Universität Regensburg

Kombination von Gesichtserkennung und Sprachsteuerung zur natürlichen Interaktion mit Haushaltsgeräten im Smart Home

Masterarbeit im Fach Medieninformatik am Institut für Information und Medien, Sprache und Kultur (I:IMSK)

Vorgelegt von: Alexander Wolz
Adresse: Untere Bergstraße 12, 63934 Röllbach
E-Mail (Universität): alexander.wolz@stud.uni-regensburg.de
E-Mail (privat): alexander.wolz@gmx.de
Matrikelnummer: 1739094
Erstgutachter: Prof. Dr. Bernd Ludwig
Zweitgutachter: Prof. Dr. Christian Wolff
Betreuer: Prof. Dr. Bernd Ludwig
Laufendes Semester: Wintersemester 2016/2017 (4. Fachsemester)
Abgegeben am: 30.03.2017

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmen

Zusammenfassung

Diese Arbeit befasst sich mit der Frage, ob eine Kombination aus Gesichtserkennung und Sprachsteuerung geeignet ist, um eine intuitive Kontaktaufnahme miteinander verbundener Haushaltsgeräte im Smart Home zu initialisieren. Der Fokus liegt hier vor allem auf der Konzeption und Umsetzung eines Prototypen, der eine intuitive Aktivierung von natürlichsprachlicher Mensch-Maschine-Kommunikation in pervasiven Umgebungen ermöglicht. Dafür wurde ein verteiltes System auf modularer Basis entwickelt, das alle mit diesem System vernetzten Haushaltsgeräte in einen Ruhemodus versetzt und eine Sprachsteuerung aktiviert, sobald der Benutzer gerade in eine der dort angeschlossenen Kameras schaut. Hierzu bestimmt eine *Head Pose Estimation* die Lage des Kopfes zur Kamera und aktiviert die Sprachsteuerung, wenn das Gesicht in einem bestimmten Winkel steht. Zuerst werden themenverwandte Arbeiten betrachtet und das darauf aufbauende Konzept vorgestellt. Danach wird der modulare Aufbau des Systems sowie die wichtigsten Grundfunktionen beschrieben. In einem anschließenden Experiment mit 30 Teilnehmern konnte herausgefunden werden, dass eine durchschnittliche Erkennung des Anwenders etwa 5 Sekunden benötigt. Die Precision liegt hierbei bei 97%, der Recall bei 88% und die False-Alarm-Rate bei 75%. Die intuitive Wahrnehmung aller Probanden wurde abschließend über einen standardisierten QUESI-Fragebogen ermittelt, der einen Gesamt-Score von 4,21 erzielte. Im letzten Kapitel wird schließlich eine Reflektion der Umsetzung sowie einen Ausblick auf weiterführende Möglichkeiten dieses Prototyps gegeben.

Abstract

This master thesis pursues the question of whether a combination of face detection and speech control is suitable for initializing an intuitive contact between connected household devices in the Smart Home. The main focus hereby lies in the design and implementation of a prototype that enables an intuitive activation of natural Human-Machine-Communicaton in pervasive environments. For this purpose, a modular distributed system has been developed, which puts all connected household devices into a silence mode and activates a voice control as soon as the user looks straight ahead into one of the connected cameras. Therefore, a Head Pose Estimation determines the location of the head and activates the speech control if the face is inside a certain angle. At first, the related work and the concept based on it is presented. Afterwards, the modular system and the most important functionalities will be explained in more detail. In a subsequent experiment with 30 participants, it could be found out that an average recognition of the user takes about 5 seconds. The Precision lies hereby at 97%, the Recall at 88% and the False-Alarm-Rate at 75%. The intuitive perception of all participants was finally determined by means of a standardized QUESI questionnaire, which achieved a total score of 4,21. Finally, the last chapter provides a reflection on the implementation, as well as an outlook on further possibilities of this prototype.

