte es passieren, dass der

Benutzer durch das Zapping versehentlich das Medium wechselt. Um dies zu vermeiden gibt es sowohl für das Radio als auch das Fernsehen jeweils ein Shared-Attribute mit der kleinsten sowie der größten ID. Dadurch kann die Erweiterung, wenn der Benutzer einen vorherigen oder nächsten Sender auswählen möchte, anhand von der ID des aktuell wiedergegebenen Senders überprüfen, ob es im aktuellen Medium einen vorherigen / nächsten gibt. Wenn nicht, bricht die Erweiterung die Ausführung der Eingabe ab. (vgl. [Vid16b])

- `command=fullscreen`
  Diesen Steuerbefehl nutzt die Erweiterung, damit der Benutzer beim Fernseher den Vollbildmodus ein- und ausschalten kann. (vgl. [Vid16b])
- `command=volume&val=<val>`
  Über diesen Steuerbefehl lässt sich die Lautstärke des VLC-Players über den Parameter `val` entweder als Absolutwert setzen, oder mittels einem vorangestellten Plus / Minus um den Wert des Parameters `val` erhöhen / reduzieren (vgl. [Vid16b]). Die Erweiterung zur Steuerung des VLC-Players nutzt diesen Steuerbefehl zum einen, damit der Benutzer die Lautstärke des Radios / Fernsehers reduzieren und erhöhen kann. Die Größe der Schritte, in welchen die Erhöhung / Reduzierung von der Lautstärke erfolgt, ist jeweils in einem Parameter des Menüelements festgelegt. Zum anderen ermöglicht die Erweiterung dem Benutzer über diesen Steuerbefehl das Radio / den Fernseher stumm sowie den Ton wieder einzuschalten. Um beim Wiedereinschalten des Tons die vorherige Lautstärke setzen zu können, existiert ein Shared-Attribute mit der zum Zeitpunkt vor dem Stummschalten.

Die Implementierung des Zappings durch die Fernseh- beziehungsweise Radiosender zeigt, dass die Verwendung einer gemeinsamen Wiedergabeliste nicht ideal ist. Jedoch bietet die Web-Schnittstelle des VLC-Players keinen Steuerbefehl zur Auswahl einer Wiedergabeliste oder zum Wechsel zwischen mehreren Wiedergabelisten (vgl. [Vid16b]). Eine Alternative zur Vermeidung einer gemeinsamen Wiedergabeliste wäre es, auf diese zu verzichten und stattdessen den wiederzugebenden Sender als Media Resource Locator an den VLC-Player zu senden (vgl. [Vid16b]). Das hätte jedoch zur Folge, dass ein Zapping nicht mehr möglich ist, oder die Erweiterung zur Steuerung des VLC-Players ihre eigene interne Wiedergabeliste benötigt, was diese wiederum komplexer macht.

WLAN-Steckdosenleiste

Die Erweiterung zur Ansteuerung der WLAN-Steckdosenleiste der Wöhlke EDV-Beratung GmbH (vgl. [Wöh16a]) besteht ebenfalls aus 2 Klassen. Die WLAN-Steckdosenleisten besitzt einen eigenen Web-Server, über den sich die einzelnen Steckdosen mittels HTTP-Anfragen ein- und ausschalten lassen (vgl. [Wöh16a] u. [Wöh16b]). Aus diesem Grund handelt es sich bei einer der beiden Klassen ebenfalls um eine mit dem Namen `AndroidHttpClient`. Die Steckdose sowie, ob diese ein- oder ausgeschaltet werden soll, ist in der HTTP-Anfrage als HTTP-GET-Parameter spezifizierbar (vgl. [Wöh16b]). Die URL für diese besitzt folgenden Aufbau:
`http://[IP-Adresse der WLAN-Steckdosenleiste]/websteckdose/cgi-bin/schalten`

`?steckdose_nr=[Nummer der Steckdose]&steckdose_soll=[ob die Steckdose ein- oder ausgeschaltet werden soll]` (vgl. [Wöh16b]) Für die Nummer der Steckdose sind die Zahlen 1, 2 und 3 möglich (vgl. [Wöh16b]). Der zweite Parameter muss den Wert 1 besitzen, um die Steckdose einzuschalten oder den Wert 0, um sie auszuschalten (vgl. [Wöh16b]). Bei der Klasse `WoehlkeActionClass` handelt es sich um die zweite Klasse in der Erweiterung zur Steuerung der WLAN-Steckdosenleiste. Die Methode `action` in ihr, bekommt als Hash-Map 2 Parameter übergeben, die in den dazugehörigen Menüelementen festgelegt sind. Bei diesen handelt es sich zum einen um die Nummer der zu schaltenden Steckdose sowie um deren zukünftigen Schaltzustand.

6. Evaluation

Dieses Kapitel beschreibt die Evaluierung der Anwendung zur Steuerung des barrierefreien Smarthomes. Das Unterkapitel 6.1 konzentriert sich auf die quantitative Beurteilung. Diese untersucht die unterstützten Eingabemethoden in Kombination mit der Benutzeroberfläche auf ihre Geschwindigkeit sowie Genauigkeit. Sprich, wie viele Entscheidungen der Benutzer in einer bestimmten Zeiteinheit tätigen kann.

Das Unterkapitel 6.2 besitzt den selben Aufbau wie das Unterkapitel 6.1, fokussiert jedoch die qualitative Evaluierung der Eingabemethoden sowie der Benutzeroberfläche durch Probanden aus der Zielgruppe, um zu verdeutlichen, dass die entwickelten Eingabemethoden in Abhängigkeit von den Einschränkungen des Benutzers unterschiedlich geeignet sind.

6.1. Quantitative Benutzbarkeitstests

In diesem Unterkapitel ist die Vorbereitung und Instrumentalisierung des Prototypen für die quantitativen Benutzbarkeitstests mit unversehrten Probanden enthalten. Darüber hinaus berichtet es über deren Durchführung und schließt mit einer Auswertung der gewonnenen Daten ab.

6.1.1. Vorbereitung

Der Abschnitt Vorbereitung legt die Instrumentalisierung des Prototyps für die quantitativen Benutzbarkeitstests dar und gibt danach einen Überblick über die verwendete Konfiguration des Prototyps während diesen.

Instrumentalisierung

Die Abbildung 6.1 zeigt einen Screenshot der Einstellungen für die quantitativen Benutzbarkeitstests. Sind diese eingeschaltet, wird in der Methode `onCreate` der Klasse `AbstractMenuActivity` eine spezielle Benutzeroberfläche für die quantitativen Benutzbarkeitstests erzeugt, anstelle des in der XML-Datei definierten Menüs zur Steuerung des barrierefreien Smarthomes. Die Abbildung 6.2 zeigt ebenfalls einen Screenshot. Auf diesem ist die Benutzeroberfläche für die quantitativen Benutzbarkeitstests zu sehen. Sie zielen darauf ab, die

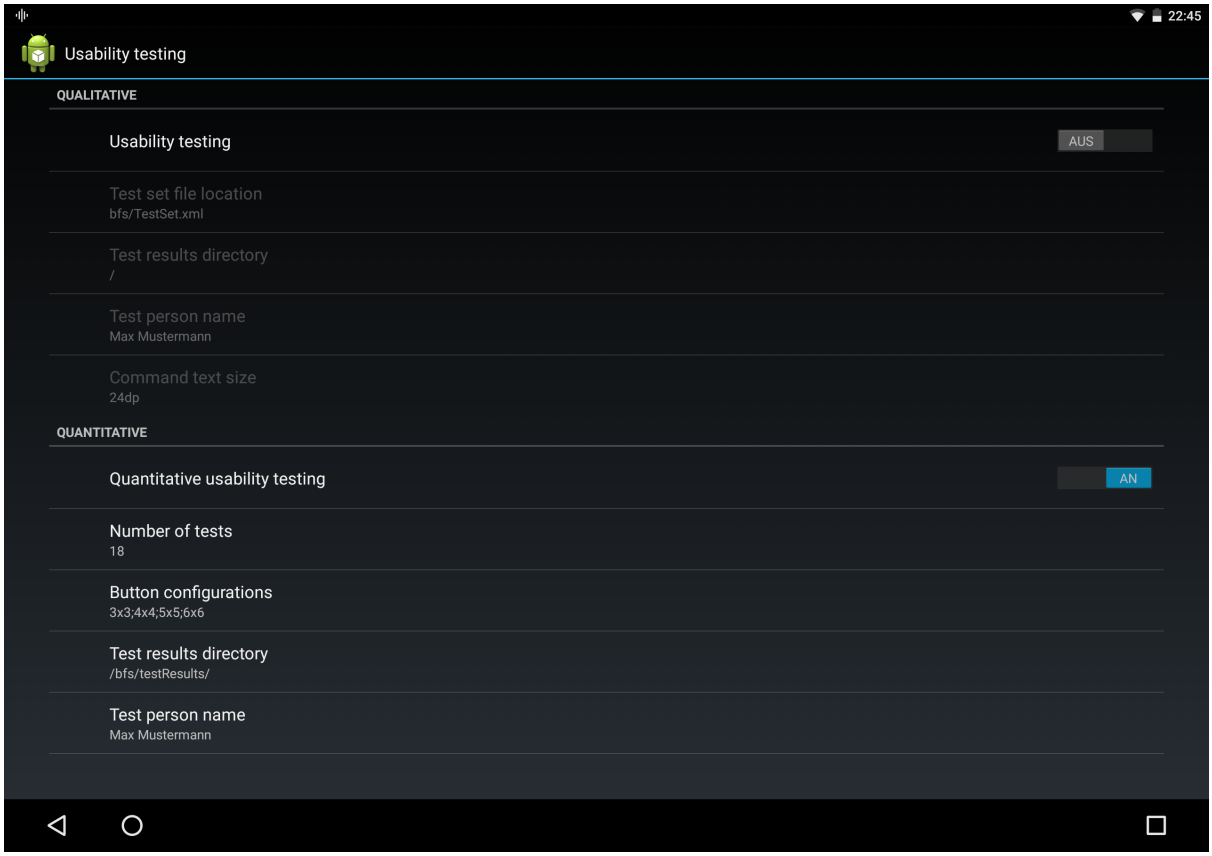

Abbildung 6.1.: Screenshot des Einstellungsmenüs für Benutzbarkeitstests.

Geschwindigkeit und Präzision der verschiedenen Eingabemethode zu evaluieren. Aus diesem Grund gibt es nur zwei Arten von Buttons. Sie unterscheiden sich durch das Icon, bei welchem es sich entweder um ein Kreuzchen oder ein lachendes Smiley handelt. Letzteres besitzt immer nur jeweils einen Button. Der Button mit dem Smiley ist derjenige, den der Benutzer betätigen soll. Auf die Buttons der Benutzeroberfläche für das barrierefreie Smarthome wird verzichtet, da es bei den quantitativen Benutzbarkeitstests um die Geschwindigkeit und Präzision der Eingabemethoden geht. Diese sind am besten messbar, wenn der Proband eine möglichst schlichte Benutzeroberfläche hat. Durch die Verwendung der zwei Icons auf den Buttons muss er keine Button-Beschriftung lesen, keine unbekannten Icons interpretieren und sich auch nicht mit der Navigation innerhalb der Menüstruktur beschäftigen. Stattdessen kann er sich ganz darauf konzentrieren, den Button mit dem Smiley mit der zu testenden Eingabemethode zu betätigen. Ein weiterer Vorteil dieser Buttons ist, dass sich immer exakt die maximale Anzahl an Buttons, entsprechend der zu testenden Konfiguration darstellen lässt. Letzteres ist für die im Kapitel 6.1.3 beschriebene Auswertung der quantitativen Benutzbarkeitstests erforderlich. Mit dem Menü des barrierefreien Smarthomes wäre es hingegen sehr aufwendig, dieses so zu konfigurieren, dass es immer so viele Menüelemente gibt, damit die maximale Anzahl an Buttons zur Auswahl steht.

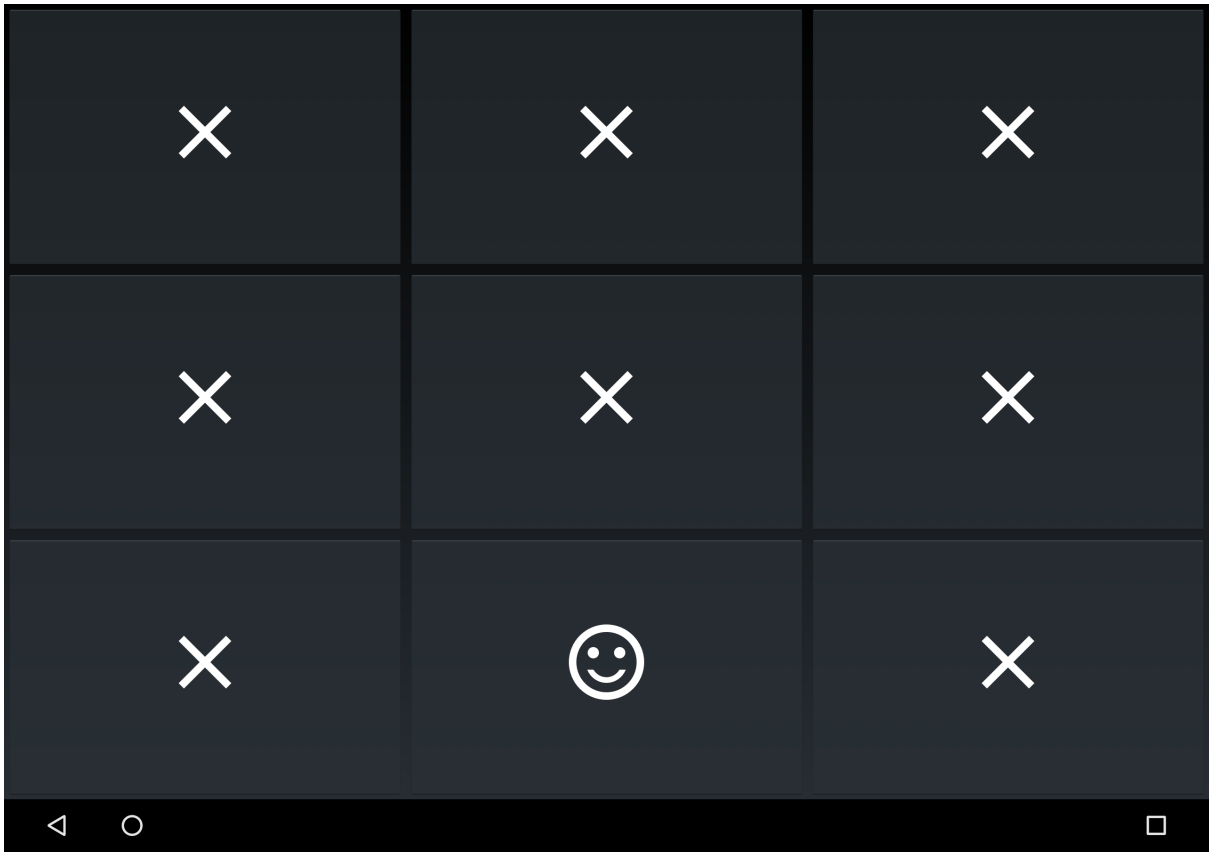

Abbildung 6.2.: Screenshot der Benutzeroberfläche für die quantitativen Benutzbarkeitstests. (Das Smiley- und X-Icon auf den Buttons stammen von: https://material.io/icons/ (16.12.2016))

Bei welchem Button es sich um jenen mit dem Smiley handelt, entscheidet der Zufall. Jedoch gibt es die Einschränkung, dass wenn die Zahl der Testfälle kleiner gleich der Anzahl der Buttons ist, es sich nicht zwei Mal um den selben Button handeln darf. Dies zu gewährleisten ist eine von den Aufgaben der Klasse `BandwidthTester`. Während der Initialisierung des Tests erzeugt sie die Liste `tests`, wie der Quellcodeausschnitt 6.1 zeigt.

```
1 this.tests = new ArrayList<Integer>();
2
3 int limit = ((int)Math.ceil((double)this.numberOfTests /
     (double)this.numberOfButtons)) *
4 this.numberOfButtons;
5
6 int j = 0;
7 for(int i = 0; i < limit; i++){
8    this.tests.add(j);
9    j++;
```

```
10      if(j >= this.numberOfButtons){
11          j = 0;
12      }
13  }
```

Quellcodeausschnitt 6.1: Initialisierung der Testfälle.

Darüber hinaus besitzt die Klasse `BandwidthTester` eine Liste `buttons`, welche die dargestellten Buttons beinhaltet. Im Zuge der Initialisierung wird die Liste `tests` mit Index-Werten der Buttons in der Liste `buttons` gefüllt. Hierzu wird zunächst die Anzahl der Testfälle, welche sich in den in Abbildung 6.1 gezeigten Einstellungen festlegen lassen, durch die Anzahl der Buttons dividiert, zur nächsten Ganzzahl aufgerundet und mit der Anzahl der Buttons multipliziert. Das hieraus resultierende Ergebnis, im Quellcodeausschnitt 6.1 handelt es sich hierbei um die Variable `limit`, ist die Anzahl der Indexe, welche in die Liste `tests` einzufügen sind. Die Anzahl entspricht entweder der der Buttons oder einem Vielfachen von ihnen. Das garantiert, dass alle Buttons mit derselben Wahrscheinlichkeit ausgewählt werden, da ihr Index gleicht oft in der Liste `tests` enthalten ist. Die im Quellcodeausschnitt 6.1 zu sehende Schleife befüllt die Liste `tests` mit den Indexen, indem sie die Variable `j` in diese einfügt, sie in jeder Iteration jeweils um 1 inkrementiert und sie auf 0 zurücksetzt, wenn ihr Wert der Anzahl an Buttons entspricht. Die Anzahl der Iterationen limitiert die berechnete Variable `limit`. Der Quellcodeausschnitt 6.2 zeigt die Auswahl des Buttons.

```
1  public List<MyButton> getButtonsForNextTest(){
2      List<MyButton> buttons = new ArrayList<MyButton>();
3      float random = (float)Math.random() * this.tests.size();
4      int roundRandom = (int) Math.floor(random);
5      int randomButton = this.tests.remove(roundRandom);
6      for(int i = 0; i < this.numberOfButtons; i++){
7          MyButton button;
8          if(i == randomButton){
9              button =
                   ButtonFactory.createCorrectButton(this.abstractMenuActivity);
10         }else{
11             button =
                   ButtonFactory.createFalseButton(this.abstractMenuActivity);
12         }
13         buttons.add(button);
14     }
15     return buttons;
16 }
```

Quellcodeausschnitt 6.2: Bestimmung des als nächstes zu betätigenden Buttons.

Sie erfolgt mit Hilfe der `random` Methode von der Klasse `Math`. Die von ihr erzeugte Zufallszahl wird mit der Anzahl der in der Liste `tests` enthaltenen Indexe multipliziert und danach abge-

rundet. Der hierdurch berechnete Index dient dazu, den Index des Buttons in der Liste `buttons`, aus der Liste `tests` abzufragen, den der Proband als nächstes betätigen soll und der deshalb das Smiley als Icon bekommt. Wenn der Proband den richtigen Button betätigt hat und noch nicht am Ende des Tests angelangt ist, wird die Darstellung der Buttons in der Benutzeroberfläche über die Methode `updateUI` der Klasse `AbstractMenuActivity` aktualisiert. Hierbei handelte es sich um die selbe Methode, die auch genutzt wird, um die Benutzeroberfläche zu aktualisieren, wenn der Benutzer in der Menüstruktur der App zur Steuerung des barrierefreien Smarthomes navigiert und in Folge dessen andere Buttons darzustellen sind. Diesen Vorgang beschreibt das Unterkapitel 5.4 ausführlicher.

Die Abbildung 6.1 zeigt zudem, dass die zu testenden Button-Konfigurationen, in der Form „AnzahlHorizontaleButtonsxAnzahlVertikaleButtons“, getrennt durch Semikolons einstellbar sind. Während des Tests einer Eingabemethode durchläuft der Proband diese Button-Konfigurationen. Für jede gibt es einen Test mit der jeweils eingestellten Anzahl an Testfällen. Zwischen den einzelnen Tests wird ein Dialog angezeigt, wie er auf dem in Abbildung 6.3 gezeigten Screenshot zu erkennen ist. Dieser Dialog dient zum einen als Pause und zum anderen dazu, dass die als nächste zu testende Button-Konfiguration auswählbar ist um beispielsweise einen Test wiederholen zu können. Anderenfalls ist die nächste zu testende Button-Konfiguration schon vorausgewählt und der Dialog muss lediglich über den Button „OK“ geschlossen werden, um mit dem nächsten Test zu beginnen.

Neben der Durchführung der quantitativen Benutzbarkeitstests hat die Klasse `BandwithTester` noch eine weitere Funktion. Sie protokolliert die Ergebnisse eines Tests und schreibt diese in eine CSV-Datei. Folgende Daten erfasst sie grundsätzlich:

- Dauer des Tests
- Anzahl der richtig betätigten Buttons (Buttons mit Smiley)
- Anzahl der falsch betätigten Buttons (Buttons ohne Smiley)
- Datum und Uhrzeit des Testendes
- verwendete Eingabemethode
- verwendete Button-Konfiguration

Des Weiteren berechnet die Klasse `BandwithTester` aus den zuvor genannten Werten noch weitere, welche die in Kapitel 6.1.3 beschriebene Auswertung der quantitativen Benutzbarkeitstests benötigt und schreibt sie ebenfalls in die CSV-Datei. Dies dient lediglich zur Erleichterung der Auswertung.