1 Einleitung

Im *Internet der Dinge* sind physische Gegenstände, angefangen von Banknoten bis hin zu Fahrrädern, durch ein Netzwerk miteinander verbunden und beteiligen sich so aktiv am Internet, um Informationen über sich und ihre Umgebung auszutauschen. Dies ermöglicht einen sofortigen Zugang zu den Informationen über die physische Welt und der darin enthaltenen Objekte, was einerseits zu innovativen Dienstleistungen, aber auch zu einer Steigerung von Effizienz und Produktivität führen kann (Commission of The European Communities, 2008). Da diese intelligenten Gegenstände unterschiedliche Aufgaben besitzen, werden sie weiterhin domänenspezifischen Anwendungsbereichen zugeordnet, wozu insbesondere das Gesundheitswesen, die Automobilindustrie, der Transportsektor, aber auch das *Smart Home* gehören (Al-Fuqaha et al., 2015).

Für die Zahl der weltweit mit dem Internet verbundenen Gegenstände prognostiziert das Marktforschungsunternehmen Gartner (2014) ein rasantes Wachstum. Es wird hochgerechnet, dass bis zum Jahr 2020 rund 25 Milliarden Dinge an das Internet angebunden sind. Der am schnellsten wachsende Bereich ist hierbei der Verbrauchersektor, wozu auch das Smart Home gehört. Waren es 2015 noch rund 2,9 Milliarden verbundene Geräte, so soll sich die Zahl bis 2020 auf etwa 13,2 Milliarden fast verfünffachen.

In einer weiteren Studie untersucht Gartner (2015) das Wachstum in der Kategorie *Smart City*, die unter anderem mit den Anwendungsbereichen Gesundheitswesen, Transport und Smart Home interagiert. Hier sollen bis 2018 insgesamt rund 3,3 Milliarden Gegenstände vernetzt sein. Das entspricht einem Wachstum von über 40 Prozent zum Vorjahreswert von etwa 2,3 Milliarden. Vergleicht man die Zahlen in Tabelle 1, wird schnell deutlich, dass die Investitionen im Bereich *Smart Commercial Buildings* bis 2017 den größten Anteil ausmachen, jedoch ab 2018 die Führung an den Bereich *Smart Home* abgeben werden.

Smart City Subcategory	**2015**	**2016**	**2017**	**2018**
Healthcare	3,4	5,3	8,4	13,4
Public Services	78,6	103,6	133,1	167,4
Smart Commercial Buildings	377,3	518,1	733,7	1.064,8
Smart Homes	174,3	339,1	621,8	1.073,7
Transport	276,9	347,5	429,2	517,4
Utilities	260,6	314,0	380,6	463,5
Others	8,6	13,3	20,8	32,3
Total	**1.179,7**	**1.641,0**	**2.327,7**	**3.332,5**

Tabelle 1: Verbundene Dinge in Millionen (Gartner, 2015)

Dies liegt Gartner (2015) zufolge mitunter an der wachsenden Marktreife von Smart Home Plattformen und deren Ökosystem aus Haushaltsgeräten, Infotainmentsystemen und Sensoren. Wachstumstreibend sind hierbei intelligente Fernseher, Beistellgeräte und Leuchten, sowie Werkzeuge der Hausautomatisierung (wie etwa intelligente Thermostate), Sicherheitstechnik und Küchengeräte. Gerade aber in letzterem Bereich ist derzeit ein starker Trend zu beobachten:

> „ Der Einfluss moderner Anzeige- und Bedientechnologien prägt zunehmend auch die Gestaltung von Hausgeräten. [...] Im Zuge der Vernetzung werden künftig nicht-essentielle Funktionen verstärkt auf mobile Geräte ausgelagert. Damit können die Bedienoberflächen der Hausgeräte selbst noch weiter vereinfacht werden. “ (Robert Bosch Hausgeräte GmbH, 2016, S. 1)

Viele Hersteller rüsten ihre Hausgeräte also mit einer Schnittstelle aus, um diese entfernt mit mobilen Endgeräten anzusteuern. Auf den ersten Blick erscheint dies recht komfortabel, doch fällt einem schnell auf, dass jeder Anbieter seine eigene App bereitstellt (Stabit, 2016). Mittlerweile gibt

es aber durchaus quelloffene und herstellerunabhängige Ansätze um diese Schnittstellen zu kombinieren. Beispielsweise bietet das Projekt *openHAB* systemübergreifende Automatisierungsregeln und einheitliche Benutzeroberflächen als native und webbasierte Apps an. (Kreuzer & Eichstädt-Engelen, 2015).