Konfiguration

Neben der Implementierung der quantitativen Benutzbarkeitstests umfasste die Vorbereitung zudem die Festlegung der Konfigurationen für das Face-Tracking und Scanning. Die Einstellung

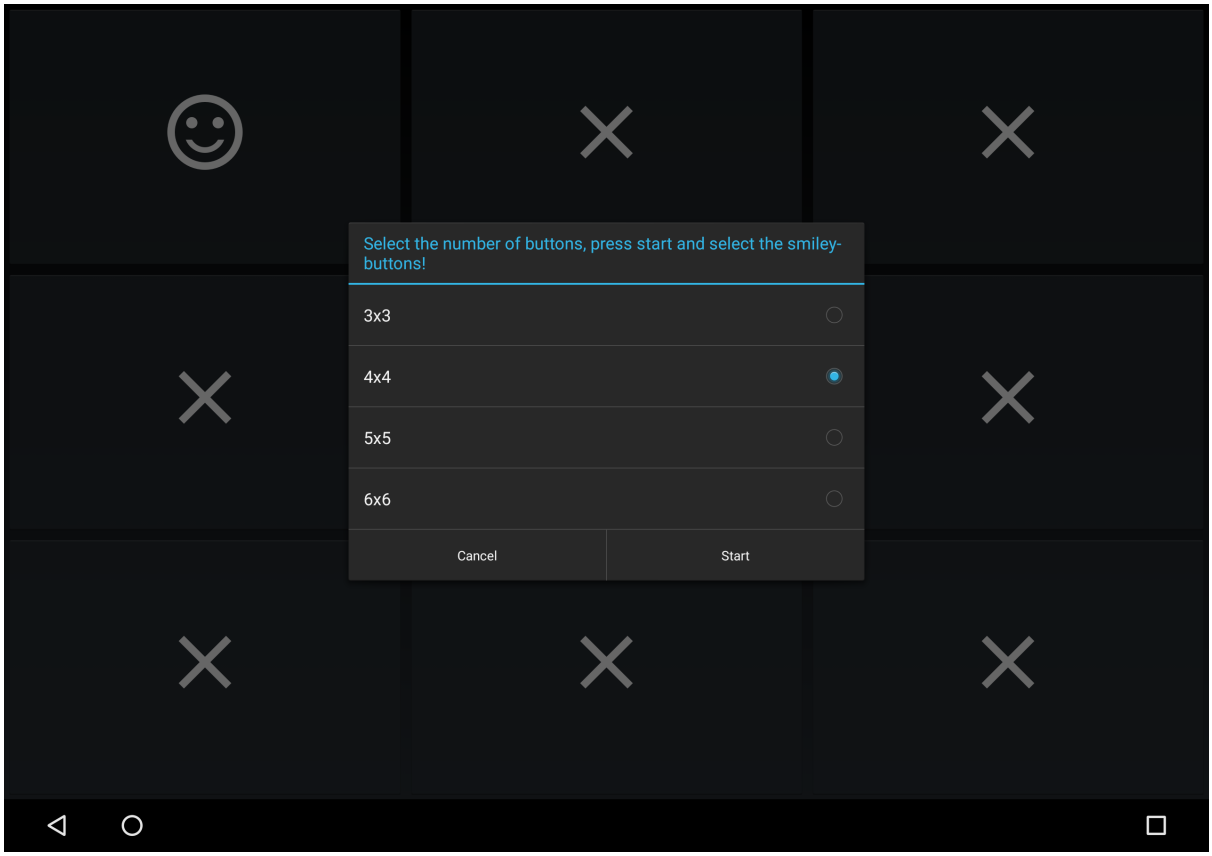

Abbildung 6.3.: Screenshot des Dialogs für die Button-Konfiguration des folgenden Tests. (Das Smiley- und X-Icon auf den Buttons stammen von: https://material.io/icons/ (16.12.2016))

der Parameter für erstere Eingabemethode fand basierend auf den gewonnenen Erkenntnissen während der Entwicklungszeit statt. Beim 1-Button-Scanning hat lediglich ein Parameter Einfluss auf die quantitativen Benutzbarkeitstests. Bei diesem handelt es sich um die Geschwindigkeit beziehungsweise den Zeitabstand, der zwischen der Fokussierung eines Buttons und der des nächsten liegt. Über eine Pilotstudie mit zwei Personen wurde eine geeignete Geschwindigkeit ermittelt. Hierzu testete sie neun verschiedene Geschwindigkeiten im Bereich von 700 bis 150 Millisekunden mit einer 4x4-Matrix als Button-Konfiguration und jeweils 18 Testfällen. Weitere Button-Konfigurationen waren nicht Gegenstand der Pilotstudie, da es in dieser darum ging, wie kurz der zeitliche Abstand zwischen der Fokussierung zweier Buttons mindestens sein muss, damit die Probanden in der Lage sind, den richtigen Button zu betätigen. Dabei stellte sich heraus, dass ein zeitlicher Abstand von 300 Millisekunden zwischen der Fokussierung zweier Buttons am besten geeignet zu sein scheint, da die durchschnittliche Testdauer bei diesem am geringsten war. Das lässt sich dadurch erklären, dass bei einer geringeren Geschwindigkeit es im Durchschnitt länger dauert, bis der Button mit dem Smiley fokussiert und vom Probanden betätigbar ist. Wenn der zeitliche Abstand zwischen der Fokussierung

zweier Buttons jedoch 250 Millisekunden oder weniger beträgt, ist das Scanning so schnell, dass die Probanden den Button mit dem Smiley nicht rechtzeitig betätigen können und ihn dadurch verpassen, was zu Folge hat, dass sie warten müssen, bis dieser erneut fokussiert ist, wodurch sich die Testdauer erhöht. In Folge dessen wurde angenommen, dass mit einer Geschwindigkeit von 300 Millisekunden in den quantitativen Benutzbarkeitstests die besten Ergebnisse für die Eingabemethode 1-Button-Scanning erzielbar sind.

6.1.2. Durchführung

Die Benutzbarkeitstests zur quantitativen Evaluation fanden an drei aufeinander folgenden Tagen mit insgesamt 16 Probanden statt. Sie testeten jeweils folgende Eingabemethoden:

- Touch
- Maus
- Tastatur
- Face-Tracking
- Scanning mit automatischem Fortschritt und Bluetooth Switch
- Scanning mit manuellem Fortschritt und Bluetooth Switch
- Sprachsteuerung

Das Scanning zusätzlich zum Bluetooth Switch noch mittels Blinzeln zu evaluieren, wurde ausgelassen, da die Benutzbarkeitstests sonst je Proband länger als eine Stunde gedauert hätten, wodurch es für die Probanden zu anstrengend geworden wäre.

Die Probanden testeten jede Eingabemethode mit 4 verschiedenen Button-Konfigurationen, um zu untersuchen, wie präzise die einzelnen Eingabemethoden sind, da mit steigender Anzahl der Schaltflächen sich deren Größe reduziert, wodurch sie sich mit manchen Eingabemethoden womöglich schwieriger betätigen lassen. Folgende Anzahlen und Anordnungen von Schaltflächen waren Gegenstand der quantitativen Evaluation:

- 9 Schaltflächen angeordnet in einer 3x3-Matrix
- 16 Schaltflächen angeordnet in einer 4x4-Matrix
- 25 Schaltflächen angeordnet in einer 5x5-Matrix
- 36 Schaltflächen angeordnet in einer 6x6-Matrix

Die Anzahl an Buttons in vertikaler sowie horizontaler Richtung war bewusst in jeder der 4 Konfigurationen gleich gewählt. Hierdurch war sichergestellt, dass das Verhältnis von Höhe und Breite eines einzelnen Buttons bei allen vier getesteten Konfigurationen gleich war. Mit jeder Konfiguration und Eingabemethode mussten die Probanden 18 Buttons betätigen. Für die Konfigurationen mit 25 und 36 Buttons war sichergestellt, dass der Proband pro Eingabemethode keinen Button zwei Mal betätigen soll. Bei den Konfigurationen mit 9 und 16 Buttons, dass keiner mehr als zwei Mal vorkommt. Sowohl die Reihenfolge, in der die Probanden die Eingabemethoden testeten als auch die, in welcher die Konfigurationen hinsichtlich der Button-Anzahl getestet wurden, basierten auf einer Latin square Verteilung. Sie stellte sicher, dass die Probanden nicht alle mit ein und derselben Eingabemethode den Benutzbarkeitstest beginnen und beenden, wodurch mögliche Ermüdungseffekte über alle Eingabemethoden verteilt sind. Für die verschiedenen Konfigurationen bei der Anzahl an Schaltflächen war die Latin sqare Verteilung vor allem bei den für die Probanden eher unbekannten Eingabemethoden, sprich dem 1- und 2-Button-Scanning, der Sprachsteuerung sowie dem Face-Tracking wichtig, um den Lerneffekt zu kompensieren. Dieser trat insbesondere beim Face-Tracking auf, obwohl die Probanden dieses im Vorfeld mehrere Minuten ausprobieren konnten.

Bei der Eingabemethode 1-Button-Scanning wurde die im Vorfeld ermittelte Geschwindigkeit von 300 Millisekunden verwendet. Für die Bedienung mittels Maus wurde die bei Android standardmäßig eingestellte Mauszeigergeschwindigkeit genutzt. Für die Sprachsteuerung durften die Probanden die Sprache wählen, in der sie ihrer Meinung nach am besten sind. 15 Personen entschieden sich für Deutsch und 1 Person für Englisch. Für die Eingabemethode Face-Tracking gab es einerseits für alle Probanden gleichbleibende und andererseits Einstellungen, die individuell an den jeweiligen Probanden anzupassen waren. Der Grund hierfür ist, dass die Probanden unterschiedlich große und geformte Augen hatten, was sich auf die Augenöffnungswahrscheinlichkeit auswirkt. Um das Blinzeln dennoch problemlos für die Eingabe nutzen zu können, waren die Werte zur Erkennung, ob das linke / rechte Auge auf oder zu ist, auf die jeweilige Testperson abzustimmen. Die konstanten Einstellungswerte für die Steuerung mittels Gesichtsverfolgung waren folgende:

- Auflösung der Kamera: 800 x 600 Pixel
- Framerate der Kamera: 30 Frames pro Sekunde
- Zeit, die das linke / rechte Auge mindestens zum Blinzeln geschlossen sein musste: 150 Millisekunden
- Zeit, die das linke / rechte Auge maximal zum Blinzeln geschlossen sein durfte: 2000 Millisekunden
- Auge(n) mit dem / denen das Blinzeln möglich war: linkes Auge, rechtes Auge, beide Augen (Die Probanden durften mit beide Augen blinzeln. Jedoch wurde nicht berücksichtigt, ob sie nur mit dem linken beziehungsweise dem rechten Auge oder beiden Augen blinzeln. Sprich, wenn sie mit beiden Augen blinzelten, wurde das Blinzelereignis von dem Auge ausgelöst, das als erstes wieder offen war oder, wenn sie gleichzeitig wieder offen waren, das vom linken Auge. Dadurch gab es keine Verzögerung durch Abwarten,

ob sich das zweite Auge ebenfalls öffnet. Weshalb es zu der Verzögerung kommen würde, erläutert der Abschnitt 5.4.4.)

- Zeitlicher Mindestabstand zwischen dem Blinzeln: 1499 Millisekunden
- Eingabewert für minimale horizontale Position des Cursors: +20 Grad (Kopfrotation)
- Eingabewert für maximale horizontale Position des Cursors: -20 Grad (Kopfrotation)
- Glättungsfaktor des Tiefpassfilters für die Eingabewerte zur Steuerung der horizontalen Position des Cursors: 16
- Eingabewert für die minimale vertikale Position des Cursors: 240 Pixel (Y-Position der Nasenwurzel im Bild)
- Eingabewert für die maximale vertikale Position des Cursors: 310 Pixel (Y-Position der Nasenwurzel im Bild)
- Glättungsfaktor des Tiefpassfilters für die Eingabewerte zur Steuerung der vertikalen Position des Cursors: 14

Die genannten konstanten Einstellungswerte resultieren aus den Erfahrungen des Autors während der Entwicklungszeit. Nach seinem Empfinden stellen sie eine geeignete Konfiguration dar. Vermutlich gibt es bei diesen Einstellungswerten jedoch noch Optimierungspotenzial, weshalb es für weitere Entwicklungsschritte ratsam ist, in einer ausführlichen Benutzerstudie diverse Einstellungsparameter zu testen.

Zu Beginn des Benutzbarkeitstests wurden von den Probanden jeweils folgende Daten erhoben:

- Alter
- Geschlecht
- Brillenträger (ja / nein)
- Bart (ja / nein)
- Haare (ungefähre Länge, ob sie das Gesicht bedecken)
- verwendete Sprache bei der Sprachsteuerung

Das Alter und Geschlecht wurden standardmäßig erfasst. Ob jemand Brillenträger ist, sein Bart sowie seine Frisur sind im Hinblick auf die Auswertung des Face-Trackings ermittelt worden, falls diese Faktoren sich auf die Gesichtserkennung auswirken. Die verwendete Sprache bei der Sprachsteuerung sollte es ermöglichen, gegebenenfalls sprachenspezifische Unterschiede zu ermitteln beziehungsweise in den Ergebnissen zu berücksichtigen. Beispielsweise, wenn in diesen erhebliche Abweichungen zwischen einer bestimmten Sprache und den übrigen existieren.

Des Weiteren erhielten alle Teilnehmer der Benutzerstudie eine Erklärung zu den Eingabemethoden. Diese umfasste neben einer Einführung in deren Bedienung auch eine Beschreibung ihres Nutzens, sodass die Probanden nachvollziehen konnten, weshalb es wichtig ist, dass sie Eingabemethoden testen, die aus ihrer Situation heraus betrachtet völlig unpraktisch sind. Eine solche Eingabemethode war beispielsweise das Scanning mit dem Button Switch, der aus der Sicht eines unversehrten Benutzers gegenüber einer herkömmlichen Tastatur oder der Touch-Bedienung keinen Vorteil hat. Indem ihnen erklärt wurde, welche Einschränkungen die Zielgruppe des barrierefreien Smarthomes hat und sie deshalb aus der Sicht der Probanden, eher auf umständliche Eingabemöglichkeiten angewiesen sind und diese mit herkömmlichen Eingabemethoden verglichen werden sollen. So konnten sie sich unter der Benutzerstudie mehr vorstellen, was vermutlich auch die Motivation erhöht hat. Da das Face-Tracking für die meisten Probanden vermutlich die schwierigste und unbekannteste Eingabemethode war, konnten sie diese im Vorfeld mehrere Minuten ausprobieren. Dabei ließen sich gleich die Einstellungen zur Erkennung des Blinzelns vornehmen. Die unterschiedliche Körpergröße der Probanden wurde mittels eines höhenverstellbaren Stuhls ausgeglichen. Zum Abschluss des Benutzbarkeitstests durften sich die Probanden bei Interesse jeweils ihre Ergebnisse anschauen, was nebenbei zwei positive Effekte hatte. Zum einen konnte dabei gleich überprüft werden, ob alle Testergebnisse korrekt gespeichert wurden und zum anderen entwickelte sich teilweise noch eine Diskussion über die getesteten Eingabemethoden, in welcher die Probanden Verbesserungsvorschläge einbrachten. Einer der Testteilnehmer äußerte am Ende der Benutzerstudie den Wunsch, die Steuerung via Gesichtsverfolgung mit mehr als 36 Schaltflächen ausprobieren zu wollen.

Während der Durchführung der quantitativen Benutzbarkeitstests gab es auch Probleme, welche nicht unerwähnt bleiben sollen. Eines war, dass manche Teilnehmer mit dem Face-Tracking besser zurecht kamen als gedacht, wodurch die 36 Buttons zu wenige waren, um diese Eingabemethode an ihre Grenzen zu bringen. Als Reaktion darauf, testeten 3 Probanden im Anschluss an das eigentliche Testprogramm das Face-Tracking noch mit bis zu 100 Buttons. Des Weiteren gab es bei dieser Eingabemethode noch Schwierigkeiten, das Blinzeln von Brillenträgern zu erkennen. Jedoch trat dies nur bei 2 von 4 Probanden mit Brille auf, welche das Face-Tracking in Folge dessen ohne diese testeten. Bei den beiden anderen Brillenträgern gab es keine Probleme. Ein Proband dessen Bart, im Vergleich zu den anderen, groß und dicht war, hatte ebenfalls Schwierigkeiten. Diesen äußerten sich darin, dass bei ihm der Cursor viel unruhiger war, sodass für ihn in der Konfiguration mit 36 Buttons eine Bedienung nahezu unmöglich war. Eine leichte Besserung schaffte das Verdecken des Bartes mit einem weißen Blatt Papier. Neben dem Face-Tracking hatte auch die Sprachsteuerung Probleme. Bei Letzterer waren diese zum einen auf Erkennungsprobleme und zum anderen auf die Tatsache, dass sie nicht ausschließlich Bestandteil der getesteten App war, zurückzuführen. Die Schwierigkeiten bei der Spracherkennung trat bei Probanden mit einem Dialekt auf. In Folge dessen kam es insbesondere bei ihnen immer wieder vor, dass ein Test unterbrochen oder abgebrochen wurde, weil die zur Sprachsteuerung verwendete App Voice Access aufgrund einer falsch verstandenen Spracheingabe die App zur Steuerung des barrierefreien Smarthomes geschlossen oder eine andere gestartet hat. Dies kann deshalb passieren, weil mittels der App Voice Access das

gesamte Android Betriebssystem bedienbar ist. Da es sich bei dieser jedoch noch um eine Testversion handelt (vgl. [Goo16m]), ist es nicht auszuschließen, dass das eine oder andere Problem bei der Spracherkennung auch auf sie zurückzuführen ist.

6.1.3. Auswertung

An den quantitativen Benutzbarkeitstests nahmen 16 Probanden teil. Fünf von ihnen waren Frauen und 11 Männer. Sie waren im Durchschnitt 25,44 ±3,45 Jahre alt. Die jüngste Person war 20 und die älteste 34 Jahre alt. Für die Auswertung der quantitativen Benutzbarkeitstests werden die Bandbreiten der jeweiligen Eingabemethoden verglichen sowie die Geschwindigkeit, mit welcher die Probanden die Buttons nacheinander betätigten. Die Bandbreite ist in Bits pro Sekunde angeben. Jedes Bit stellt einen Button dar, der zum Zeitpunkt der Betätigung eines Buttons auf der Benutzeroberfläche sichtbar war. Wenn ein Proband folglich bei einer 3x3-Matrix einen Button betätigte, übertrug er damit 9 Bits. Bei einer 6x6-Matrix wären es 36 Bits gewesen. Dies trägt dem Umstand Rechnung, dass bei einer höheren Anzahl an Buttons zwischen mehr Optionen gewählt werden kann, weshalb eine Entscheidung durch die Betätigung eines Buttons wertvoller ist. Die Geschwindigkeit resultiert aus der Testdauer und der in dieser Zeit getätigten Eingaben. Hat der Proband beispielsweise in 10 Sekunden 5 Buttons betätigt, beträgt die Geschwindigkeit 0,5 Buttons pro Sekunde. Im Abschnitt A.2 des Anhangs befinden sind in den Tabellen A.1 und A.2 für jede getestete Eingabemethode und Button-Konfiguration die maximal, durchschnittlich und minimal übertragenen Bits / Sekunde sowie betätigten Buttons / Sekunde gerundet auf 4 Stellen nach dem Komma.

Im Folgenden findet ein Vergleich aller getesteten Eingabemethoden und Button-Konfigurationen hinsichtlich der Bandbreite und Geschwindigkeit statt. Eine ausführliche Betrachtung der barrierefreien Eingabemethoden Face-Tracking, Sprachsteuerung, 1- und 2-Button-Scanning schließt daran an. Die in diesem Zusammenhang getätigten Aussagen zu einer zu- oder abnehmenden Anzahl an pro Sekunde übertragener Bits beziehungsweise betätigter Buttons beschränken sich auf die Anzahl 9 bis 36 Buttons. Wie weit sich gewisse Trends fortsetzen, kann erst mit Hilfe einer zweiten Studie, die noch weitere Button-Konfigurationen evaluiert, ermittelt werden.

Vergleich der Eingabemethoden

Das Diagramm in Abbildung 6.4 zeigt die durchschnittlich übertragenen Bits pro Sekunde je Eingabemethode und Button-Konfiguration. Bei sämtlichen Eingabemethoden nimmt die Bandbreite mit der Anzahl der Buttons zu, was vermuten lässt, dass mit einer 6x6-Matrix in Kombination mit den getesteten Eingabemethoden das Maximum noch nicht erreicht ist. Des Weiteren zeigt das Diagramm in Abbildung 6.4, dass die Eingabemethoden Touch, Maus und Tastatur ein Vielfaches von den Bandbreiten der barrierefreien Eingabemethoden Face-Tracking, Sprachsteuerung, sowie 1- und 2-Button-Scanning erreicht haben. Die Eingabemethode Touch

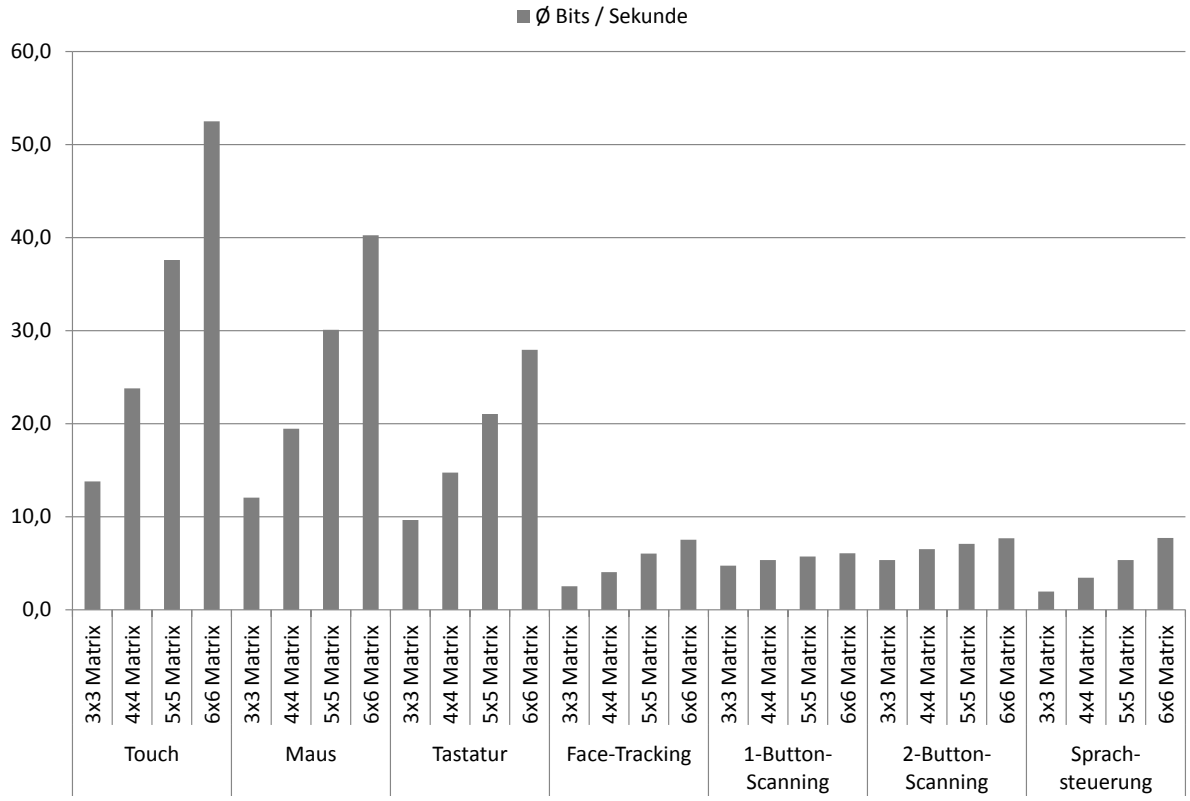

Abbildung 6.4.: Insgesamt übertragene Bits / Sekunde im Durchschnitt.

erzielte bei allen 4 getesteten Button-Konfigurationen die höchste Bandbreite. Das Diagramm in Abbildung 6.4 zeigt die gesamte Anzahl an durchschnittlich übertragenen Bits. Das bedeutet, dass dazu auch Fehleingaben zählen. Eine Fehleingabe liegt dann vor, wenn der Proband anstelle des Buttons mit dem Smiley einen anderen Button betätigte. Viele Fehleingaben können ein Hinweis darauf sein, dass eine Eingabemethode weniger präzise ist als eine andere. Aus diesem Grund unterscheidet das Diagramm in Abbildung 6.5 zwischen korrekt und falsch übertragenen Bits pro Sekunde im Durchschnitt. Die Formulierung falsch übertragene Bits pro Sekunde ist so zu verstehen, dass der Proband mittels der Eingabemethoden eine falsche Eingabe „übertragen“ hat. Die falsch übertragenen Bits sind rot dargestellt und die korrekt übertragenen Bits hellblau. Die Summe aus den richtig und falsch übertragenen Bits ergibt die in Abbildung 6.4 dargestellten Ergebnisse. Aus dem Diagramm in Abbildung 6.5 geht hervor, dass die barrierefreien Eingabemethoden Face-Tracking sowie 1-Button-Scanning die höchsten Bandbreiten bei den falsch übertragenen Bits pro Sekunde haben.

Das Diagramm in Abbildung 6.6 veranschaulicht ausschließlich die im Durchschnitt falsch übertragenen Bits pro Sekunde. Aus diesem lassen sich folgende 2 Erkenntnisse ableiten. Zum einen unterscheiden sich die beiden barrierefreien Eingabemethoden Face-Tracking und 1-Button-Scanning mit den meisten falsch übertragenen Bits pro Sekunde hinsichtlich der Zunahme bei diesen. Beim Face-Tracking nimmt die Anzahl der falsch übertragenen Bits mit

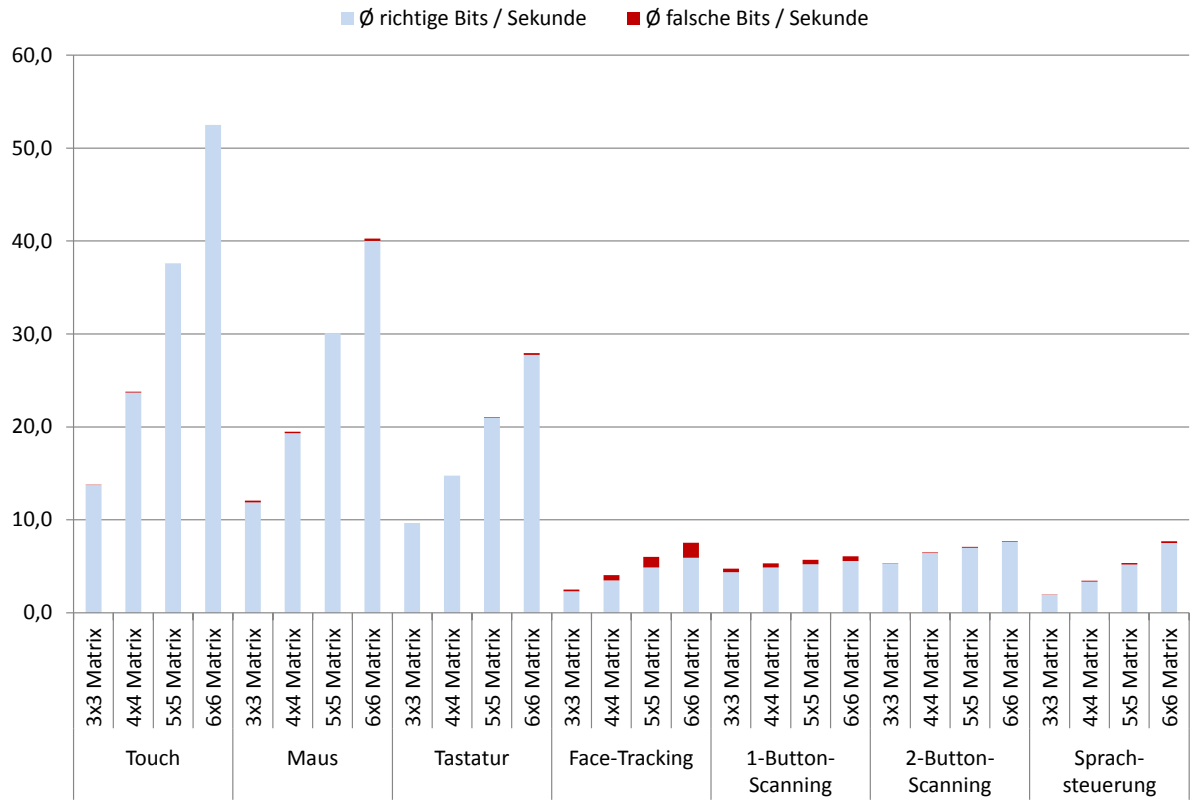

Abbildung 6.5.: Durchschnittlich richtig beziehungsweise falsch übertragene Bits / Sekunde.

der Anzahl der getesteten Buttons zu, was darauf schließen lässt, dass bei einer Konfiguration mit wenigen, es nicht so oft zu Fehleingaben kommt. Beim 1-Button-Scanning ist hingegen genau das Umgekehrte der Fall. Bei den getesteten Button-Anzahlen nimmt die Bandbreite an falsch übertragenen Bits kaum zu, was den Schluss zulässt, dass mit einer steigenden Anzahl an Buttons bis mindestens 36 die Fehleingaben sinken. Zweitens zeigt das Diagramm in Abbildung 6.6, dass es auch bei anderen Eingabemethoden zu Fehleingaben kam. In dem Diagramm in Abbildung 6.5 waren diese aufgrund dessen Skalierung kaum sichtbar. Jedoch ist es offensichtlich, dass die Eingabemethode Touch am besten abschneidet und die barrierefreien Eingabemethoden Face-Tracking und 1-Button-Scanning am schlechtesten. Erstaunlich ist, dass das 2-Button-Scanning und die Sprachsteuerung hinsichtlich der Fehleingaben nicht merklich schlechter sind als Maus und Tastatur. Dies bezieht sich allerdings nur auf die Anzahl der falsch übertragenen Bits. Bei den korrekt übertragenen Bits sind Maus und Tastatur überlegen.

Neben der Bandbreite in Form von Bits pro Sekunde ist es auch interessant, mit welcher Eingabemethoden und Button-Konfiguration sich am schnellsten Entscheidungen treffen lassen. Das Diagramm in Abbildung 6.7 zeigt die Anzahl der im Durchschnitt richtig und falsch betätigten Buttons pro Sekunde je Eingabemethoden und Button-Konfiguration. Auch hier sind Touch, Maus und Tastatur eindeutig besser als die barrierefreien Eingabemethoden.

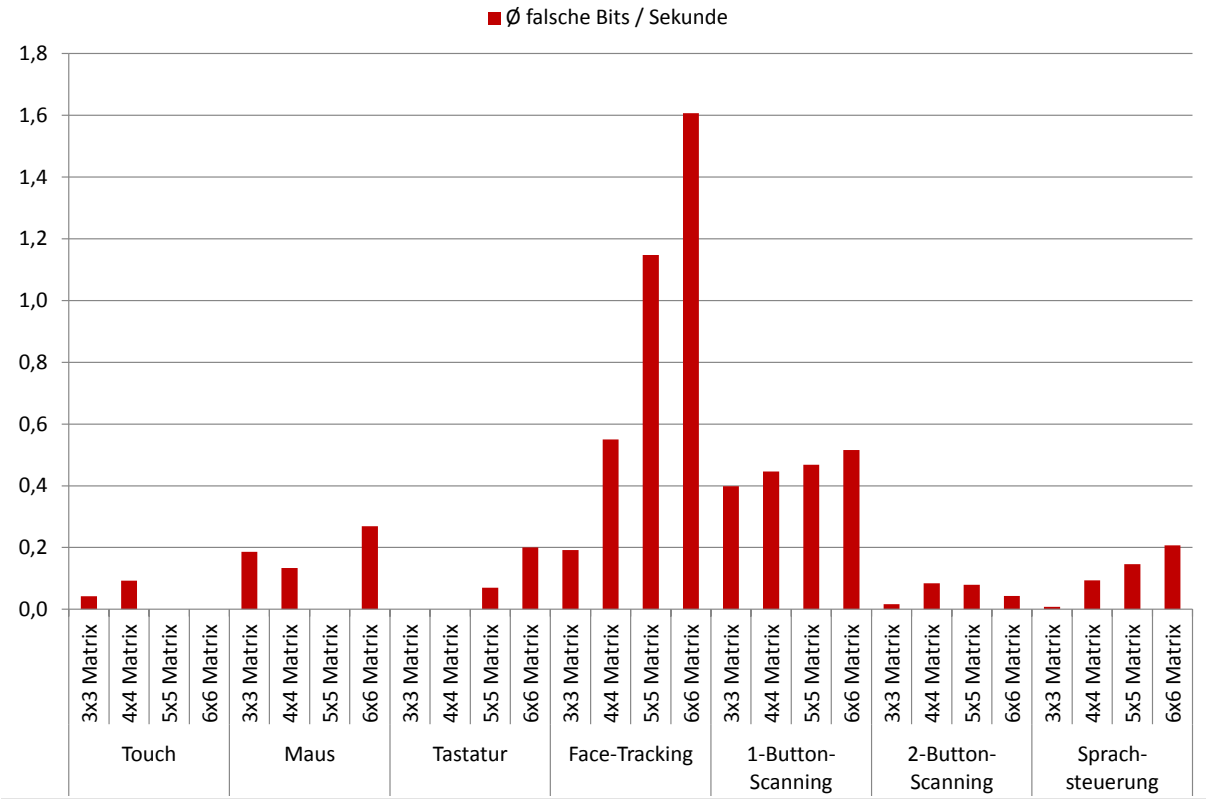

Abbildung 6.6.: Durchschnittlich falsch übertragene Bits / Sekunde.

Innerhalb der ersteren Drei, bleibt die Eingabemethode Touch mit zunehmender Anzahl an getesteten Buttons stabil, was die Geschwindigkeit betrifft. Bei der Maus sowie der Tastatur ist dies nicht der Fall. Bei ihnen lässt sich die mit der steigenden Anzahl an Buttons sinkende Geschwindigkeit auf längere Wege zurückführen. Bei der Maus mussten die Probanden diese bei der 6x6-Matrix im Durchschnitt weiter nach links, rechts, oben und unten bewegen als bei der 3x3-Matrix. Des Weiteren werden die Buttons kleiner, wenn ihre Anzahl steigt, was dazu führt, dass der Mauszeiger präziser und damit langsamer bewegt werden muss. Bei der Tastatur entstehen längere Wege dadurch, dass bei einer höheren Anzahl an Buttons, durchschnittlich mehr Eingaben mit den Pfeiltasten erforderlich sind, bis der richtig Button fokussiert und betätigbar ist.

Bei den barrierefreien Eingabemethoden Face-Tracking sowie 1- und 2-Button-Scanning nimmt die Geschwindigkeit mit steigender Anzahl der Buttons in der Benutzerstudie ebenfalls ab. Jedoch beim Face-Tracking weniger stark als bei den anderen beiden. Bei diesen ist die Ursache darin zu finden, dass vergleichbar zu der Eingabemethode Tastatur, der Zeitaufwand zur Fokussierung des richtigen Buttons beziehungsweise im Rahmen der quantitativen Benutzbarkeitstests jenen mit dem Smiley, im Durchschnitt mit der Anzahl der getesteten Buttons gestiegen ist. Beim 1-Button-Scanning liegt das daran, dass es im Durchschnitt länger dauert, bis der entsprechende Button fokussiert ist. Selbiges trifft auch auf das 2-Button-Scanning zu,

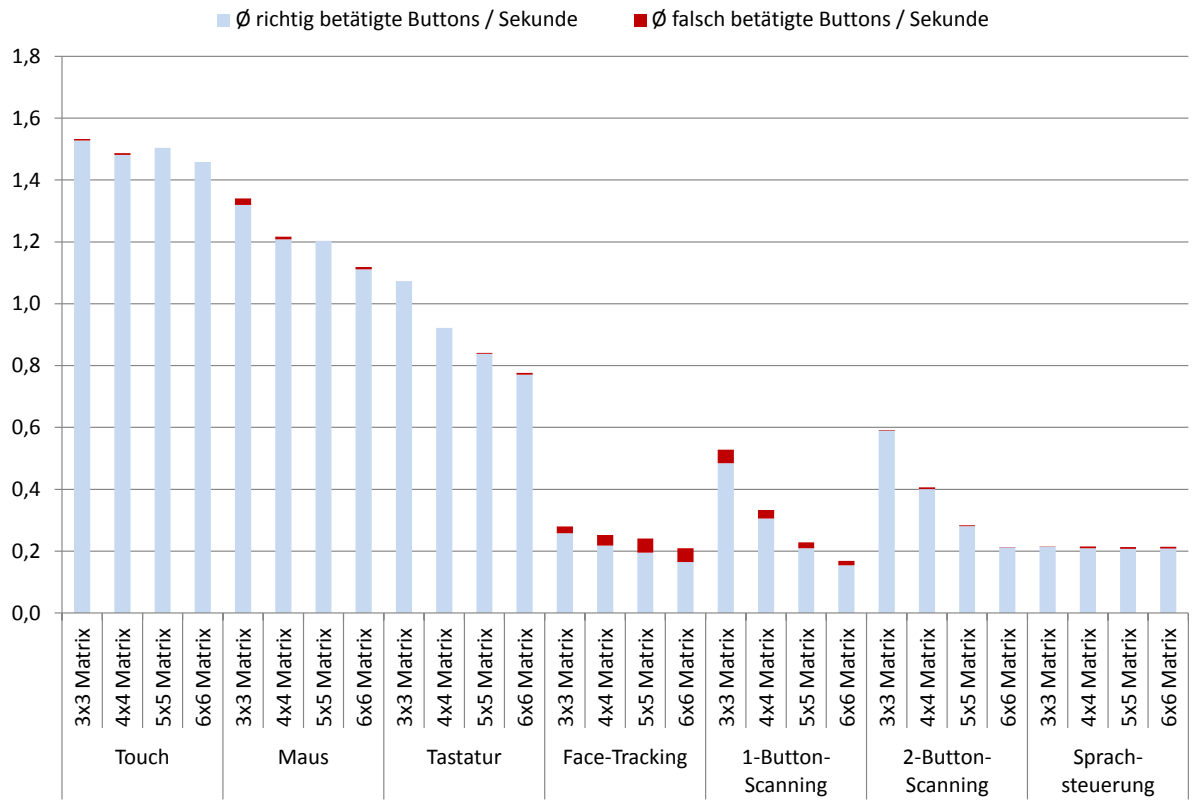

Abbildung 6.7.: Durchschnittlich richtig beziehungsweise falsch betätigte Buttons / Sekunde.

wobei hier der Benutzer über einen Taster jeweils den nächsten Button fokussieren muss, bis er den gewünschten erreicht und betätigen kann. Bei einer niedrigen Anzahl an Buttons sind das 1- und 2-Button-Scanning aus diesem Grund jedoch am schnellsten unter den 4 evaluierten barrierefreien Eingabemethoden, was daran liegt, dass der durchschnittliche Zeitaufwand, der nötig ist, bis der richtige Button fokussiert ist, dann gering ist. Während den quantitativen Benutzbarkeitstests war dies besonders deutlich bei den Button-Konfigurationen in Form einer 3x3- oder 4x4-Matrix der Fall. Das Diagramm in Abbildung 6.7 zeigt auch, dass die Sprachsteuerung die Eingabemethode mit der kontinuierlichsten Geschwindigkeit war.

1-Button-Scanning

Das Diagramm in Abbildung 6.5 zeigt unter anderem die durchschnittlich korrekt sowie falsch übertragenen Bits pro Sekunde mittels der Eingabemethode 1-Button-Scanning. Die zu ihr gehörenden grünen Balken zeigen, dass die Bandbreite mit der getesteten Anzahl der Buttons nur gering zunimmt, insbesondere wenn man bedenkt, dass bei der 3x3-Matrix durch die Betätigung eines Buttons 9 Bits und bei der 6x6-Matrix 36 Bits übertragen werden, was einer Vervierfachung entspricht. Hingegen erhöht sich die durchschnittliche Bandbreite bei 36 Buttons im Vergleich zu 9 Buttons nur um ca. 1 Bit. Das liegt daran, dass mit der Anzahl

der getesteten Buttons, im Mittel die Dauer zunimmt, bis der richtige Button fokussiert ist, da die Geschwindigkeit der Fokussierung konstant ist. Diesen Umstand verdeutlicht auch das Diagramm in Abbildung 6.7. Es zeigt die Anzahl der durchschnittlich korrekt und falsch betätigten Buttons pro Sekunde. Mit Zunahme der Buttons bis einschließlich 36, nimmt diese kontinuierlich ab, was bedeutet, dass der Zeitabstand zwischen den Eingaben zunimmt und darauf zurückzuführen ist, dass es bei einer höheren Anzahl an Buttons länger dauert, bis der richtige fokussiert ist, was sich wiederum negativ auf die Bandbreite auswirkt.

Erfreulich ist, dass die durchschnittlich falsch übertragenen Bits pro Sekunde mit der Anzahl der getesteten Buttons nur minimal zunehmen, wie im Diagramm in Abbildung 6.6 zu sehen ist. Betrachtet man die Anzahl der falsch betätigten Buttons in dem Diagramm aus Abbildung 6.7, dann hat diese mit einer steigenden Anzahl an Buttons in den Tests sogar abgenommen. Ersterer Effekt ist darauf rückführbar, dass die Schwierigkeit für die Probanden beim 1-Button-Scanning war, den richtigen Button, also jenen mit dem Smiley, rechtzeitig, sprich in den 300 Millisekunden, in welchen er fokussiert ist, zu betätigen und diese unabhängig von der Button-Konfiguration ist. Der zweite Effekt, also die Abnahme der Fehleingaben mit steigender Anzahl an Buttons ist dadurch erklärbar, dass die Probanden beispielsweise bei der 3x3-Matrix im Vergleich zur 6x6-Matrix im Mittel weniger Zeit hatten, um den Button mit dem Smiley zu „entdecken“, wodurch sie öfters zu langsam waren und fälschlicherweise den darauf folgenden Button fokussiert haben. Dies war nach Beobachtung des Autors dieser Arbeit insbesondere dann der Fall, wenn sich der Button mit den Smileys in der obersten Button-Zeile links befunden hat. Dies ist plausibel, da das 1-Button-Scanning die Buttons von links nach rechts ausgehend von oben fokussiert. In der Praxis wird die Anzahl der Fehleingaben bei erfahrenen Benutzern vermutlich geringer sein, da sie aufgrund ihrer Erfahrung schon wissen, wo sich der Button, den sie als nächstes zu betätigen haben, befindet. Dadurch fällt die Zeit für das Suchen beziehungsweise den Überraschungseffekt weg.