Es gibt jedoch Alltagssituationen, in denen das Bedienen eines mobilen Endgeräts oder das Betätigen von Tasten kaum oder gar nicht möglich ist. Ein typisches Szenario wäre hierbei das Backen und Kochen in der Küche. Man knetet beispielsweise gerade seinen Hefeteig und möchte währenddessen ein intelligentes Haushaltsgerät bedienen. Das Bedienfeld des Geräts sowie das Smartphone erkennen wegen der fettigen und klebrigen Finger keine Touch-Eingaben, weshalb man zunächst seine Hände waschen muss. Eine alternative Bedienung könnte daher über die integrierte Sprachsteuerung moderner Smartphones geschehen, falls die Hersteller-Apps dies unterstützen. Da aber die Dunstabzugshaube gerade auf höchster Stufe brummt, werden die Sprachbefehle nicht richtig erkannt.

Das Ziel dieser Masterarbeit ist daher das Entwickeln eines prototypischen Systems, das durch die Kombination von Gesichtserkennung und Sprachsteuerung eine intuitive Kontaktaufnahme mit den Haushaltsgeräten initialisieren kann, ohne dass hierbei mobile Endgeräte, Bedienfelder oder Tasten mit den Händen bedient werden müssen. Gleichzeitig sollen alle mit dem System verbundene Haushaltsgeräte in einen Ruhemodus wechseln, um die darauf folgende Spracherkennung nicht zu beeinträchtigen.

Dazu wird zunächst der aktuelle Stand der Forschung betrachtet und das darauf aufbauende Konzept definiert. Anschließend werden der Aufbau und die wichtigsten Funktionen der Implementierung dargestellt. In einem abschließenden Experiment wird sodann die Effektivität der Gesichtserkennung sowie die subjektive Wahrnehmung des Gesamtsystems evaluiert, bevor schließlich eine Reflektion der Umsetzung und ein Ausblick auf weiterführende Möglichkeiten gegeben wird.

2 Stand der Forschung und Technik

Um die geeignete Technologie für den Prototypen finden zu können, sollte man sich zunächst mit der entsprechenden Theorie und dem aktuellen Stand der Technik vertraut machen. Die Anforderungen, die sich an Geräte mit natürlicher Benutzerinteraktion stellen, wurden bereits in zahlreichen Projekten analysiert und daraufhin Lösungen erarbeitet, die in diesem Kapitel nun näher betrachtet werden.

2.1 Smart Home

Der Begriff *Smart Home* ist in der Literatur bis heute noch nicht eindeutig definiert und wird synonym mit den Begriffen *Connected Home*, *Elektronisches Haus*, *Intelligentes Wohnen*, *Smart House*, *Smart Environment*, *Home of the Future*, *Smart Living* und *Aware Home* verwendet (Strese et al., 2010). Abzugrenzen sind diese aber von Bezeichnungen wie *Smart Building* oder *intelligentes Gebäude*, worunter in der Regel mehrere, räumlich getrennte Bauten zu verstehen sind, die unter betriebswirtschaftlichen Gesichtspunkten verwaltet werden (Strese et al., 2010).

Mitglieder der Fokusgruppe Connected Home (2014) beschreiben das Smart Home als einen Oberbegriff für technische Verfahren und Systeme in Wohnräumen und -häusern, in deren Mittelpunkt eine Erhöhung von Wohn- und Lebensqualität, Sicherheit und effizienter Energienutzung auf Basis vernetzter und fernsteuerbarer Geräte und Installationen sowie automatisierbarer Abläufe steht. Weiterhin definiert Strese et al. (2010) dies als ein privat genutztes Heim, in welchem die zahlreichen Geräte der Hausautomation (wie Heizung, Beleuchtung, Belüftung), Haushaltstechnik (wie z.B. Kühlschrank, Waschmaschine), Konsumelektronik und Kommunikationseinrichtungen zu intelligenten Gegenständen werden, die sich an den Bedürfnissen der Bewohner orientieren. Durch Vernetzung dieser Gegenstände untereinander können neue Assistenzfunktionen und Dienste zum

Nutzen des Bewohners bereitgestellt werden und einen Mehrwert generieren, der über den einzelnen Gebrauch der im Haus vorhandenen Anwendungen hinausgeht (Strese et al., 2010).

Die Hausgeräte müssen also nicht zwingend an das Internet angebunden sein, obwohl das Smart Home als Anwendungsbereich des Internet der Dinge zählt. Sie können auch autark im eigenen Netz agieren, was man dann als *Intranet der Dinge* bezeichnet. Allerdings wird dieser Grundgedanke gegenwärtig in Frage gestellt, da viele Produkte sich nach dem Einschalten zunächst über das Internet bei ihren Herstellerservern anmelden und von dort ihre Befehle beziehen (Kreuzer & Eichstädt-Engelen, 2015).

Dementsprechend geht es beim Smart Home in erster Linie um die Vernetzung elektronischer Verbraucher in Privathaushalten sowie deren Automation und Fernsteuerung. Basierend auf der Literaturrecherche von Badica et al. (2013) wurden vier Hauptanwendungsgebiete identifiziert, die zum Teil auch miteinander verbunden sein können:

1. Ältere Menschen und Hauspflege
2. Energieeffizienz
3. Komfort und Unterhaltung
4. Sicherheit

Der Bereich *Ältere Menschen und Hauspflege* deckt die Anforderungen einer immer älter werdenden Bevölkerung ab, die in Beziehung mit Gesundheit, Einsamkeit, Behinderung sowie kognitiver Einschränkung stehen, während im Bereich *Komfort und Unterhaltung* die Umgebungssteuerungen für beispielsweise Licht und Hintergrundmusik, aber auch Erweiterte Benutzeroberflächen mit Gesten- oder Sprachsteuerung im Vordergrund stehen (Badica et al., 2013). Ersteres ist stark mit dem Feld des *Ambient Assisted Living (AAL)* verwandt, das technische Systeme zur Unterstützung von Hilfsbedürftigen im Alltag umfasst. Ziel ist der Erhalt und die Förderung der Selbständigkeit von Personen bis ins hohe Alter und die Qualitätsverbes-

serung von Hilfs- und Unterstützungsdienstleistungen, sowie Angeboten im häuslichen Bereich (Georgieff, 2008). Das Anwendungsgebiet *Energieeffizienz* zielt auf eine Reduzierung des Energiekonsums elektronischer Verbraucher ab, wobei sich der Bereich *Sicherheit* auf die Erkennung von Gefahrensituationen oder Einbrüchen fokussiert (Badica et al., 2013).

Da aber eine kontaktlose Interaktion mit Haushaltsgeräten einerseits zur aktiven Steigerung der Lebensqualität älterer oder beeinträchtigter Menschen führen kann, aber auch für jeden anderen als komfortable Alternative zur haptischen Bedienung zu sehen ist, wird der Untersuchungsbereich dieser Masterarbeit im ersten Schritt explizit auf *Komfort und Unterhaltung* festgesetzt. Die zwei anderen Bereiche *Energieeffizienz* sowie *Sicherheit* sind für diese Masterarbeit daher nicht von Relevanz, weswegen darauf nicht weiter eingegangen wird. Auch das Thema Heimautomatisierung wird hier nicht weiter abgedeckt, das sich die Forschungsfrage mehr um eine selektive Kontaktaufnahme mit den Geräten, als um die Automatisierung dieser konzentriert.

Im Smart Home gibt es eine Vielzahl von Protokollen und Standards, um Verbraucher und Elektrogeräte miteinander zu vernetzen. Dies kann mit Feldbussystemen wie *KNX* oder Funktechnologien wie etwa *Bluetooth*, *ZigBee* oder *WLAN* geschehen (Sietmann, 2016). Vor allem im Küchen- und Wohnbereich integrieren sich aber verstärkt Einzelgeräte per Funk in das eigene Heimnetzwerk. So gibt es neben Fernsehern auch intelligente Wasserkocher, Kaffeemaschinen und Kühlschränke, die sich zusätzlich bequem per App bedienen lassen (Stabit, 2016).

Weiterhin öffnen die Herstellern verstärkt ihre Protokolle und APIs, um Entwicklern und Drittanbietern eine Möglichkeit zu bieten, die Geräte mit zusätzlichen Services erweitern zu können (Stabit, 2016). Ein Beispiel hierfür ist die *Smart Home Cloud API* von Samsung (2017), mit der die gesamte Samsung Smart Home Produktreihe mit fest definierten Webschnittstellen über das Internet oder das Heimnetzwerk angesprochen werden kann.