2-Button-Scanning

Das Diagramm in Abbildung 6.5 zeigt die durchschnittliche Bandbreite der Eingabemethode 2-Button-Scanning. Wie schon beim 1-Button-Scanning nahmen die durchschnittlich richtig übertragenen Bits mit der Anzahl der getesteten Buttons zu, jedoch kommt es auch hier zu einer Reduzierung der Eingabegeschwindigkeit, wie in dem in Abbildung 6.7 enthaltenen Diagramm zu erkennen ist, welches die durchschnittliche Anzahl der richtig und falsch betätigten Buttons pro Sekunde darstellt. Der Grund hierfür ist, wie schon bei der Eingabemethode 1-Button-Scanning der, dass der Zeitbedarf bis zur Fokussierung des richtigen Buttons mit deren Anzahl steigt. Der Vergleich der Bandbreiten von den Eingabemethoden 1- und 2-Button-Scanning zeigt, dass Zweitere eine höhere besitzt. Das ist darauf zurückzuführen, dass die Anzahl der richtig betätigten Buttons pro Sekunden beim 2-Button-Scanning größer ist als beim 1-Button-Scanning, was ein Vergleich mit dem Diagramm in der Abbildung 6.7 zeigt. Dies kann 2 Gründe haben. Möglicherweise waren die Probanden im Durchschnitt schneller beim Fokussieren des nächsten Buttons, als das 1-Button-Scanning. Sprich sie brauchten jeweils weniger als

300 Millisekunden um den nächsten zu fokussieren. Der zweite mögliche Grund ist, dass die Probanden beim 2-Button-Scanning im Vergleich zum 1-Button-Scanning seltener den Button mit dem Smiley „verpassten“ und in Folge dessen fälschlicherweise auch nicht den nachfolgenden Button betätigten, da sie das „Verpassen“ noch rechtzeitig bemerkten, denn das Diagramm in Abbildung 6.7 veranschaulicht, dass es beim 2-Button-Scanning deutlich weniger Fehleingaben als beim 1-Button-Scanning gab. Das „Verpassen“ bedeutete jeweils, dass die Probanden erneut warten mussten, bis der richtige Button fokussiert ist, was zur Konsequenz hatte, dass im Durchschnitt weniger Buttons pro Sekunde betätigt wurden. Möglich ist auch, dass die höhere Bandbreite des 2-Button-Scannings gegenüber dem 1-Button-Scanning auf eine Kombination aus beiden Gründen rückführbar ist.

Das 2-Button-Scanning hat unter den barrierefreien Eingabemethoden, abgesehen von der 3x3-Matrix als Button-Konfiguration, die niedrigste Bandbreite bei den falsch übertragenen Bits beziehungsweise am wenigsten fälschlich betätigte Buttons, wie ein Blick auf das Diagramm in der Abbildung 6.7 zeigt. Die wenigen Fehleingaben sind darauf zurückzuführen, dass die Probanden die Fokussierung der Buttons selbst steuern konnten, es also seltener ein „Verpassen“ wie beim 1-Button-Scanning oder ein fälschliches Auswählen wie beim Face-Tracking gab. Die wenigen Fehleingaben, die dennoch stattfanden, resultieren vermutlich daraus, dass die Probanden manchmal sprichwörtlich über das Ziel hinausschossen, indem sie die Taste für die Fokussierung des nächsten Buttons zu oft drückten und anschließend auch noch den Button betätigten. Allerdings sind die Fehleingaben beim 2-Button-Scanning sehr gering, weshalb es schwierig ist, eine oder mehrere Ursachen zu finden. Insgesamt machten die 16 Probanden 6 Fehleingaben. Dem gegenüber stehen 218 richtige. Deshalb ist es auch schwierig, eine Erklärung dafür zu finden, weshalb die in Abbildung 6.6 dargestellte Bandbreite an falsch übertragenen Bits bei der 4x4- und 5x5-Matrix größer ist als bei den beiden anderen Button-Konfigurationen. Es kann schlichtweg Zufall sein.

Sprachsteuerung

Im Gegensatz zu den beiden Eingabemethode 1- und 2-Button-Scanning steigt die durchschnittliche Bandbreite der Sprachsteuerung zwischen der 3x3- und 6x6-Matrix erheblich an. Das Diagramm in Abbildung 6.5 veranschaulicht dies. Der konstante Anstieg der Bandbreite liegt an der kontinuierlichen Eingabegeschwindigkeit. Das Diagramm in Abbildung 6.7 veranschaulicht, dass die 16 Probanden im Durchschnitt ca. 0,2 Buttons je Sekunden betätigten und das bei allen 4 getesteten Button-Konfigurationen. Hieraus geht hervor, dass sich die im Rahmen der Tests durchgeführte Erhöhung der Button-Anzahl nicht negativ auf die Eingabegeschwindigkeit auswirken. Jedoch ist diese Entwicklung für eine beliebige Anzahl an Buttons ausgeschlossen, da der Platz auf der Benutzeroberfläche begrenzt ist und die Beschriftung des Buttons oder seine Nummer für den Benutzer lesbar bleiben müssen. Die Eingabegeschwindigkeit ist viel mehr durch die derzeitige Umsetzung der verwendeten Voice Access App limitiert, denn der Großteil des Zeitbedarfs für eine Eingabe war während der quantitativen Benutzbarkeitstests auf diese zurückzuführen. Dafür gab es folgende 3 Ursachen. Nachdem der Proband

eine Spracheingabe tätigte, musste zunächst gewartet werden, bis die Voice Access App die Spracherfassung beendet. Das ist erforderlich, da im Vorfeld nicht bekannt ist, wie lange die Eingabe des Benutzers ist, weshalb sie deren Ende erst nach einer Phase, in der dieser nicht redet, erkennen kann. Anschließend führt sie die Spracherkennung online durch, was ebenfalls Zeit in Anspruch nimmt (vgl. [Goo16i]). Wenn die Spracheingabe einem Button anschließend zuordenbar ist, betätigt sie ihn. Dies erfolgt animiert, was den dritten Grund für den Zeitbedarf darstellt.

Die im Diagramm von Abbildung 6.6 dargestellten durchschnittlich falsch übertragenen Bits pro Sekunde sind sehr gering. Das lässt sich darauf zurückführen, dass die Probanden zur Betätigung eines Buttons dessen entsprechende Nummer sagen mussten. Die Nummerierung der Buttons erfolgte durch die Voice Access App (vgl. [Goo16a]). Dem geringen Anteil an Fehleingaben nach, lassen sich die Zahlen bei der Spracheingabe sehr gut voneinander unterscheiden, was dazu führte, dass die Probanden nur selten einen falschen Button betätigten. Das Diagramm in Abbildung 6.7 zeigt eine geringe Zunahme der Fehleingaben mit einer steigenden Zahl von Buttons. Hieraus lässt sich die Überlegung ableiten, dass die Spracherkennung größerer Zahlen eventuell schwieriger ist. Bei einer 3x3-Matrix besaßen die Buttons die Nummern 4 bis einschließlich 12. Bei der 6x6-Matrix waren sie von 4 bis einschließlich 39 durchnummeriert.

Was die Diagramme zu der Sprachsteuerung nur indirekt berücksichtigen, aber erhebliche Probleme bereitete, waren falsch verstandene Spracheingaben, welche keinem der zur Auswahl stehenden Buttons zuordenbar waren. Diese traten besonders intensiv bei Probanden auf, welche einen Dialekt hatten. Das Diagramm in Abbildung 6.8 zeigt den maximalen, durchschnittlichen und minimalen je Probanden berechneten Durchschnitt an betätigten Buttons pro Sekunde je Button-Konfiguration. Daraus, dass der Abstand zwischen den minimalen und durchschnittlichen Werten deutlich größer als jener zwischen den durchschnittlichen und maximalen ist, lässt sich ableiten, dass es bei einigen Probanden überdurchschnittlich große Probleme bei der Spracherkennung gegeben haben muss. Diese sind in der Eingabegeschwindigkeit deshalb enthalten, weil die Probanden nicht verstandene Spracheingaben so oft wiederholen mussten, bis sie das waren, was zu einer geringeren Eingabegeschwindigkeit führte. Aus diesem Grund sind die falsch verstandenen Spracheingaben indirekt in den Ergebnissen enthalten.

15 von den 16 Personen testeten die Sprachsteuerung auf Deutsch und 1 auf Englisch. Die mittels Englisch erzielte Bandbreite an richtig übertragen Bits pro Sekunde liegt bei allen 4 getesteten Button-Konfigurationen über dem Durchschnitt. Jedoch erlaubt dies hinsichtlich der Frage, ob Deutsch oder Englisch besser oder gleich gut geeignet ist für die Sprachsteuerung mittels der App Voice Access, aufgrund der ungleichen Verteilung an Tests, keine Schlussfolgerung.

Face-Tracking

Das Diagramm in Abbildung 6.5 zeigt, dass bei der Eingabemethode Face-Tracking die Anzahl der durchschnittlich pro Sekunde richtig übertragenen Bits mit zunehmender Anzahl an

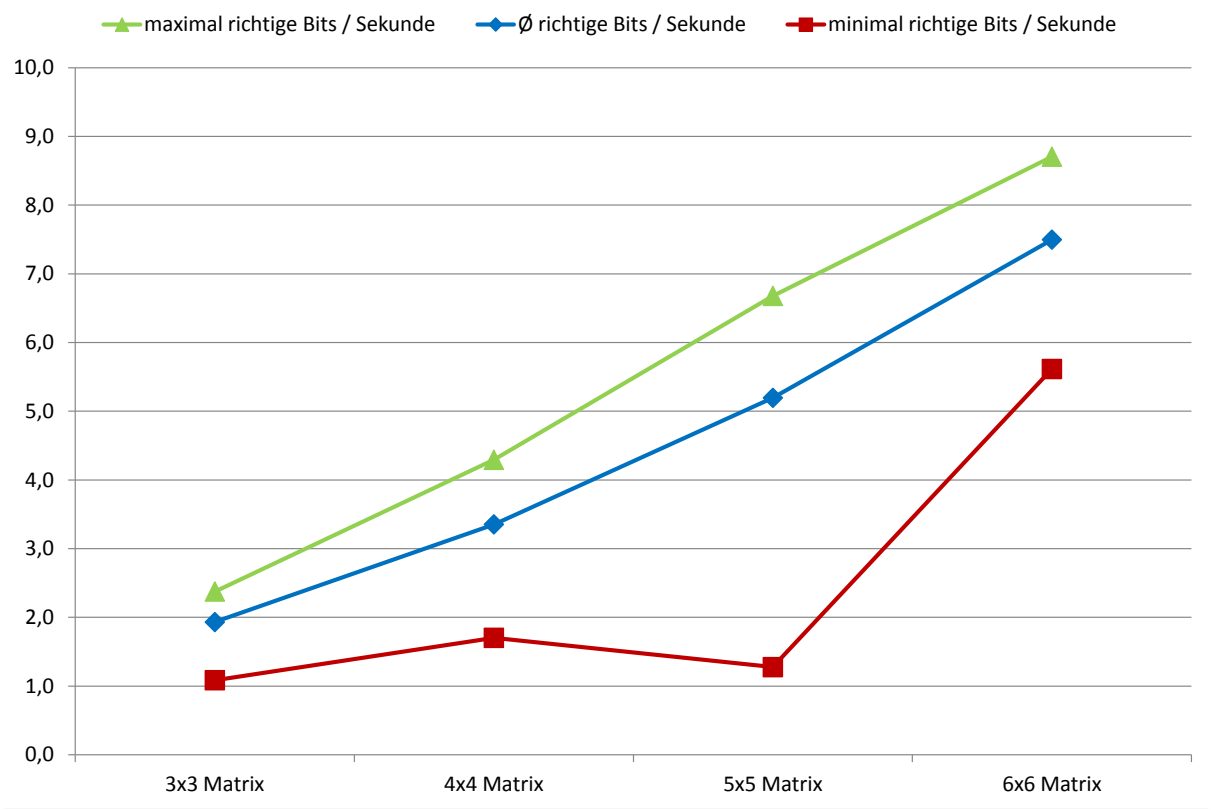

Abbildung 6.8.: Maximal, durchschnittlich sowie minimal übertragene Bits / Sekunde mit der Eingabemethode Sprachsteuerung.

getesteten Buttons gestiegen ist. Jedoch nimmt mit ihr auch deren Größe ab, was ihre Auswahl und Betätigung mittels Face-Tracking erschwert. Dies geht aus dem Diagramm in Abbildung 6.7 hervor. Es zeigt die, mit der Anzahl an getesteten Buttons sinkende, durchschnittliche Eingabegeschwindigkeit in Form von betätigten Buttons pro Sekunde.

Das Diagramm mit den im Durchschnitt übertragenen Bits pro Sekunde zeigt jedoch auch eine steigende Zahl an falsch übertragenen Bits in Abhängigkeit zu einer zunehmenden Anzahl an Buttons in den Tests. Die steigende Fehlerrate ist darauf zurückzuführen, dass je kleiner die Buttons sind, desto schwieriger deren Auswahl ist, da der Cursor, der hierfür notwendig ist und mittels Face-Tracking gesteuert wird, unruhig ist. Seine Unruhe ist jedoch nicht bei allen Probanden gleich. Eine weitere Schwierigkeit, die insbesondere bei kleinen Buttons zum Tragen kommt, ist vermutlich die, dass der Benutzer während des Blinzelns, welches den ausgewählten Button betätigt, den Cursor ruhig halten muss, um nicht versehentlich einen benachbarten Button auszuwählen und zu betätigten. Den Beobachtungen des Autors nach, fiel dies einigen Probanden schwer.

Im Vorfeld der quantitativen Benutzbarkeitstests kam die Überlegung auf, ob das Tragen einer Brille oder eines Bartes das Face-Tracking beeinflusst. Nachdem der Cursor je nach Proband

unterschiedlich unruhig war, könnte ein Bart oder eine Brille die Ursache dafür sein. Von den 16 Probanden hatten 5 einen Bart. Lediglich bei demjenigen mit dem ausgeprägtesten Bart kam es zu offensichtlichen Problemen. Diese äußerten sich darin, dass bei ihm der Cursor extrem unruhig war. Das Diagramm in Abbildung 6.9 lässt die Schlussfolgerung, dass ein Bart das Face-Tracking beeinflusst, nicht zu. Sie zeigen die im Durchschnitt richtig und falsch übertragenen Bits pro Sekunde der Probanden mit und jenen Probanden ohne Bart. Offensichtliche Abweichungen zwischen Bartträgern und bartlosen Testpersonen sind nicht erkennbar. Das Diagramm in Abbildung 6.10 unterscheidet zwischen Probanden mit sowie ohne Brille. Im Gegensatz zum Bart, lässt sich aus ihnen ein möglicher Einfluss durch das Tragen einer Brille auf das Face-Tracking ableiten. Sowohl die durchschnittlich falsch als auch die im Durchschnitt richtig übertragenen Bits pro Sekunde sind bei den Probanden ohne Brille etwas besser. Wichtiger als die Ergebnisse aus den Diagrammen ist jedoch die Erkenntnis, dass bei 2 von den 6 Brillenträgern das Face-Tracking mit aufgesetzter Brille überhaupt nicht nutzbar war, da das Blinzeln nur sehr eingeschränkt oder überhaupt nicht erkannt wurde. Deshalb haben diese 2 Probanden die quantitativen Benutzbarkeitstests schließlich ohne Brille durchgeführt. Das Diagramm mit den durchschnittlichen Bandbreitenwerten der Brillenträger beinhaltet demzufolge nur die Werte von den 4 Probanden, bei denen die Brille keine Probleme verursachte. Das lässt sich die These zu, dass offensichtliche nur manche Brillen negative Auswirkungen auf das Face-Tracking hatten und die anderen wenige bis keine.

Das Diagramm in Abbildung 6.11 zeigt die durchschnittliche Bandbreite an richtig und falsch übertragenen Bits je Sekunde mit einer höheren Anzahl an Buttons. Daneben befinden sich die Ergebnisse in Form von Zahlen in den Tabellen A.3 und A.4 des Anhangs. Die 7x7-, 8x8, 9x9- sowie 10x10-Matrixen wurden im Anschluss an das eigentliche Testprogramm in der Endphase der quantitativen Benutzbarkeitstests mit 3 beziehungsweise die 9x9-Matrix lediglich mit 2 Testteilnehmern getestet. Der Anlass hierfür war, dass es mit den ursprünglich getesteten Button-Konfigurationen bei der Eingabemethode Face-Tracking kontinuierlich zu einem Anstieg der Bandbreite kam und die quantitativen Benutzbarkeitstests ursprüngliche die Grenze des Möglichen von dieser ermitteln sollten, also jene Anzahl an Buttons, mit der die höchste Bandbreite erzielbar ist. Auch wenn die Durchführung mit 3 Probanden keine so aussagekräftigen Schlüsse erlaubt, lässt sich aus dem Diagramm ableiten, dass ungefähr bei der 8x8-Matrix beziehungsweise 64 Buttons die Grenze des Möglichen bei der Eingabemethode Face-Tracking in Kombination mit dem Prototyp erreicht ist. Bei dieser Konfiguration wurden durchschnittlich die meisten Bits pro Sekunde richtig übertragen. Bei der 9x9-Matrix sowie der 10x10-Matrix als Button-Konfiguration ist die Bandbreite hingegen wieder geringer. Bei Letzter übersteigt jene der im Durchschnitt falsch übertragenen Bits diese sogar.

Betrachtet man das Diagramm in Abbildung 6.6 zeigt sich unabhängig von den im vorherigen Abschnitt behandelten erweiterten quantitativen Benutzbarkeitstests, dass das Face-Tracking besonders bei den getesteten Konfigurationen mit den höheren Button Anzahlen die mit Abstand meisten Fehleingaben respektive falsch übertragenen Bits pro Sekunde durchschnittlich hatte. Mit dieser Erkenntnis sind die Durchschnittswerte bei den richtig übertragenen Bits pro Sekunde immer in Kombination mit den Fehleingaben zu betrachten. Denn in der Praxis wird es oftmals so sein, dass jede Fehleingabe mindestens eine richtige Eingabe zur Korrektor erfor-

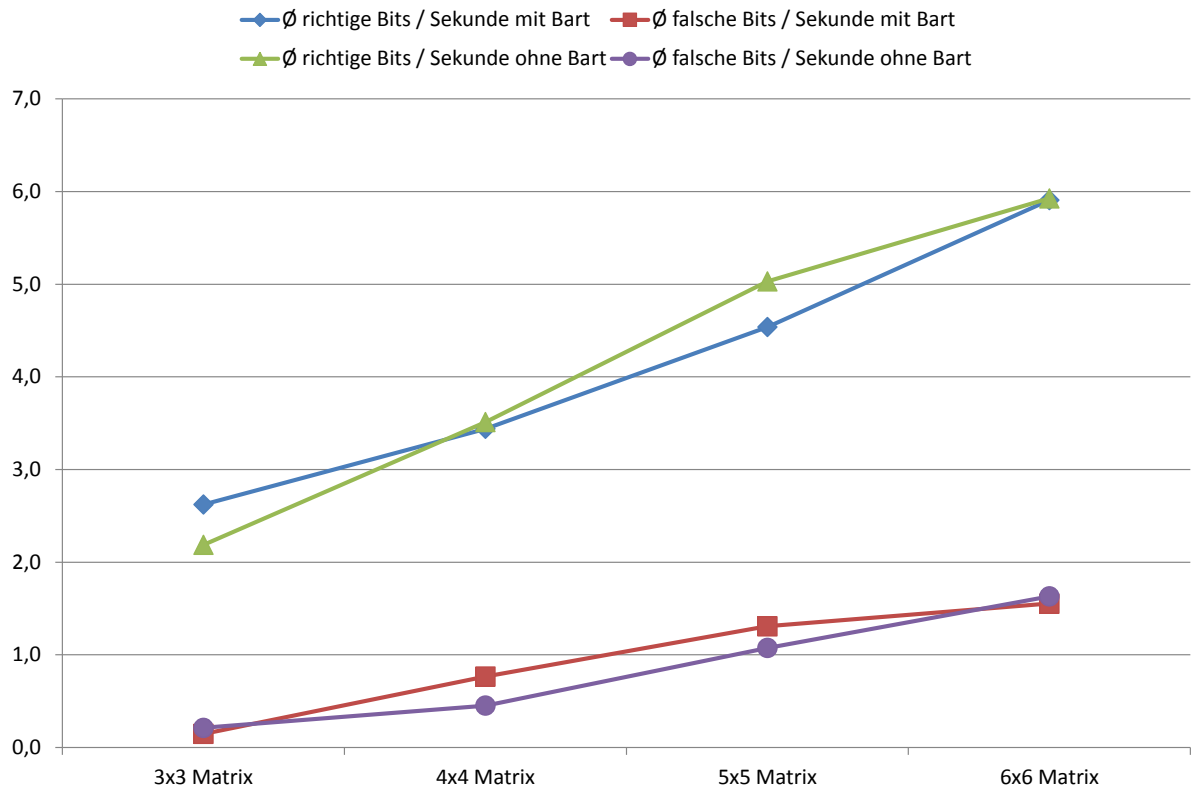

Abbildung 6.9.: Im Durchschnitt richtig sowie falsch übertragenen Bits / Sekunde in Abhängigkeit mit sowie ohne Bart.

dert. Unter diesem Gesichtspunkt betrachtet, wird anhand des Diagramms in Abbildung 6.11 schnell deutlich, dass die Button-Konfiguration mit der durchschnittlich höchsten Bandbreite an richtig übertragenen Bits pro Sekunde in der Praxis nicht unbedingte die leistungsfähigste ist. Dies gilt natürlich für sämtliche getestete Eingabemethoden. Jedoch kommt dieser Aspekt insbesondere beim Face-Tracking und 1-Button-Scanning, die im Vergleich zu den übrigen einen sehr hohen Anteil an Fehleingaben besitzen, eine größere Bedeutung zu.

6.2. Qualitative Benutzbarkeitstests

Dieses Unterkapitel beschreibt die Vorbereitung und Durchführung der qualitativen Benutzbarkeitstests mit Probanden aus der Zielgruppe des barrierefreien Smarthomes sowie deren anschließende Auswertung.

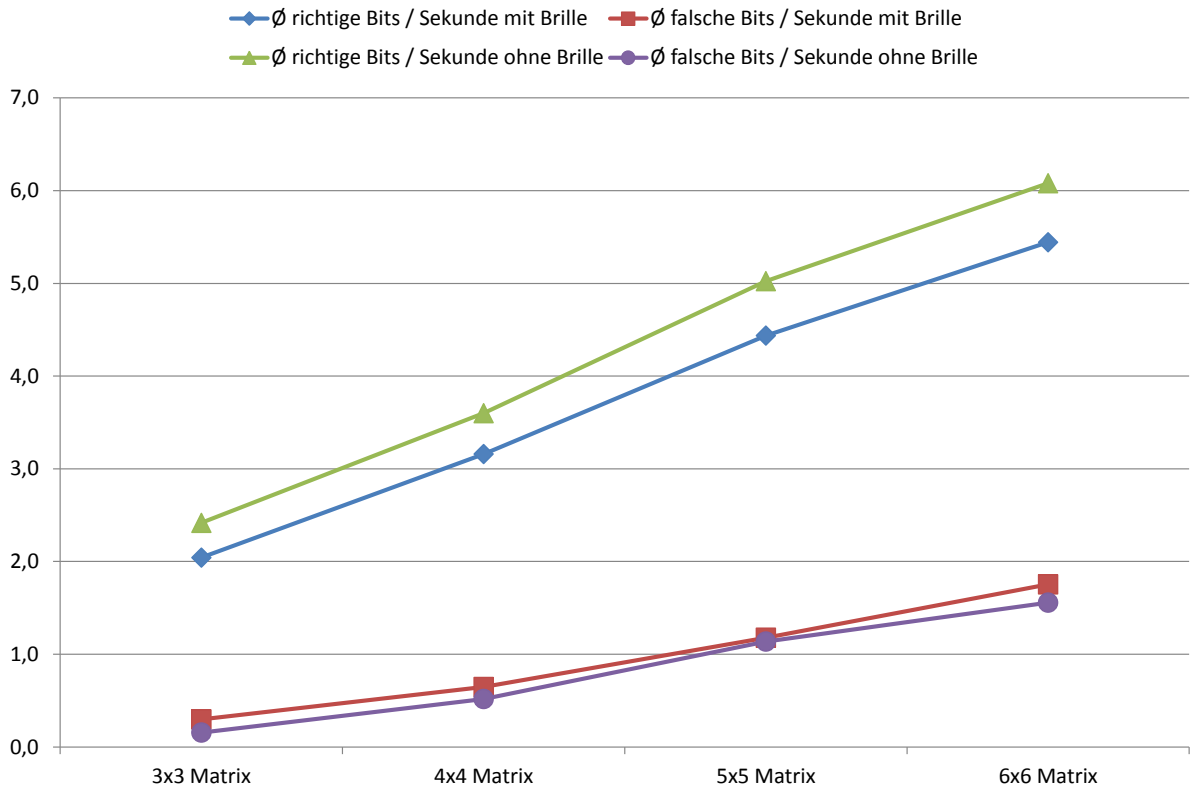

Abbildung 6.10.: Im Durchschnitt richtig sowie falsch übertragenen Bits / Sekunde mit sowie ohne Brille.

6.2.1. Vorbereitung

Die Vorbereitung der qualitativen Benutzbarkeitstests umfasste die Instrumentalisierung des Prototyps, um diese durchführen zu können. Hierzu zählte nachfolgend auch die Formulierung geeigneter Testaufgaben. Des Weiteren fielen die Probandenauswahl sowie die Erstellung eines Bewertungsbogen für deren qualitative Rückmeldungen in die Vorbereitungsphase. Diese 4 Aspekte werden im folgenden dargestellt.

Instrumentalisierung Prototyp

Die Abbildung 6.1 zeigt einen Screenshot von den Einstellungen für die qualitativen Benutzbarkeitstests. Sind diese eingeschaltet, ändert sich im Gegensatz zu den in Abschnitt 6.1 beschriebenen quantitativen Benutzbarkeitstests nicht die Benutzeroberfläche. Allerdings befindet sich in dieser dann zusätzlich ein Textfeld am oberen Displayrand, welches dem Benutzer die nächste Aufgabe im Rahmen des Benutzbarkeitstests anzeigt. Unter einer solchen ist zu verstehen, dass der Proband beispielsweise den TV-Sender ZDF auswählen soll.

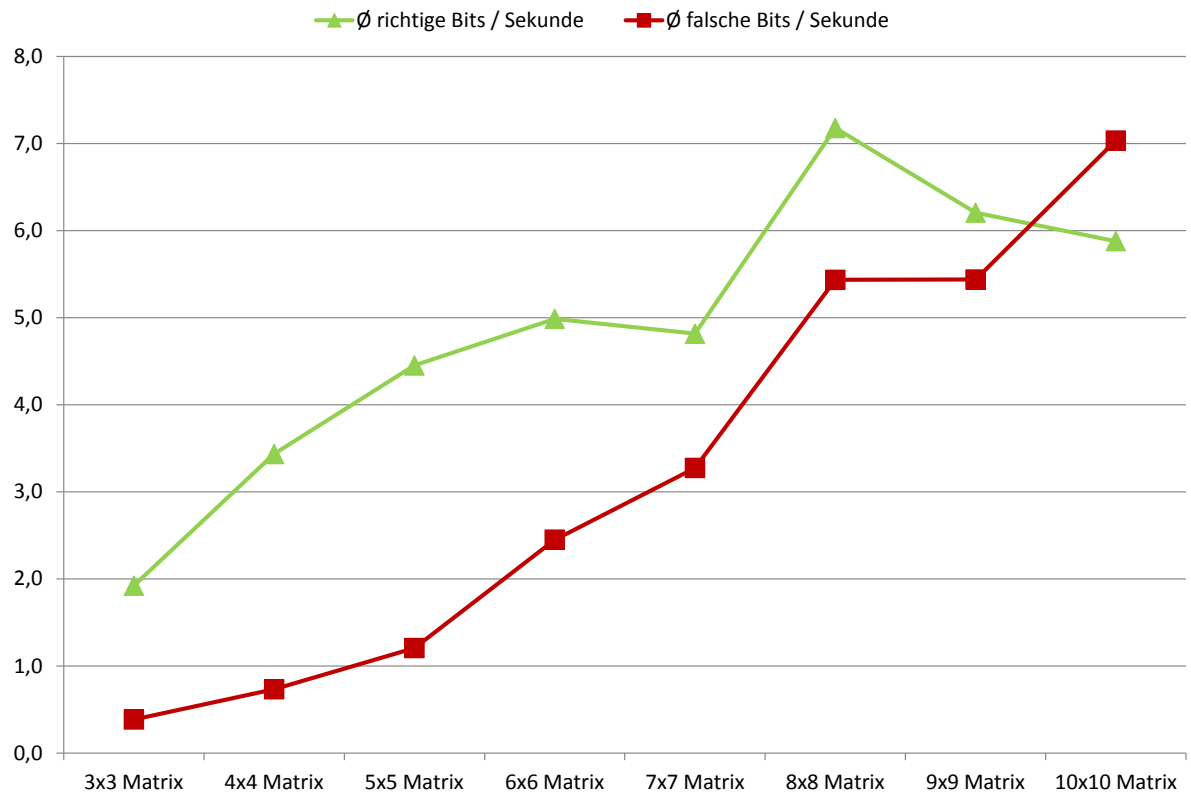

Abbildung 6.11.: Durchschnittlich richtig und falsch übertragene Bits / Sekunde bei den 3 Probanden die das Face-Tracking mit bis zu 100 Buttons testeten.

In den in Abbildung 6.1 zu sehenden Einstellungen lässt sich der Pfad zu einer XML-Datei festlegen. Aus ihr werden die Aufgaben für die Benutzbarkeitstests geladen. Im Quellcodeausschnitt 6.3 ist der Teil einer solchen XML-Datei zu sehen.

```
1  <test_set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
2     <test id="1" name="Test">
3        <test_case>
4           <description>TV-Sender Pro7 in der Senderliste auswählen</description>
5           <destination>/tv/tv_channelList/tv_pro7</destination>
6        </test_case>
7        <test_case>
8           <description>Zum nächsten Kanal / TV-Sender wechseln</description>
9           <destination>/tv/tv_channelUp</destination>
10       </test_case>
11       ...
12    </test>
13 </test_set>
```

Quellcodeausschnitt 6.3: Auszug aus einer XML-Datei mit den Testaufgaben.

Sie besteht aus mindestens einem Test. In der XML-Datei handelt es sich bei diesem um das Element `test`. Wenn die XML-Datei mehrere Tests beinhaltet, wird einer zufällig ausgewählt. Jeder Test besteht aus mindestens einer Aufgabe, repräsentiert durch das Element `test_case` in der XML-Datei. Jede Aufgabe besteht aus zwei Attributen. Bei diesen handelt es sich zum einen um die Beschreibung der Aufgabe und zum anderen um den Pfad zu dem Button, den der Proband betätigen soll. Bei der Aufgabenbeschreibung handelt es sich um den Text, der in dem Textfeld angezeigt wird. Der Pfad setzt sich aus den IDs der Buttons zusammen, welche der Proband ausgehend vom Hauptmenü betätigen muss, um das Ziel innerhalb der Menüstruktur zu erreichen beziehungsweise die Aufgabe zu erledigen.

Die Überprüfung, ob die Aufgabe erledigt ist, findet in der Klasse `UsabilityTestManager` statt. Sie ruft während der qualitativen Benutzbarkeitstests die in 5.4.2 beschriebene Klasse `CommandExecutor` auf, wenn sie einen Befehl ausführt und überreicht ihr diesen als Parameter. Da in diesem enthalten ist, welcher Button vom Proband betätigt wurde, kann die Klasse `UsabilityTestManager` zum einen überprüfen, ob der Pfad von der Wurzel zum Button mit jenem in der Aufgabe übereinstimmt und zum anderen die Benutzereingabe protokollieren. Wenn der Proband das Ziel erreicht hat, also die beiden Pfade übereinstimmen, wählt die Klasse `UsabilityTestManager` die nächste Aufgabe aus und macht deren Beschreibung im dafür vorgesehenen Textfeld für den Benutzer sichtbar oder zeigt im Fall, dass der Test zu Ende ist, dies an. Daneben ist die Klasse `UsabilityTestManager` für die Protokollierung der Benutzereingaben verantwortlich. Bei jeder Ausführung eines Befehls schreibt sie folgende Daten in eine CSV-Datei:

- Die Zeit in Millisekunden, die seit dem Start des Tests, nicht der einzelnen Aufgabe, bis zum Zeitpunkt der protokollierten Eingabe vergangen ist.
- Die Art des Befehls, also ob der Proband in der Menü-Struktur nach oben, unten, links oder rechts navigiert hat.
- Der Pfad von der Wurzel der Menü-Struktur zum Button, welchen der Proband betätigt hat. Dadurch ist es möglich, im Nachhinein bei der Auswertung des Tests festzustellen, ob der Proband den kürzest möglichen Weg zum Ziel durch die Menü-Struktur genommen hat oder Fehleingaben stattgefunden haben und wie er diese korrigiert hat.

Probandenauswahl

Die Durchführung der qualitativen Benutzbarkeitstests fand mit den Interviewpartnern aus Abschnitt 3.1 statt. Dadurch war keine erneute Suche nach geeigneten Personen aus der Zielgruppe erforderlich. Des Weiteren passten sie aufgrund ihrer Einschränkungen sehr gut in die Gruppe der möglichen zukünftigen Benutzer des barrierefreien Smarthomes. Letzterer Aspekt ist nach Kuniavsky ein wichtiges Kriterium für die Auswahl geeigneter Probanden (vgl. [KGM12, S. 265]). Selbiger weißt jedoch auch darauf hin, dass bei der Auswahl zusätzlich zu beachten ist, dass die Probanden nicht nur in die Zielgruppe passen, sondern auch die Art von Rückmeldungen zu dem getesteten Produkt geben können, welche die Organisatoren der

Benutzerstudie benötigen (vgl. [KGM12, S. 265]). Aus diesem Grund wurden die qualitativen im Gegensatz zu den quantitativen Benutzbarkeitstests nicht mit unversehrten Probanden durchgeführt, sondern mit solchen, welche Einschränkungen bei der Benutzung ihrer Hände haben und somit in die Zielgruppe des barrierefreien Smarthomes fallen. Der Grund hierfür ist, dass die qualitativen Benutzbarkeitstests Erkenntnisse zu der Benutzeroberfläche, der Navigation innerhalb des Menüs sowie der Eignung der barrierefreien Eingabemethoden Face-Tracking, Sprachsteuerung und 1- sowie 2-Button-Scanning bringen sollten. Dies war mit Probanden aus der Zielgruppe am besten möglich, zumal die Anwendung zur Steuerung des barrierefreien Smarthomes auf ihre Bedürfnisse ausgelegt sein soll. Dagegen zielten die quantitativen Benutzbarkeitstests auf einen Vergleich der Eingabemethoden hinsichtlich der mittels ihnen maximal übertragbaren Entscheidungen pro Zeiteinheit ab. Durch Probanden mit Einschränkungen, wäre es unklar gewesen, ob diese oder Charakteristika der Eingabemethode der limitierende Faktor sind, weshalb für diese Art von Benutzbarkeitstests unversehrte Probanden die geeigneteren waren.

Die Anzahl der Teilnehmer an den Interviews stimmte zudem mit den von Kuniavsky empfohlenen 5 Probanden für einen Benutzbarkeitstest überein (vgl. [KGM12, S. 267]). Laut ihm stellen 5 Probanden den besten Kompromiss zwischen den Ergebnissen, dem Aufwand und den Kosten dar (vgl. [KGM12, S. 267]). Nichtsdestotrotz empfiehlt er, für jeden Benutzbarkeitstest 6 bis 10 Probanden zu suchen (vgl. [KGM12, S. 267]). Dadurch entsteht natürlich eine Ausfallsicherheit, welche aus der Sicht des Autors dieser Arbeit jedoch nicht im Verhältnis zum Aufwand stand, da die Suche nach Probanden, welche in die Zielgruppe passen, sehr schwierig ist. Des Weiteren wusste er von den Interviews, dass sich die Teilnehmer auf diese gefreut hatten, da es eine Abwechslung in ihrem Alltag darstellt, weshalb es moralisch betrachtet schwierig gewesen wäre, mehr Probanden zu suchen, als letztendlich dann wirklich an den qualitativen Benutzbarkeitstests teilnehmen.

Testaufgaben

Benutzbarkeitstests entsprechen im Endeffekt Interviews, in deren Rahmen der Teilnehmer eine Abfolge von Aufgaben erledigen soll (vgl. [KGM12, S. 259]). Zur Definition dieser Aufgaben bietet es sich an, zunächst die zu evaluierenden Funktionen des Prototyps zu bestimmen (vgl. [KGM12, S. 268]). Die qualitativen Benutzbarkeitstests sollten folgende Funktionalitäten der Anwendung zur Steuerung des barrierefreien Smarthomes testen:

- Eingabemethode Face-Tracking
- Eingabemethode Sprachsteuerung
- Eingabemethoden 1- und 2-Button-Scanning
- Verständlichkeit der Benutzeroberfläche
- Navigation in der Menüstruktur

- wenn möglich auch die Eingabemethoden Touch, Maus und Tastatur

Die 3 beziehungsweise 4 Eingabemethoden gegeneinander zu testen, stellt einen erhöhten Aufwand dar, bietet aber die Möglichkeit, Stärken und Schwächen von diesen besser zu ermitteln (vgl. [KGM12, S. 270]]).

Als nächstes sind die einzelnen Aufgaben zu definieren. Sie sollten jeweils ein realistisches Benutzungsszenario darstellen und sich auf eine oder eine Gruppe der zuvor zum Testen festgelegten Funktionalitäten konzentrierten (vgl. [KGM12, S. 270]). Die im Rahmen dieser Arbeit definierten Aufgaben sollen jeweils mit den Eingabemethoden getestet werden, die der Proband benutzen kann. Bei der Anzahl der Buttons ist es geplant, mit einer 3x3-Matrix zu starten und sie entsprechend den Fähigkeiten des jeweiligen Probanden zu steigern. Jedoch ist es genauso möglich, die Tests mit weniger als 9 Buttons durchzuführen, wenn die Fähigkeiten beziehungsweise Einschränkungen des Probanden dies erfordern. Die nachfolgende Auflistung beinhaltet die Aufgaben, welche die Probanden der Reihe nach lösen sollten. Der Ausgangspunkt für die erste Aufgabe ist das Hauptmenü.

- Auswahl des Fernsehsenders ARD im Untermenü Favoriten.
- Vollbild im Untermenü TV aktivieren.
- Auswahl des Radiosenders SWR 3 im Untermenü Senderliste.
- Auswahl des Fernsehsenders Pro7 im Untermenü Senderliste.
- Auswahl des nächsten Fernsehsenders im Untermenü TV.

Dadurch, dass die fünf Aufgaben mit jeder für den Probanden bedienbaren Eingabemethode getestet wurden, stellten sie diesbezüglich eine breite Testabdeckung dar. Des Weiteren testeten sie auch die Verständlichkeit der Benutzeroberfläche und Menü-Struktur. Bei ersterer dadurch, dass sowohl Sender auszuwählen als auch Funktionalitäten, wie den Wechseln in den Vollbildmodus zu betätigen waren und die entsprechenden Buttons sich in verschiedenen Untermenüs befanden. Darüber hinaus ist es mit der aus 5 Aufgaben bestehenden Abfolge auch möglich, die Navigation in der Menüstruktur zu testen. Durch den Wechsel zwischen Fernseh- und Radiosendern muss der Proband in der Menüstruktur mehrmals vom Hauptmenü bis zum tiefsten Untermenü und zurück navigieren. Zudem befindet sich der Radiosender SWR 3 im entsprechenden Untermenü sehr weit „hinten“, wodurch auch die horizontale Navigation innerhalb eines solchen getestet wird.

Die Anzahl an Aufgaben ist bewusst gering gehalten. Auch wird vermutlich kein Proband in der Lage sein, alle sieben Eingabemethoden zu testen. Beides soll sicherstellen, dass die Tests nicht zu lange dauern. Zwar würde eine höhere Anzahl an Aufgaben kein Problem darstellen, was die von Kuniavsky empfohlene Testdauer von ein bis zwei Stunden betrifft, jedoch werden aufgrund ihrer Einschränkungen nicht alle Probanden in der Lage sein, den Prototyp in diesem zeitlichen Umfang zu testen (vgl. [KGM12, S. 268]). Sofern die Probanden im Anschluss an die qualitativen Benutzbarkeitstests, nachdem alle für sie möglichen Eingabemethoden und Button-Konfigurationen getestet wurden, noch in der Lage und bereit zu weiteren Tests sind,

war es geplant, die in Abschnitt 6.1 beschriebenen quantitativen Benutzbarkeitstests in einem deutlich reduzierteren Umfang durchzuführen.

Feedback

Im Vergleich zu den in Kapitel 3.1 behandelten Interviews, war der Fragebogen klein. Er setzte sich primär aus zwei Teilen zusammen. Der erste Teil bestand aus einer Bewertung der getesteten Eingabemethoden. Diese sollten die Probanden anhand einer Likert-Skala vornehmen. Hierzu konnten sie zwischen folgenden Likert-Items wählen:

- geeignet
- eher geeignet
- teils-teils geeignet
- eher ungeeignet
- ungeeignet

Die Bewertung fand unmittelbar nach dem Test der jeweiligen Eingabemethode statt, da zu diesem Zeitpunkt die Erfahrungen am präsentesten waren. Gleichzeitig sollte dieses Verfahren auch einen Vergleich der Eingabemethoden ermöglichen, weshalb die Probanden ihre Bewertungen, nachdem sie alle Eingabemethoden getestet und sie somit für sich persönlich vergleichen konnten, noch einmal ändern durften. Die Likert-Skala sollte die Eingabemethoden einfacher vergleichbar machen, da sie trotz der qualitativen Benutzbarkeitstests quantitative Daten liefert.

Der zweite Teil des Fragebogens enthielt qualitative Fragen. Diese bezogen sich auf die Vor- und Nachteile der Eingabemethoden sowie die Meinung der Probanden bezüglich dieser und der Benutzeroberfläche. Des Weiteren versuchte er auch Verbesserungsvorschläge im Bezug auf den Prototypen zu erlangen.

6.2.2. Durchführung

Die Durchführung der qualitativen Benutzbarkeitstests fand auf zwei Tage verteilt statt. Von den ursprünglich geplanten 5 Teilnehmern wurde leider einer kurzfristig krank, wodurch es nur 4 gab. Am Anfang erhielten die Teilnehmer jeweils eine Einführung in die Benutzeroberfläche. Hierzu bekamen sie unter anderem auch eine Übersicht der Menüstruktur in Papierform gezeigt. Des Weiteren konnten sie sich mit allen, von ihnen benutzbaren Eingabemethoden, vertraut machen. Dieses Vorgehen sollte sicherstellen, dass während den Tests möglichst keine Verständnisprobleme auftreten.