Das Smart Home bildet somit ein komplexes Zusammenspiel unterschiedlichster Technologien und Endgeräte. Dem gegenüber steht allerdings das Bedürfnis des Endnutzers, eine möglichst intuitive Steuerung des gesamten Systems zu haben. Natürliche Benutzerschnittstellen könnten dabei helfen, die Steuerung der eigenen vier Wände schnell erlernbar zu gestalten.

2.2 Natural User Interfaces

Wurden Computer in der ersten Generation noch mit Kommandozeilenbasierten Systemen (*CLI*) wie etwa MS-DOS bedient, die Codes in kryptischer Textform zum Umgang mit dem Computer nutzten, so hat sich in der zweiten Generation durch den Einzug der Desktop-Metapher schließlich die graphische Benutzeroberfläche (*GUI*) etabliert (Henseler, 2011).

Unter *Natural User Interfaces* versteht man gemäß Blake (2012) die nächste Generation an Benutzerschnittstellen. Man kann mit ihnen durch verschiedene Eingabemodalitäten wie etwa Multi-Touch, Motion-Tracking, Sprache oder Stylus interagieren. Als Beispiele nennt Bollhoefer et al. (2009) unter anderem die Touchscreen-Terminals an Bahnhöfen, Spracherkennung bei Callcenter-Services, Fingerabdruck- oder Iris-Scans bei Zugangskontrollen oder zukünftig die Systemsteuerung durch direkte Impulse der Synapsen im menschlichen Gehirn. Weiterhin gibt er folgende Definition:

> „Ein Natural User Interface (NUI) beschreibt ein Interface, welches unmittelbar durch einen oder mehrere Sinne des Benutzers bedient wird. Es bildet somit einen Oberbegriff für viele Arten der Interaktion an der Mensch-Maschine-Schnittstelle.“ (Bollhoefer et al., 2009, S. 6)

Legt man den Fokus jedoch auf die Kombination von Ein- und Ausgabe, welche man als natürlich erachtet, dann ergänzt sich die Palette zusätzlich um Eingabemodalitäten wie etwa Gesten und Körpersprache, Gesichtsaus-

druck und Blick, aber auch um Ausgabemodalitäten wie etwa Geruch, Berührung, sowie audiovisuellem Feedback (Jain et al., 2011).

Laut Jain et al. (2011) bildet eine Kombination aus Ein- und Ausgabe einen Modus und multimodale Erfahrung ist jene die sich entweder auf eine Kombination von mehr als einer Eingabe oder mehr als einer Ausgabe als Teil der natürlichen Interaktion bezieht. Die Interaktion des Menschen mit seiner Umwelt ist multimodal und eine multimodale Interaktion ist das, was eine natürliche Erfahrung ausmacht.

Dabei ist die Idee der multimodalen Interaktion aber bei Weitem nicht erst in den letzten Jahren entstanden. Bereits in den frühen 1980ern gab es Ansätze, verschiedene Eingabemodalitäten miteinander zu verbinden. Ein populäres Beispiel hierfür ist *Put-That-There*, das am Massachusetts Institute of Technology von Bolt (1980) entwickelt wurde.

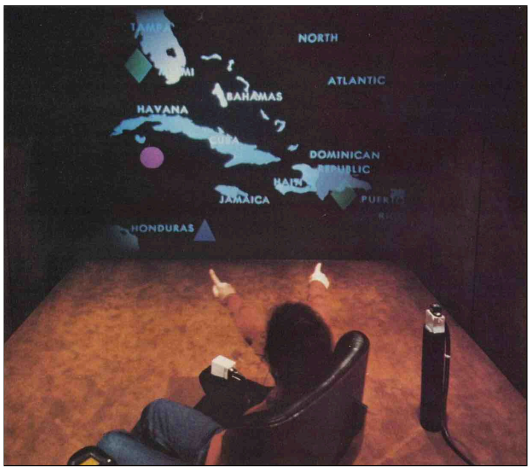


Abbildung 1: Put