Die Einstellungen für das Face-Tracking sind gegenüber den quantitativen Benutzbarkeitstests gleich geblieben. Die Geschwindigkeit für die automatische Fokussierung des nächsten Buttons beim 1-Button-Scanning ist mit den Probanden jeweils im Vorfeld an deren Fähigkeiten angepasst worden. Sämtliche Tests wurden mit einer 3x3-Matrix als Button-Konfiguration durchgeführt. Zwar hätten sie die Tests auch mit einer anderen Button-Konfiguration durchführen dürfen, jedoch wollte dies keiner. Mehrere Button-Konfigurationen zu testen, war nicht möglich, da die einführenden Erklärungen, das Ausprobieren sowie die Tests für die Probanden schon mit einer anstrengend waren. Dies war insbesondere bei den Testpersonen der Fall, die wenig technikaffin sind und für die somit schon die Nutzung eines Tablets an sich, eine völlig neue Erfahrung darstellte.

Während der Benutzbarkeitstests hat sich der Autor im Gespräch mit den Testpersonen im Hinblick auf den Prototyp neutral verhalten, wie von Kuniavsky empfohlen (vgl. [KGM12, S. 288]). Dieser rät darüber hinaus, mit den Probanden während der Tests einen Dialog zu führen, indem der Moderator des Tests sie im Vorfeld einer Eingabe fragt, was sie in Folge von dieser erwarten und warum sie sich für das jeweilige Vorgehen entschieden haben (vgl. [KGM12, S. 288]]). Obwohl der Autor dieser Arbeit überzeugt ist, dass diese Fragen hilfreich sind, hat er auf sie während den Tests verzichtet um die Teilnehmer nicht zu überfordern, da er bei den meisten dein Eindruck hatte, dass die Testaufgaben an sich schon ihre volle Aufmerksamkeit abverlangten. Stattdessen wurden die Probanden immer im Anschluss an einen Test bezüglich ihrer Meinung zu der zuvor getesteten Eingabemethode befragt. Wenn sie während des Tests schon von sich aus Dinge anmerkten, wurden diese natürlich erfasst. Kuniavsky findet es zudem hilfreich, den Probanden in manchen Situationen beim Lösen einer Testaufgabe zu unterstützen (vgl. [KGM12, S. 288-289]). Eine solche Unterstützung war je nach Proband teilweise intensiv erforderlich.

Quantitative Benutzbarkeitstests, wie in Unterkapitel 6.1 beschrieben, durchzuführen um zu ermitteln, wie sich die diversen Einschränkungen der Testteilnehmer auf die Bandbreiten der unterschiedlichen Eingabemethoden und deren Verhältnis untereinander auswirken, war mit keiner einzigen Testperson möglich. Der Grund hierfür war in der Regel Ermüdung in Folge der qualitativen Benutzbarkeitstests und bei einem die Tatsache, dass ihrer Meinung nach keine der verfügbaren Eingabemethoden für sie geeignet ist.

6.2.3. Auswertung

Die 4 an den qualitativen Benutzbarkeitstests teilnehmenden Personen waren im Durchschnitt 50,75 ±13,55, die jüngste 32 und die älteste 70 Jahre alt. Zudem handelte es sich bei ihnen jeweils zur Hälfte um Frauen sowie Männer. Im Folgenden geht dieses Unterkapitel zunächst auf die Eingabemethoden und anschließend auf die Benutzeroberfläche ein.

Die Tabelle 6.1 beinhaltet die Bewertungen sämtlicher getesteter Eingabemethoden anhand der in Abschnitt 6.2.1 erstellten Likert-Items. Die Eingabemethoden, welche in der Tabelle 6.1 nicht enthalten sind, konnte keiner der Probanden testen. Ursächlich dafür war entweder, dass

Tabelle 6.1.: Bewertung der Eingabemethoden (1 = geeignet, 2 = eher geeignet, 3 = teils-teils geeignet, 4 = eher ungeeignet, 5 = ungeeignet).

	Testperson				
Eingabemethode	A	B	C	D	Durchschnitt
Touch	2	3	-	-	teils-teils geeignet (2,5)
Face-Tracking	5	1	5	3	eher ungeeignet (3,5)
1-Button-Scanning	2	3	-	-	teils-teils geeignet (2,5)
2-Button-Scanning	2	-	-	-	eher geeignet (2,0)
Sprachsteuerung	5	4	5	3	eher ungeeignet (4,25)

die Bedienung aufgrund der Einschränkungen nicht möglich war oder die Zeit beziehungsweise Konzentrationsfähigkeit der Probanden für weitere Tests nicht mehr ausreichte. Die Eingabemethoden Face-Tracking sowie Sprachsteuerung hat jeder Proband ausprobiert. Zahlenmäßig betrachtet, erzielte das Face-Tracking im Durchschnitt eine etwas bessere Bewertung als die Sprachsteuerung. Gerundet betrachtet haben beide entsprechend der Likert-Skala ein „eher ungeeignet“ erhalten. Bei den anderen getesteten Eingabemethoden ist zwar ebenfalls eine durchschnittliche Bewertung angegeben, jedoch ist diese wenig aussagekräftig, da sie lediglich aus maximal zwei Werten resultiert. Interessant sind insbesondere die Bewertungen der Probanden A und B, da sie viele Eingabemethoden testen konnten. Dadurch haben beide eine oder mehrere für ihre Situation geeignete Eingabemethode(n) gefunden, was sich darin widerspiegelt, dass beide mindestens eine mit „geeignet oder eher geeignet“ bewertet haben. Aus der Tabelle 6.1 geht zudem hervor, dass das Face-Tracking sowie die Sprachsteuerung die Eingabemethoden sind, die von Benutzern mit einer eingeschränkten Mobilität ihrer Hände am ehesten nutzbar sind. Dass die Teilnehmer der qualitativen Benutzbarkeitstests das Face-Tracking von „geeignet“ bis „ungeeignet“ bewerten haben, lässt den Schluss zu, dass für dieses zwar Geschick und Übung erforderlich ist, es aber trotz der im Durchschnitt schlechten Bewertung nicht grundsätzlich ungeeignet ist. Auf die Sprachsteuerung trifft dies nicht zu. Mit ihr kam keiner der Probanden zufriedenstellend zurecht.

Die Tabelle 6.2 zeigt die Ergebnisse von der Durchführung der in Abschnitt 6.2.1 beschriebenen Testaufgaben. Die Testdauer ist in Sekunden angeben und umfasst die Zeit, welche die Testperson zur Durchführung der Testaufgaben mit der jeweiligen Eingabemethode benötigte. Die Fehleingaben sind die Differenz zwischen der Anzahl der betätigten Buttons und jenen 21 Button-Betätigungen, die zur Abarbeitung der Testaufgaben mindestens erforderlich waren. Sprich, nicht jede Fehleingabe war selbst eine, jedoch die Folge einer solchen, da zur Korrektur betätigte Buttons ebenfalls als Fehleingabe gezählt wurden. Ein Vergleich der Eingabemethoden ist mit den Testergebnissen aus Tabelle 6.2 nur schlecht möglich, da die wenigsten Probanden es schafften, eine größere Anzahl an Eingabemethoden mit ihnen zu testen. Die Abweichung zwischen durchgeführten Tests und bewerteten Eingabemethoden rührt daher, dass die Probanden zu Beginn der Benutzbarkeitstests sich mit diesen zunächst

Tabelle 6.2.: Testdauer in Sekunden sowie die Fehleranzahl der Probanden bei der Lösung der Testaufgaben, je Eingabemethode.

	Testperson									
	A		B		C		D		Durchschnitt	
Eingabemethode	Dauer	Fehler	Dauer	Fehler	Dauer	Fehler	Dauer	Fehler	Dauer	Fehler
Touch	294	24	-	-	-	-	-	-	249	24
Face-Tracking	-	-	502	43	367	5	364	15	411	21
1-Button-Scanning	318	22	-	-	-	-	-	-	318	22
2-Button-Scanning	405	30	-	-	-	-	-	-	405	30
Sprachsteuerung	-	-	-	-	-	-	330	0	330	0

vertraut machen durften, wodurch sie sich zu ihnen eine Meinung bilden konnten. Die Aufgaben konnten sie jedoch aufgrund mangelnder Konzentrationsfähigkeit und Zeit nicht mehr mit allen Eingabemethoden testen. Interessant an den Ergebnissen in Tabelle 6.2 ist dennoch, dass Proband D bei der Spracheingabe 0 und beim Face-Tracking 15 Fehleingaben hatte, was wie schon die Ergebnisse der quantitativen Benutzbarkeitstests in Abschnitt 6.1.3 zeigt, dass bei der Sprachsteuerung selten Fehleingaben auftreten. Auffällig ist auch, dass Testperson C die Eingabemethode Face-Tracking mit „ungeeignet“ bewertet hat, gleichzeitig beim Lösen der Testaufgabe nur 5 Fehleingabe machte, was eigentlich zur Schlussfolgerung führen würde, dass er mit dieser Eingabemethode zurechtkam. Der Grund dafür ist vermutlich der, dass die Testperson C an ihrem Computer schon einen Face-Tracker mit Marker und hochwertiger Kamera nutzt, welche eine präzisere Bedienung ermöglicht als das Face-Tracking der App zur Steuerung des barrierefreien Smarthomes.

Die qualitativen Benutzbarkeitstests sollten jedoch nicht nur quantitative Ergebnisse, sondern auch qualitatives Feedback hervorfördern. Hierzu werden zum einen die von den Teilnehmern von sich aus getätigten Anmerkungen zu dem Prototyp, ihre Antworten auf die Frage nach den Vor- und Nachteilen der jeweiligen Eingabemethode sowie die Beobachtungen des Autors ausgewertet.

Die Testpersonen A und B probierten beide die Eingabemethode Touch aus. Für Erstere stellte diese aufgrund der großen Buttons keine Schwierigkeiten dar. Die Testperson B hatte diese hingegen, da sie die Bewegungen der zur Bedienung genutzten Hand nicht gut kontrollieren kann. Laut ihrer eigenen Einschätzung würde sie viel Übung und Glück benötigen, um die App zur Steuerung des barrierefreien Smarthomes mittels Touch erfolgreich bedienen zu können.

Die Eingabemethode Face-Tracking konnten drei der vier Testpersonen ausprobieren. Testperson A konnte das Face-Tracking nicht zufriedenstellend bedienen, da sie aufgrund ihrer Beeinträchtigungen die Bewegung ihres Kopfes nur schwer kontrollieren kann. Die Testpersonen C und D kritisierten an dem Face-Tracking, dass der Cursor zu unruhig ist und nicht so reagiert, wie sie es sich vorstellen. Allerdings ist anzumerken, dass die Bedingungen für das Face-Tracking bei der Testperson D erschwert waren, da diese die Benutzbarkeitstests nur liegend durchführen konnte. Für die Testperson B war das Face-Tracking hingegen die

beste der getesteten Eingabemethode. Kritik hatte sie keine an ihr anzumerken. Probleme, die Bewegungen des Kopfes zu kontrollieren, hatte sie nach eigenen Angaben, jedoch würde sie das Face-Tracking deshalb sogar gut finden, um eben genau diese durch Übung zu reduzieren.

Das 1-Button-Scanning konnte nicht mit einer Fortschrittsgeschwindigkeit von 300 Millisekunden, wie bei der Bandbreitenmessung in Unterkapitel 6.1 getestet werden, sondern mit 2 beziehungsweise 3 Sekunden. Testperson A kam damit sehr gut zurecht und meinte, dass ihr diese Eingabemethode nach einer Eingewöhnungszeit lieber wäre als die herkömmliche Fernbedienung für den Fernseher. Für Testperson B war die Bedienung der Taste des Button Switches schwierig. Der Grund hierfür war, dass sie die Hand nicht so gut steuern kann, wodurch für sie selbst eine Fortschrittsgeschwindigkeit von 3 Sekunden noch zu schnell war. Darüber hinaus betätigte sie aufgrund der motorischen Schwierigkeiten im Bereich der Hand den Taster häufig versehentlich mehrmals kurz hintereinander, was Fehleingaben zur Folge hatte. Besonders problematisch war, dass in einem neuen Untermenü als erstes jener Button fokussiert ist, der einen zurück in das übergeordnete Menü bringt, wodurch es mehrmals vorkam, dass die Testperson B den Button in ein Untermenü tätigte, aber durch die unkontrollierte Mehrfachbetätigung des Tasters sofort wieder in eine oder mehrere Menüebenen weiter oben gekommen ist.

Bei der Sprachsteuerung hatten alle 4 Testpersonen Probleme. Ursächlich hierfür war, dass sie nicht laut und deutlich genug sprechen konnten, weshalb die App Voice Access es oftmals gar nicht erkannt hat, wenn sie was sagten oder es falsch verstanden hat. Die Testpersonen A und B hatten darüber hinaus noch weitere Schwierigkeiten mit der Sprachsteuerung als Eingabemethode. Erstere gab an, dass es sie schnell verunsichert hat, wenn die App zur Steuerung des barrierefreien Smarthomes eine unbeabsichtigte Aktion ausführte in Folge eines falsch verstandenen Sprachbefehls. Bei der Testperson B war neben dem lauten sowie deutlichen Sprechen das Problem, dass sie nicht wie erforderlich mit der Sprachsteuerung interagieren konnte. Anstatt die Zahl oder die Beschriftung von einem der angezeigten Buttons zu sagen, redete sie dialogartig beziehungsweise in ganzen Sätzen, was die App Voice Access nicht verarbeiten konnte. Auch war es für die Testperson in den Augen des Autors nicht verständlich, dass nur die sichtbaren Buttons mittels Sprachsteuerung betätigbar sind und beispielsweise im Radiomenü nicht der Fernsehsender 3sat einstellbar ist. Die Testpersonen A und D machten zudem noch Verbesserungsvorschläge bezüglich der App Voice Access. Erstere würde es begrüßen, wenn der aus der Spracheingabe erkannte Text in der Benachrichtigungszeile durch die App Voice Access in einer größeren Schrift dargestellt wird. Von der Testperson D kam die Anregung, die akustische Rückmeldung im Anschluss an eine Spracheingabe, ob diese einem Button zuordenbar ist oder nicht, wegzulassen.

Die Tabelle 6.3 zeigt die Bewertungen der Testpersonen zum einen für Verständlichkeit sowie das Erscheinungsbild der Benutzeroberfläche und zum anderen das Navigationskonzept in der Menüstruktur. Im Vergleich zu den Eingabemethoden fallen diese größtenteils positiv und einheitlich aus. Keiner der Probanden bewertete schlechter als „teils-teils geeignet“. Testperson A begründete ihre positive Bewertung zudem ausführlich damit, dass sich die Größe der Buttons indirekt über deren Anzahl sowie die Schriftgröße ihrer Beschriftung individuell konfigurieren

Tabelle 6.3.: Bewertung der Benutzeroberfläche (1 = geeignet, 2 = eher geeignet, 3 = teils-teils geeignet, 4 = eher ungeeignet, 5 = ungeeignet).

	Testperson				
Eingabemethode	A	B	C	D	Durchschnitt
Verständlichkeit	1	3	2	1	eher geeignet (1,75)
Navigation	1	2	2	1	eher geeignet (1,5)

lässt. Des Weiteren findet sie es speziell für die Steuerung eines Fernsehers praktisch, dass das Menü auf dem mobilen Endgerät dargestellt ist, das näher bei ihr ist und nicht auf dem entfernteren Fernseher, wodurch es für sie besser lesbar ist. Auch findet Testperson A die Benutzeroberfläche insgesamt übersichtlicher im Vergleich zu jenen der Fernseher.

7. Zusammenfassung und Ausblick

Dieses Kapitel blickt zunächst mit einer Übersicht über den Realisierungsstatus der Anforderungen zurück. Im nächsten Schritt arbeitet es den Mehrwert der Umsetzung gegenüber den Bedienungshilfen von Android heraus. Im Anschluss daran fasst es die Ergebnisse der im vorherigen Kapitel behandelten Benutzbarkeitstests zusammen und stellt sie zudem jenen aus den verwandten Arbeiten aus Abschnitt 2.4 gegenüber. Zum Abschluss erfolgt noch ein Ausblick auf Erweiterungsmöglichkeiten der Anwendung zur Steuerung des barrierefreien Smarthomes.

7.1. Umgesetzte Anforderung

Die Tabelle 7.1 beinhaltet die in 3.2 definierten Anforderungen zusammen mit ihrem Realisierungsstatus. Das Symbol ✓ bedeutet, dass sie umgesetzt und das Symbol ✗, dass sie nicht realisiert wurden. Die Tabelle 7.1 unterscheidet zu dem, zwischen der Umsetzung des in Kapitel 4 beschriebenen Konzepts sowie dem Prototyp in Kapitel 5. Die vorherigen Kapitel haben die umgesetzten Anforderungen schon ausführlich behandelt. Deshalb soll dieser Abschnitt kurz zusammenfassen, was zu der Nichtrealisierung weniger Anforderungen führte. Das Konzept unterstützt die Android Bedienungshilfen (Anforderung F2), indem es unter anderem für jedes Menüelement auch eine aussprechbare Variante der Bezeichnung vorsieht, was die Verständlichkeit der Sprachwiedergabe mittels TalkBack fördert. Der Prototyp implementiert die dafür benötigten Methoden allerdings nicht, da blinde und sehbehinderte Personen nicht dem Kreis der Testpersonen bei den in Kapitel 6 beschrieben Benutzbarkeitstests angehörten. Aus diesem Grund unterstützt er die Android Bedienungshilfen nur teilweise. Die Anforderung F4 Sprachsteuerung ist ebenfalls nur konzeptionell berücksichtigt, da für diese in den Benutzbarkeitstest die App Voice Access von Google verwendet wurde. Werde prototypisch noch konzeptionell wurde die Anforderung F8 Ein- und Ausschalten, welche es ermöglichen sollte, die Steuerung des barrierefreien Smarthomes barrierefrei ein- und auszuschalten. Der Gründe dafür waren zum einen, wie schon bei Anforderung F2, dass die Umsetzung im Prototyp für die Benutzbarkeitstests nicht erforderlich war und zum anderen das Ein- und Ausschalten eines Smartphones / Tablets ohne die Benutzung einer Hand, nicht ohne weiteres möglich ist. Die Anforderung N2 Verzicht auf Hilfsmittelhardware, ist sowohl im Konzept als auch dem Prototyp nur deshalb teilweise umgesetzt, da für die beiden Eingabemethoden 1- und 2-Button-Scanning, Hilfsmittel-Hardware erforderlich ist, wenn diese mittels Taster von Personen aus der Zielgruppe genutzt wird.

Tabelle 7.1.: Übersicht der im Konzept und Prototyp realisierten (✓) sowie nicht realisierten (✗) Anforderungen.

	Anforderung	Konzept	Prototyp
F1	Unterschiedliche Eingabemethoden	✓	✓
F2	Unterstützung der Android Bedienungshilfen	✓	✓ / ✗
F3	Steuerung via Face-Tracking	✓	✓
F4	Sprachsteuerung	✓	✗
F5	Steuerung mittels Taster oder Tastatur	✓	✓
F6	Konfigurierbare Anzahl an Buttons	✓	✓
F7	Konfigurierbare Schriftgröße	✓	✓
F8	Ein- und Ausschalten	✗	✗
N1	Eignung für unterschiedliche Geräte	✓	✓
N2	Verzicht auf Hilfsmittelhardware	✓ / ✗	✓ / ✗
N3	Erweiterbares Menü	✓	✓
N4	Unterstützung von Erweiterungen	✓	✓
N5	Erweiterbarkeit der Eingabemöglichkeiten	✓	✓

7.2. Mehrwert gegenüber den Android Bedienungshilfen

Die App zur Steuerung des barrierefreien Smarthomes bietet gegenüber den in 2.3 behandelten Android Bedienungshilfen mehrere Vorteile.

In den Einstellungen der Benutzeroberfläche lässt sich die Anzahl der maximal gleichzeitig dargestellten Buttons festlegen. Dadurch ist indirekt auch deren Größe beeinflussbar. Große Buttons können die Bedienung mittels Touch für Benutzer mit motorischen Einschränkungen, beispielsweise in Folge einer Tetraplegie mit tiefer Verletzungshöhe, erleichtern. Des Weiteren ist die Schriftgröße für die Button-Beschriftung frei wählbar. Die Bedienungshilfe von Android ermöglichen zwar ebenfalls Einstellungen für die Schrift- und Anzeigengröße, jedoch fällt die maximale Vergrößerung kleiner aus (vgl. [Goo16k]). Ein weiterer Vorteil von der Benutzeroberfläche der App zur Steuerung des barrierefreien Smarthomes ist, dass sie einfach gehalten und zudem über die Definitionsdatei für die Menüstruktur auch an die kognitiven Fähigkeiten des Benutzers anpassbar ist.

Die Entwickelten Eingabemethoden 1- und 2-Button-Scanning ähneln jenem Scanning des Schalterzugriffs der Android Bedienungshilfen (vgl. [Goo16g]). Im Detail haben Erstere 2 jedoch entscheidende Vorteile. Das 1- und 2-Button-Scanning zusätzlich zu Tastern auch über Blinzeln möglich. Dadurch ist es für Benutzer zugänglich, welche gar keine motorischen Fähigkeiten in ihren Händen besitzen. Des Weiteren ist beim Blinzeln keine Hilfsmittel-Hardware erforderlich. Letzteres ist grundsätzlich auch beim Schalterzugriff von Android nicht nötig, da die Gerätetasten nutzbar sind (vgl. [Goo16j]). Jedoch handelt es sich hierbei lediglich um eine

Entwickleroption und es wäre für die Benutzer aus der Zielgruppe schwierig, die Gerätetasten, welche häufig klein sind, zu bedienen (vgl. [Goo16j]). Insbesondere im Hinblick auf Benutzer mit kognitiven Einschränkungen oder wenig Erfahrung in der Bedienung elektronischer gerät, hat das 1- und 2-Button-Scanning, dadurch, dass es Bestandteil der App zur Steuerung des barrierefreien Smarthomes ist, den Vorteil, dass sie diese nicht versehentlich verlassen können. Auch müssen sie dadurch zwischen weniger Optionen wählen, da nur die Buttons der App gescannt werden.

Dass keine Hilfsmittel-Hardware benötigt wird, ist auch ein Vorteil des Face-Trackings. Diese barrierefreie Eingabemethode gibt es in den Bedienungshilfen von Android noch gar nicht. Des Weiteren ist damit die App zur Steuerung des barrierefreien Smarthomes von Anwendern bedienbar, die gar keine motorische Fähigkeiten mehr in ihren Händen besitzen. Sie könnten die App alternativ nur über die Sprachsteuerung der Bedienungshilfen steuern oder müssten auf Hilfsmittel-Hardware zurückgreifen. In den qualitativen Benutzbarkeitstests in 6.2 zeigte sich aber, dass die Probanden erhebliche Probleme mit der Sprachsteuerung hatten, da ihre Eingaben oftmals gar nicht oder falsch verstanden wurden. Bei den quantitativen Benutzbarkeitstests in 6.1 mit unversehrten Probanden trat dies nicht so massiv auf, was womöglich darauf zurückzuführen ist, dass wie in Abschnitt 2.2 beschrieben, je nach Verletzungshöhe eine Querschnittlähmung auch die Atmung und dadurch indirekt die sprachlichen Fähigkeiten beeinträchtigt. Zusätzlich ist es möglich, dass die Sprache des Benutzers aufgrund einer Mehrfachbehinderung oder zusätzlicher Krankheit / Verletzung für die Steuerung damit zu eingeschränkt ist. Unabhängig von den körperlichen Voraussetzungen, kann zudem eine laute Umgebung die Sprachsteuerung negativ beeinträchtigen. Jene von Google in Form der App Voice Access hat darüber hinaus den Nachteil, dass sie eine Internetverbindung benötigt, welche jedoch nicht in jeder Situation verfügbar oder bezahlbar ist (vgl. [Goo16i]). Die entwickelte Eingabemethode Face-Tracking besitzt diese 2 Einschränkungen hingegen nicht.

7.3. Bewertung der Testergebnisse

In den in Abschnitt 6.1 durchgeführten quantitativen Benutzbarkeitstests erzielten die barrierefreien Eingabemethoden deutlich geringere Bandbreiten als die herkömmlichen. Das ist nicht außergewöhnlich. Die in Abschnitt 2.4 beschriebenen Studien zu barrierefreien Eingabemethoden haben vergleichbare Ergebniskonstellationen. Dies soll exemplarisch anhand der getesteten Eingabemethoden Maus und Face-Tracking erläutert werden. Williams et al. ermittelten in ihrer Studie mit einer Maus eine ungefähr 5 mal höhere Bandbreite, als mit ihren barrierefreien Eingabemethoden, die auf einem Headset mit Lagesensoren beziehungsweise der Elektromyografie basierten (vgl.[WK08]). Das Verhältnis von barrierefreien zu herkömmlichen Eingabemethoden ähnelt somit jenem in dieser Arbeit. Die Eingabemethode Maus erzielte in ihr ausgehend von der 3x3 Button-Matrix den 5,11-, 5,54-, 6,17- beziehungsweise 6,76-fachen Durchsatz an richtig übertragenen Bits im Vergleich zum Face-Tracking. Ebenso erreichte in

dem von Jeong et al. durchgeführten Benutzbarkeitstest die Maus gegenüber der barrierefreien Eingabemethode ungefähr die 7-fache Geschwindigkeit (vgl. [JKS05]).

Das Diagramm in Abbildung 6.6 veranschaulicht, dass mit der Eingabemethode Face-Tracking die meisten Fehleingaben passierten. Auch dabei handelt es sich nicht um ein Problem der Umsetzung in dieser Arbeit, denn in der Studie von Caltenco et al. bewerteten die Teilnehmer Gaze-Tracker als unzuverlässig, was die Vermutung zulässt, dass die hohe Fehlerrate ein allgemeiner Nachteil dieser Eingabemethode ist (vgl. [CBJA12]). Zudem erzielten Lee et al. mittels Eye-Tracking, welches auf einer USB-Webcam mit Infrarot LEDs basierte, bei einer 5x4 Button-Matrix eine Trefferquote von 94,6% (vgl. [EM13]). Dem gegenüber steht eine mit dem Face-Tracking in dieser Arbeit ausgehend von der 3x3 Button-Matrix erzielte Trefferquote von 92,39%, 86,38%, 80,95% sowie 78,66%, welche im Anbetracht der Tatsache, dass die Frontkamera eines Tablets verwendet wurde, nicht schlechter erscheint.

Die Vergleiche mit den Ergebnissen anderer Studien zeigen somit, dass sich über die Mobile Vision API mittels Face-Tracking eine barrierefreie Eingabemethode realisieren lässt, die ohne Hilfsmittel- oder anderer kostspieliger Hardware auskommt und dennoch vergleichbare oder teilweise sogar bessere Ergebnisse erzielt als manch andere barrierefreie Eingabemethode.

Die Leistungsfähigkeit des Face-Trackings wäre über eine andere Kamera zudem noch steigerbar. Bian et al. nutzten in ihrer Benutzerstudie eine Tiefenkamera um die Nasenposition zu verfolgen (vgl. [BHCM14]). Die Tiefenkamera stellt sich dabei als vorteilhaft gegenüber einer RGB-Kamera heraus (vgl. [BHCM14]). Je nach Umsetzung erreichten sie mehr als die Hälfte der Bandbreite und Geschwindigkeit einer herkömmlichen Maus (vgl. [BHCM14]). Dieser Optimierungsansatz setzt jedoch voraus, dass die Mobile Vision API Tiefenkameras unterstützt oder die App zur Steuerung des barrierefreien Smarthomes eine andere Bibliothek / Schnittstelle nutzt, die dazu in der Lage ist.

Des Weiteren sollten die quantitativen Benutzbarkeitstests in einem nächsten Schritt mit einer größeren Anzahl an Buttons durchgeführt werden. Denn die vorliegenden Ergebnisse ermöglichen bis jetzt einen Vergleich der Eingabemethoden, aber noch keine Aussage zu deren maximalen Bandbreite. Letzteres liegt daran, dass die Bandbreiten bis zu einer 6x6 Button-Matrix zunehmen und nicht erkennbar ist, ab welcher Anzahl an Buttons das bei der jeweiligen Eingabemethode nicht mehr der Fall und somit ihre maximale Bandbreite erreicht ist. Lediglich beim Face-Tracking, welches mit 3 Probanden bis zu einer 10x10 Button-Matrix getestet wurde, lassen die Ergebnisse vermuten, dass dessen Bandbreite bei mehr als 8x8 Buttons wieder abnimmt und die maximale Bandbreite somit bei ungefähr 7 Bits / Sekunde liegt. Belastbarere Daten würde jedoch auch für das Face-Tracking eine zweite und umfangreichere Benutzerstudie liefern.

Die Bewertung der Benutzeroberfläche ist in den qualitativen Benutzbarkeitstests positiv ausgefallen. Das ist unter anderem auf die großen Buttons zurückzuführen. Sie hatten in der Vertikalen einen Durchmesser von ca. 4,5cm und in der Horizontalen von ca. 7cm. Nach den Beobachtungen des Autors wären die Testpersonen, welche mit ihren Händen die Eingabemethode Touch nutzen konnten, mit der von Guerreiro et al. als ausreichend vorgeschlagenen

Button-Größe von 12mm erheblich schwieriger zurecht gekommen (vgl. [GNJG10b]). In Folge dessen lässt diese Arbeit im Hinblick auf die Gestaltung von Benutzeroberfläche eher die Empfehlung zu, dass diese an die motorischen Fähigkeiten ihrer Benutzers anpassbar sein sollte und es nicht, wie von Guerreiro et al. angedeutet, möglich ist, eine einheitliche Benutzeroberfläche für Benutzer mit und ohne Einschränkungen zu entwickeln (vgl. [GNJG10b]).

7.4. Ausblick

Während der Implementierung des Prototyps sowie im Rahmen der Benutzbarkeitstests sind Ideen zur Erweiterung und Optimierung der App zur Steuerung des barrierefreien Smarthomes entstanden. Die folgenden Auflistung soll diese in einem kurzen Abriss darstellen:

- Die Eingabemethoden 1- und 2-Button-Scanning sollten in ihren Einstellungen um einen Parameter erweitert werden, über den sich ein Mindestabstand zwischen 2 Button-Betätigungen einstellen lässt. Grund hierfür ist, dass es in den qualitativen Benutzbarkeitstests vorkam, dass ein Proband aufgrund geringer motorischer Fähigkeiten den Taster versehentlich 2 mal betätigte, was somit zu einer Fehleingabe führte.
- Bisher ist die Größe der Buttons statisch. Das bedeutet, dass sie bei einer eingestellten 4x4 Button-Matrix immer solche Abmessungen haben, dass 16 Buttons gleichzeitig auf der Benutzeroberfläche anzeigbar sind, auch wenn das ausgewählte Menü beispielsweise nur 5 Elemente besitzt. In solch einem Fall bleibt ein Teil der Displayfläche ungenutzt. Würden sich die Größe der Buttons in dieser Situation dynamisch an den zur Verfügung stehenden Platz anpassen, könnte das insbesondere bei der Eingabemethode Face-Tracking die Bedienung erleichtern.
- Darüber hinaus ist für zukünftige Entwicklungen eine Kombinationsmöglichkeit von Eingabemethoden interessant. Beispielsweise ließe sich das Face-Tracking mit einem Taster, der eigentlich für das Scanning vorgesehen ist, kombinieren. In Folge dessen wären Benutzer, die einen Taster noch bedienen können, in der Lage, die Auswahl des gewünschten Buttons mittels Face-Tracking vorzunehmen und ihn über den Taster zu betätigen, wodurch dann kein Blinzeln erforderlich ist.
- Ein weiterer interessanter Ansatz ist, in einer Benutzerstudie zu evaluieren, ob über eine Gruppenauswahl oder einen binären Suchbaum die Bedienung verbesserbar ist. Würde die Senderliste für den Fernseher beispielsweise in einem binären Suchbaum dargestellt, müsste der Benutzer mehrmals eine Auswahl zwischen 2 Buttons treffen, welche im Gegenzug dafür größer darstellbar sind.
- In den qualitativen Benutzbarkeitstests bewerteten einige Teilnehmer die Eingabemethode Face-Tracking negativ, weil ihnen der Cursor zu unruhig war. Um dieses Problem zu umgehen, könnte das Face-Tracking nicht nur den direkten Modus, der die Position des Kopfes direkt auf jene des Cursors abbildet, sondern noch weitere Modi unterstützen.

Zum Beispiel einen, der vergleichbar mit einem Joystick ist. Dieser könnte so konfiguriert werden, dass sich der Cursor erst bewegt, wenn die Kopfbewegung einen Schwellenwert überschritten hat. Als weiteren Modus bietet es sich an, die Kopfbewegungen in Tastatureingaben zu wandeln, wodurch sich die Buttons in der Anwendung zur Steuerung des barrierefreien Smarthomes tastaturähnlich via Face-Tracking fokussieren ließen.

A. Anhang

A.1. Fragebogen für die Interviews

Die folgenden 5 Seiten beinhalten den Fragebogen, wie er in den im Unterkapitel 3.1 beschriebenen Interviews verwendet wurde.

Barrierefreies Smarthome – Interview

Introduction

Vorstellung

Einleitung

Einverständniserklärung

Teilnahme bestätigen lassen

Aufwandsentschädigung

Warm-up

Demographie

1. Geschlecht:
2. Alter:
3. Sprachen:
4. Abschluss:
5. Haben Sie schon einmal an einem Interview teilgenommen?

Bisherige Nutzung von elektronischen Geräten

6. Welche elektronische Geräte nutzen Sie und wenn ja, seit wann?
7. Für welche Zwecke nutzen Sie die einzelnen Geräte?
8. Wie lange nutzen Sie die einzelnen Geräte durchschnittlich an einem Tag (in Stunden)?

9. Welche Erfahrungen haben Sie mit den einzelnen Geräten?

10. Sind diese Geräte speziell auf die Bedürfnisse Behinderter ausgelegt?

11. Welche Software nutzen Sie und seit wann?

12. Für welche Zwecke nutzen Sie die Software?

13. Wie lange nutzen sie die jeweilige Software durchschnittlich an einem Tag?

14. Welche Erfahrungen haben Sie mit den einzelnen Programmen?

15. Sind diese Programme speziell auf die Bedürfnisse Behinderter ausgelegt?

16. Mit welchen Betriebssystemen (Android, iOS, Windows, Linux) sind Sie vertraut?

17. Welche Erfahrungen haben Sie mit den jeweiligen Betriebssystemen bisher gemacht?

General issues

Smarthome

18. Kennen Sie den Begriff Smarthome?

19. Was stellen Sie sich unter einem Smarthome vor?

20. Welche Geräte / Anwendungen aus dem Smarthome-Bereich haben Sie schon genutzt?

21. In welchen Bereichen könnten Sie sich allgemein vorstellen, dass ein Smarthome sinnvoll ist?

Barrierefreies Smarthome

22. Wie würde ein ideales Smarthome für Sie aussehen?

23. In welchen Bereichen könnte ein barrierefreies Smarthome Sie unterstützen?

Deep focus

Produktvorstellung

Bedienung

24. Wie würden Sie das barrierefreie Smarthome am liebsten steuern / bedienen?

25. Über welches Gerät würden Sie das barrierefreie Smarthome am liebsten steuern?

26. Wie würden Sie eine Steuerung via Blick- oder Gesichtsverfolgung finden?

27. Wie würden Sie eine vollständige / teilweise Sprachsteuerung finden?

28. Wie würden Sie die Menüführung / Benutzeroberfläche gestalten?

29. Welche Vor- und Nachteile hätte Ihrer Meinung nach eine raumadaptive Benutzeroberfläche?

Hardware

30. Wo sollte sich das elektronische Gerät befinden (z. B. stationär, am Rollstuhl, etc.)?

31. Welche Displaygröße?

Privatsphäre

32. Wie sehen Sie das Thema Datenschutz im Bezug auf das barrierefreie Smarthome?

33. Welche Vor- und Nachteile sehen Sie, wenn das barrierefreie Smarthome mit dem Internet verbunden ist?

34. Wäre es für Sie ein Problem, dass die Kamera (bei einer Sprachsteuerung möglicherweise auch das Mikrofon) die ganze Zeit aktiv ist?

35. Hätten Sie eine Idee, wie man barrierefreie Smarthome geschickterweise "ein-" und "ausschaltet"?

Retrospective

36. Welchen Eindruck haben Sie von der beschriebenen Idee eines barrierefreien Smarthomes?

37. Was könnte Ihrer Meinung nach zu Problemen bei dem barrierefreien Smarthome führen?

38. Gibt es von Ihrer Seite noch Verbesserungsvorschläge und / oder Kritik an dem barrierefreien Smarthome sowie dem gerade geführten Interview?

Falls noch offen

39. Können Sie Blinzeln?

40. In wieweit können Sie Ihren Kopf bewegen (drehen, neigen)?

41. Wie kommen Sie mit dem "Sprechen" zurecht?

42. Welche Behinderung(en) haben Sie?

43. Haben Sie die Behinderung(en) von Geburt an?

44. Welche Beeinträchtigungen bestehen durch die Behinderung(en) für Sie?

45. Welche und wie viel Unterstützung ist dadurch erforderlich?

Wrap-up

Abschluss

Ausblick auf die Nutzbarkeitstests

Danke

A.2. Ergebnisse der quantitativen Benutzbarkeitstests

Die folgenden 4 Tabellen beinhaltet Ergebnisse der quantitativen Benutzbarkeitstests aus dem Unterkapitel 6.1. Die Werte in den Tabelle sind auf 4 Stellen nach dem Komma gerundet.

Tabelle A.1.: Die Tabelle beinhaltet die für jede Eingabemethode und Button-Matrix die insgesamt, richtig und falsch übertragenen Bits / Sekunde und davon noch jeweils den maximalen (max), durchschnittlichen (Ø) und minimalen (min) Wert gerundet auf 4 Nachkommastellen.

		gesamt			richtig			falsch		
Eingabemethode	Matrix	max	Ø	min	max	Ø	min	max	Ø	min
Touch	3x3	17,2469	13,7953	9,7720	17,2469	13,7527	9,7720	0,6817	0,0426	0,0000
	4x4	28,3604	23,7948	19,0716	28,3604	23,7017	19,0716	1,4896	0,0931	0,0000
	5x5	45,5419	37,6047	29,3446	45,5419	37,6047	29,3446	0,0000	0,0000	0,0000
	6x6	63,6918	52,5066	39,0150	63,6918	52,5066	39,0150	0,0000	0,0000	0,0000
Maus	3x3	15,5769	12,0600	7,9187	15,5769	11,8741	7,9187	1,1503	0,1859	0,0000
	4x4	22,9867	19,4695	15,8730	22,9867	19,3359	15,8730	1,1688	0,1336	0,0000
	5x5	34,4748	30,0923	25,0000	34,4748	30,0923	25,0000	0,0000	0,0000	0,0000
	6x6	49,0315	40,2690	31,8553	49,0315	39,9996	31,8553	2,3095	0,2694	0,0000
Tastatur	3x3	12,5952	9,6579	6,4342	12,5952	9,6579	6,4342	0,0000	0,0000	0,0000
	4x4	17,8704	14,7568	11,5463	17,8704	14,7568	11,5463	0,0000	0,0000	0,0000
	5x5	28,5950	21,0368	13,4682	28,5950	20,9667	13,4682	1,1214	0,0701	0,0000
	6x6	34,7770	27,9552	20,4823	34,7770	27,7546	20,4823	1,7772	0,2006	0,0000
Face-Tracking	3x3	3,5373	2,5149	1,6079	3,5373	2,3235	1,2584	0,6113	0,1914	0,0000
	4x4	5,7078	4,0400	1,7544	5,1370	3,4899	1,5790	1,2257	0,5501	0,1746
	5x5	8,2966	6,0248	4,3120	7,1114	4,8771	2,5037	1,9184	1,1476	0,3864
	6x6	9,6522	7,5272	4,9446	9,3651	5,9208	3,5601	4,3874	1,6065	0,0000
1-Button-Scanning	3x3	5,4871	4,7545	4,0667	4,9852	4,3557	3,1826	1,1224	0,3988	0,0000
	4x4	6,0380	5,3280	4,6775	6,0380	4,8818	4,1854	0,7642	0,4463	0,0000
	5x5	6,5967	5,7084	5,2196	6,5967	5,2407	4,0866	1,5892	0,4677	0,0000
	6x6	7,5620	6,0766	4,9890	7,1640	5,5608	3,9044	1,0846	0,5157	0,0000
2-Button-Scanning	3x3	6,3629	5,3259	4,2033	6,3629	5,3090	4,2033	0,2697	0,0169	0,0000
	4x4	8,3146	6,4998	4,2872	8,3146	6,4159	4,2872	0,6964	0,0839	0,0000
	5x5	9,0532	7,0913	5,1462	9,0532	7,0117	5,1462	0,5720	0,0796	0,0000
	6x6	9,3868	7,6914	6,2690	9,3868	7,6481	5,9391	0,3630	0,0433	0,0000
Sprachsteuerung	3x3	2,3756	1,9381	1,0845	2,3756	1,9306	1,0845	0,1204	0,0075	0,0000
	4x4	4,2929	3,4469	1,7955	4,2929	3,3530	1,7010	0,4936	0,0939	0,0000
	5x5	6,6791	5,3410	1,4168	6,6791	5,1947	1,2751	0,7330	0,1462	0,0000
	6x6	8,8109	7,7044	5,9265	8,7057	7,4978	5,6146	0,7406	0,2067	0,0000

Tabelle A.2.: Die Tabelle beinhaltet die für jede Eingabemethode und Button-Matrix die Anzahl der insgesamt, richtig und falsch betätigten Buttons / Sekunde und davon noch jeweils den maximalen (max), durchschnittlichen (Ø) und minimalen (min) Wert gerundet auf 4 Nachkommastellen.

		gesamt			richtig			falsch		
Eingabemethode	Matrix	max	Ø	min	max	Ø	min	max	Ø	min
Touch	3x3	1,9163	1,5328	1,0858	1,9163	1,5281	1,0858	0,0757	0,0047	0,0000
	4x4	1,7725	1,4872	1,1920	1,7725	1,4814	1,1920	0,0931	0,0058	0,0000
	5x5	1,8217	1,5042	1,1738	1,8217	1,5042	1,1738	0,0000	0,0000	0,0000
	6x6	1,7692	1,4585	1,0837	1,7692	1,4585	1,0837	0,0000	0,0000	0,0000
Maus	3x3	1,7308	1,3400	0,8799	1,7308	1,3193	0,8799	0,1278	0,0207	0,0000
	4x4	1,4367	1,2168	0,9921	1,4367	1,2085	0,9921	0,0731	0,0084	0,0000
	5x5	1,3790	1,2037	1,0000	1,3790	1,2037	1,0000	0,0000	0,0000	0,0000
	6x6	1,3620	1,1186	0,8849	1,3620	1,1111	0,8849	0,0642	0,0075	0,0000
Tastatur	3x3	1,3995	1,0731	0,7149	1,3995	1,0731	0,7149	0,0000	0,0000	0,0000
	4x4	1,1169	0,9223	0,7216	1,1169	0,9223	0,7216	0,0000	0,0000	0,0000
	5x5	1,1438	0,8415	0,5387	1,1438	0,8387	0,5387	0,0449	0,0028	0,0000
	6x6	0,9660	0,7765	0,5690	0,9660	0,7710	0,5690	0,0494	0,0056	0,0000
Face-Tracking	3x3	0,3930	0,2794	0,1787	0,3930	0,2582	0,1398	0,0679	0,0213	0,0000
	4x4	0,3567	0,2525	0,1097	0,3211	0,2181	0,0987	0,0766	0,0344	0,0109
	5x5	0,3319	0,2410	0,1725	0,2845	0,1951	0,1001	0,0767	0,0459	0,0155
	6x6	0,2681	0,2091	0,1374	0,2601	0,1645	0,0989	0,1219	0,0446	0,0000
1-Button-Scanning	3x3	0,6097	0,5283	0,4519	0,5539	0,4840	0,3536	0,1247	0,0443	0,0000
	4x4	0,3774	0,3330	0,2923	0,3774	0,3051	0,2616	0,0478	0,0279	0,0000
	5x5	0,2639	0,2283	0,2088	0,2639	0,2096	0,1635	0,0636	0,0187	0,0000
	6x6	0,2101	0,1688	0,1386	0,1990	0,1545	0,1085	0,0301	0,0143	0,0000
2-Button-Scanning	3x3	0,7070	0,5918	0,4670	0,7070	0,5899	0,4670	0,0300	0,0019	0,0000
	4x4	0,5197	0,4062	0,2679	0,5197	0,4010	0,2679	0,0435	0,0052	0,0000
	5x5	0,3621	0,2837	0,2058	0,3621	0,2805	0,2058	0,0229	0,0032	0,0000
	6x6	0,2607	0,2137	0,1741	0,2607	0,2124	0,1650	0,0101	0,0012	0,0000
Sprachsteuerung	3x3	0,2640	0,2153	0,1205	0,2640	0,2145	0,1205	0,0134	0,0008	0,0000
	4x4	0,2683	0,2154	0,1122	0,2683	0,2096	0,1063	0,0309	0,0059	0,0000
	5x5	0,2672	0,2136	0,0567	0,2672	0,2078	0,0510	0,0293	0,0058	0,0000
	6x6	0,2447	0,2140	0,1646	0,2418	0,2083	0,1560	0,0206	0,0057	0,0000

Tabelle A.3.: Die Tabelle beinhaltet die insgesamt, richtig und falsch übertragenen Bits / Sekunde und davon noch jeweils den maximalen (max), durchschnittlichen (Ø) und minimalen (min) Wert gerundet auf 4 Nachkommastellen der 3 beziehungsweise 2 Probanden, welche die Eingabemethode Face-Tracking mit mehr als 36 Buttons testeten.

	gesamt			richtig			falsch		
Matrix	max	Ø	min	max	Ø	min	max	Ø	min
3x3	3,3624	2,3068	1,6079	2,7510	1,9215	1,2584	0,6113	0,3853	0,1950
4x4	5,1377	4,1661	3,1517	4,0208	3,4337	2,8366	1,1169	0,7324	0,3152
5x5	6,5769	5,6566	4,3120	5,6373	4,4511	2,5037	1,8082	1,2055	0,8687
6x6	9,6522	7,4373	5,2524	6,0605	4,9872	3,6363	4,3874	2,4501	1,3468
7x7	9,7980	8,0901	7,1046	5,8788	4,8160	3,9964	3,9192	3,2740	2,7946
8x8	17,6113	12,6128	9,3152	9,9064	7,1782	5,0810	7,7049	5,4346	4,2342
9x9	12,8340	11,6448	10,4556	6,4897	6,2065	5,9234	6,9106	5,4383	3,9659
10x10	15,5846	12,9141	10,0293	9,0888	5,8793	3,5398	10,5753	7,0348	4,0395

Tabelle A.4.: Die Tabelle beinhaltet die insgesamt, richtig und falsch betätigten Buttons / Sekunde und davon noch jeweils den maximalen (max), durchschnittlichen (Ø) und minimalen (min) Wert gerundet auf 4 Nachkommastellen der 3 beziehungsweise 2 Probanden, welche die Eingabemethode Face-Tracking mit mehr als 36 Buttons testeten.

	gesamt			richtig			falsch		
Matrix	max	Ø	min	max	Ø	min	max	Ø	min
3x3	0,3736	0,2563	0,1787	0,3057	0,2135	0,1398	0,0679	0,0428	0,0217
4x4	0,3211	0,2604	0,1970	0,2513	0,2146	0,1773	0,0698	0,0458	0,0197
5x5	0,2631	0,2263	0,1725	0,2255	0,1780	0,1001	0,0723	0,0482	0,0347
6x6	0,2681	0,2066	0,1459	0,1683	0,1385	0,1010	0,1219	0,0681	0,0374
7x7	0,2000	0,1651	0,1450	0,1200	0,0983	0,0816	0,0800	0,0668	0,0570
8x8	0,2752	0,1971	0,1456	0,1548	0,1122	0,0794	0,1204	0,0849	0,0662
9x9	0,1584	0,1438	0,1291	0,0801	0,0766	0,0731	0,0853	0,0671	0,0490
10x10	0,1558	0,1291	0,1003	0,0909	0,0588	0,0354	0,1058	0,0703	0,0404

Literaturverzeichnis

[AA14] T. Ahne, S. Ahne. „Risiko Querschnittslähmung–Was tun bei Wirbelsäulenverletzungen?“ In: *retten!* 3.04 (2014), S. 258–265 (zitiert auf S. 11, 18, 19).

[BCC] A. Bulbul, Z. Cipiloglu, T. Capin. „A Face Tracking Algorithm for User Interaction in Mobile Devices“. In: *2009 International Conference on CyberWorlds*, S. 385–390. DOI: 10.1109/CW.2009.9 (zitiert auf S. 25).

[BHCM14] Z.-P. Bian, J. Hou, L.-P. Chau, N. Magnenat-Thalmann. „Human Computer Interface for Quadriplegic People Based on Face Position/Gesture Detection“. In: *Proceedings of the 22Nd ACM International Conference on Multimedia*. MM '14. New York, NY, USA: ACM, 2014, S. 1221–1224. ISBN: 978-1-4503-3063-3 (zitiert auf S. 23, 132).

[CBJA12] H. A. Caltenco, B. Breidegard, B. Jönsson, L. N. Andreasen Struijk. „Understanding Computer Users With Tetraplegia: Survey of Assistive Technology Users“. In: *International Journal of Human-Computer Interaction* 28.4 (2012), S. 258–268. ISSN: 1044-7318 (zitiert auf S. 21, 38, 132).

[Che01] Y. L. Chen. „Application of tilt sensors in human-computer mouse interface for people with disabilities“. In: *IEEE transactions on neural systems and rehabilitation engineering : a publication of the IEEE Engineering in Medicine and Biology Society* 9.3 (2001), S. 289–294. ISSN: 1534-4320 (zitiert auf S. 21).

[DAH12] M. L. Dybdal, J. S. Agustin, J. P. Hansen. „Gaze Input for Mobile Devices by Dwell and Gestures“. In: *Proceedings of the Symposium on Eye Tracking Research and Applications*. ETRA '12. Santa Barbara, California: ACM, 2012, S. 225–228. ISBN: 978-1-4503-1221-9. DOI: 10.1145/2168556.2168601. URL: http://doi.acm.org/10.1145/2168556.2168601 (zitiert auf S. 25, 54).

[DLS07] H. Drewes, A. de Luca, A. Schmidt. „Eye-gaze Interaction for Mobile Phones“. In: *Proceedings of the 4th International Conference on Mobile Technology, Applications, and Systems and the 1st International Symposium on Computer Human Interaction in Mobile Technology*. Mobility '07. New York, NY, USA: ACM, 2007, S. 364–371. ISBN: 978-1-59593-819-0. DOI: 10.1145/1378063.1378122. URL: http://doi.acm.org/10.1145/1378063.1378122 (zitiert auf S. 25).

[EM13] Eui Chul Lee, Min Woo Park. „New Eye Tracking Method as a Smartphone Interface“. In: *KSII Transactions on Internet and Information Systems* 7.4 (2013), S. 834–848. ISSN: 19767277 (zitiert auf S. 24, 132).

[Exn04] G. Exner. „Der Arbeitskreis Querschnittlähmungen des Hauptverbandes der gewerblichen Berufsgenossenschaften in Deutschland“. In: *Trauma und Berufskrankheit* 6.2 (2004). ISSN: 1436-6274 (zitiert auf S. 19).

[FN08] T. Felzer, R. Nordmann. „Evaluating the Hands-Free Mouse Control System: An Initial Case Study“. In: *Computers Helping People with Special Needs: 11th International Conference, ICCHP 2008, Linz, Austria, July 9-11, 2008. Proceedings*. Hrsg. von K. Miesenberger, J. Klaus, W. Zagler, A. Karshmer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, S. 1188–1195. ISBN: 978-3-540-70540-6. DOI: 10.1007/978-3-540-70540-6_179. URL: http://dx.doi.org/10.1007/978-3-540-70540-6_179 (zitiert auf S. 22).

[FWK07] J. Froehlich, J. O. Wobbrock, S. K. Kane. „Barrier Pointing: Using Physical Edges to Assist Target Acquisition on Mobile Device Touch Screens“. In: *Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility*. Assets '07. New York, NY, USA: ACM, 2007, S. 19–26. ISBN: 978-1-59593-573-1. DOI: 10.1145/1296843.1296849. URL: http://doi.acm.org/10.1145/1296843.1296849 (zitiert auf S. 24).

[GLF+12] S. M. Grigorescu, T. Lüth, C. Fragkopoulos, M. Cyriacks, A. Gräser. „A BCI-controlled robotic assistant for quadriplegic people in domestic and professional life“. In: *Robotica* 30.03 (2012), S. 419–431. ISSN: 0263-5747 (zitiert auf S. 22).

[GNJG10a] T. J. V. Guerreiro, H. Nicolau, J. Jorge, D. Gonçalves. „Assessing Mobile Touch Interfaces for Tetraplegics“. In: *Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services*. MobileHCI '10. New York, NY, USA: ACM, 2010, S. 31–34. ISBN: 978-1-60558-835-3. DOI: 10.1145/1851600.1851608. URL: http://doi.acm.org/10.1145/1851600.1851608 (zitiert auf S. 24).

[GNJG10b] T. Guerreiro, H. Nicolau, J. Jorge, D. Gonçalves. „Towards Accessible Touch Interfaces“. In: *Proceedings of the 12th International ACM SIGACCESS Conference on Computers and Accessibility*. ASSETS '10. Orlando, Florida, USA: ACM, 2010, S. 19–26. ISBN: 978-1-60558-881-0. DOI: 10.1145/1878803.1878809. URL: http://doi.acm.org/10.1145/1878803.1878809 (zitiert auf S. 24, 133).

[Goo16a] Google. *Change Voice Access settings*. 2016. URL: https://support.google.com/accessibility/android/answer/6151843 (besucht am 17. 12. 2016) (zitiert auf S. 114).

[Goo16b] Google. *Face Detection Concepts Overview*. 2016. URL: https://developers.google.com/vision/face-detection-concepts (besucht am 03. 11. 2016) (zitiert auf S. 12, 50–53, 80, 85).

[Goo16c] Google. *Get Started with the Mobile Vision API*. 2016. URL: https://developers.google.com/vision/android/getting-started (besucht am 09. 12. 2016) (zitiert auf S. 50).

[Goo16d] Google. *Google APIs for Android Face*. 2016. URL: https://developers.google.com/android/reference/com/google/android/gms/vision/face/Face (besucht am 08. 12. 2016) (zitiert auf S. 51, 87).

[Goo16e] Google. *Google TalkBack*. 2016. URL: https://play.google.com/store/apps/details?id=com.google.android.marvin.talkback&hl=de (besucht am 08. 12. 2016) (zitiert auf S. 46, 71).

[Goo16f] Google. *Landmark*. 2016. URL: https://developers.google.com/android/reference/com/google/android/gms/vision/face/Landmark (besucht am 09. 12. 2016) (zitiert auf S. 51).

[Goo16g] Google. *Schalterzugriff für Android einrichten*. 2016. URL: https://support.google.com/accessibility/android/answer/6301490 (besucht am 14. 12. 2016) (zitiert auf S. 20, 130).

[Goo16h] Google. *Tipps für die Nutzung des Schalterzugriffs*. 2016. URL: https://support.google.com/accessibility/android/answer/6395627 (besucht am 14. 12. 2016) (zitiert auf S. 20).

[Goo16i] Google. *Troubleshoot Voice Access*. 2016. URL: https://support.google.com/accessibility/android/answer/6377053?hl=en&ref_topic=6151842 (besucht am 09. 12. 2016) (zitiert auf S. 56, 114, 131).

[Goo16j] Google. *Über den Schalterzugriff für Android*. 2016. URL: https://support.google.com/accessibility/android/answer/6122836 (besucht am 15. 12. 2016) (zitiert auf S. 20, 130, 131).

[Goo16k] Google. *Übersicht über die Android-Bedienungshilfen*. 2016. URL: https://support.google.com/accessibility/android/answer/6006564?hl=de (besucht am 14. 12. 2016) (zitiert auf S. 19, 20, 130).

[Goo16l] Google. *Use Voice Access commands*. 2016. URL: https://support.google.com/accessibility/android/answer/6151854 (besucht am 09. 12. 2016) (zitiert auf S. 56).

[Goo16m] Google. *Voice Access (Unreleased)*. 2016. URL: https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.voiceaccess&hl=de (besucht am 08. 12. 2016) (zitiert auf S. 56, 61, 62, 107).

[GR04] E. Gamma, D. Riehle. *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. 1. Aufl., [Neuaufl.] Programmer's choice. München und Boston [u.a.]: Addison-Wesley, 2004. ISBN: 3827321999 (zitiert auf S. 59, 77).

[Har03] R. Harper. *Inside the smart home*. London und New York: Springer, 2003. ISBN: 978-1-85233-854-1 (zitiert auf S. 15).

[JDO16] JDOM. *JDOM*. 2016. URL: http://www.jdom.org/index.html (besucht am 19. 12. 2016) (zitiert auf S. 69).

[JKS05] H. Jeong, J.-S. Kim, W.-H. Son. „An emg-based mouse controller for a tetraplegic". In: *2005 IEEE International Conference on Systems, Man and Cybernetics*. Bd. 2. IEEE. 2005, S. 1229–1234 (zitiert auf S. 22, 132).

[KAR+14] J. Kangas, D. Akkil, J. Rantala, P. Isokoski, P. Majaranta, R. Raisamo. „Gaze Gestures and Haptic Feedback in Mobile Devices“. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '14. Toronto, Ontario, Canada: ACM, 2014, S. 435–438. ISBN: 978-1-4503-2473-1. DOI: 10.1145/2556288.2557040. URL: http://doi.acm.org/10.1145/2556288.2557040 (zitiert auf S. 25, 90).

[KGM12] M. Kuniavsky, E. Goodman, A. Moed. *Observing the user experience: A practitioner's guide to user research*. 2nd ed. Waltham, MA: Morgan Kaufmann, 2012. ISBN: 978-0123848697 (zitiert auf S. 27–33, 120–122, 124).

[KNL+06] L. Kauhanen, T. Nykopp, J. Lehtonen, P. Jylanki, J. Heikkonen, P. Rantanen, H. Alaranta, M. Sams. „EEG and MEG brain-computer interface for tetraplegic patients“. In: *IEEE transactions on neural systems and rehabilitation engineering : a publication of the IEEE Engineering in Medicine and Biology Society* 14.2 (2006), S. 190–193. ISSN: 1534-4320. DOI: 10.1109/TNSRE.2006.875546 (zitiert auf S. 22).

[Kod16a] Kodi. *About*. 2016. URL: https://kodi.tv/about/ (besucht am 08. 12. 2016) (zitiert auf S. 63).

[Kod16b] Kodi. *JSON-RPC API/v6*. 2016. URL: http://kodi.wiki/view/JSON-RPC_API/v6 (besucht am 08. 12. 2016) (zitiert auf S. 63).

[Kod16c] Kodi. *Remote controls*. 2016. URL: http://kodi.wiki/view/Remote_controls (besucht am 08. 12. 2016) (zitiert auf S. 63).

[LCC+10] M. E. Lund, H. V. Christiensen, H. A. Caltenco, E. R. Lontis, B. Bentsen, Andreasen Struijk, Lotte N S. „Inductive tongue control of powered wheelchairs“. In: *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual Conference* 2010 (2010), S. 3361–3364. ISSN: 1557-170X (zitiert auf S. 22).

[NW11] M. Nerlich, B. Weigel. *Praxisbuch Unfallchirurgie: Mit 161 Tabellen*. 2., vollst. überarb. und aktualisierte Aufl. Berlin [u.a.]: Springer, 2011. ISBN: 3642107893 (zitiert auf S. 16).

[Org13] W. H. Organization. *International Perspectives on Spinal Cord Injury*. Nonserial Publications. Geneva: World Health Organization, 2013. ISBN: 978 92 4 156466 3 (zitiert auf S. 11, 16–19, 28).

[Pec14] L. Pecchia. *Ambient assisted living and daily activities: 6th international workconference, IWAAL 2014, Belfast, UK, December 2-5, 2014 : proceedings*. Bd. 8868. LNCS sublibrary. SL 3, Information systems and applications, incl. Internet/Web, and HCI490. Cham und New York: Springer, 2014. ISBN: 978-3-319-13104-7 (zitiert auf S. 11, 15).

[Sma16] Smart Homes. *The integration of technology and services in the home environment*. 2016. URL: http://www.smart-homes.nl/Domotica.aspx (besucht am 11. 12. 2016) (zitiert auf S. 11, 15).

[Spa91] B. Spahn. „Fachkundenachweis Rettungsdienst“. In: Hrsg. von P. D. m. P.-M. O. Dr. med. Klaus Ellinger Dr. med. Hartmuth Frobenius. Springer Berlin Heidelberg, 1991. Kap. Wirbelsäulenverletzung und Querschnittslähmung, S. 206–212. ISBN: 978-3-642-97232-4 (zitiert auf S. 19).

[SSHH09] J. San Agustin, H. Skovsgaard, J. P. Hansen, D. W. Hansen. „Low-cost Gaze Interaction: Ready to Deliver the Promises“. In: *CHI '09 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '09. Boston, MA, USA: ACM, 2009, S. 4453–4458. ISBN: 978-1-60558-247-4. DOI: 10.1145/1520340.1520682. URL: http://doi.acm.org/10.1145/1520340.1520682 (zitiert auf S. 25).

[Sta16] Statistisches Bundesamt. *Sozialleistungen: Schwerbehinderte Menschen*. 2016. URL: https://www.destatis.de/DE/Publikationen/Thematisch/Gesundheit/BehinderteMenschen/Schwerbehinderte2130510139004.pdf;jsessionid=8D754AAAB4D613729528E85F675EA38E.cae2?__blob=publicationFile (besucht am 09. 12. 2016) (zitiert auf S. 19).

[TRF+05] Y. Tanimoto, Y. Rokumyo, K. Furusawa, A. Tokuhiro, Y. Suzuki, K. Takami, H. Yamamoto. „Development of a computer input device for patients with tetraplegia“. In: *Computer Standards & Interfaces* 28.2 (2005), S. 166–175. ISSN: 09205489 (zitiert auf S. 23).

[Uen14] T. Ueno. *A development of the interface to operate Smartphones for quadriplegic people (The 3rd)*. 2014. URL: http://www.yokohama-rf.jp/common/pdf/report/26-5.pdf (besucht am 15. 12. 2016) (zitiert auf S. 23).

[Vid16a] VideoLAN. *VLC Features*. 2016. URL: http://www.videolan.org/vlc/features.html (besucht am 08. 12. 2016) (zitiert auf S. 63).

[Vid16b] VideoLAN. *VLC HTTP requests*. 2016. URL: https://wiki.videolan.org/VLC_HTTP_requests/ (besucht am 08. 12. 2016) (zitiert auf S. 63, 92–94).

[Wir11] R. Wirdemann. *Scrum mit User Stories*. 2., erweiterte Auflage. München: Hanser, Carl, 2011. ISBN: 9783446426603 (zitiert auf S. 37).

[WK08] M. R. Williams, R. F. Kirsch. „Evaluation of head orientation and neck muscle EMG signals as command inputs to a human-computer interface for individuals with high tetraplegia“. In: *IEEE transactions on neural systems and rehabilitation engineering : a publication of the IEEE Engineering in Medicine and Biology Society* 16.5 (2008), S. 485–496. ISSN: 1534-4320 (zitiert auf S. 21, 131).

[Wöh16a] Wöhlke EDV-Beratung. *Wöhlke Websteckdose - Die IP-Steckdose mit WLAN*. 2016. URL: https://www.woehlke-websteckdose.de/index.php?id=websteckdose_ip-steckdose_servic (besucht am 08. 12. 2016) (zitiert auf S. 63, 94).

[Wöh16b] Wöhlke EDV-Beratung. *Wöhlke Websteckdose - Häufig gestellte Fragen*. 2016. URL: https://www.woehlke-websteckdose.de/index.php?id=websteckdose-faq0 (besucht am 08. 12. 2016) (zitiert auf S. 63, 94, 95).

[WW06] M. Wyndaele, J.-J. Wyndaele. „Incidence, prevalence and epidemiology of spinal cord injury: what learns a worldwide literature survey?“ In: *Spinal cord* 44.9 (2006), S. 523–529 (zitiert auf S. 11, 19).

Alle URLs wurden zuletzt am 19. 12. 2016 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift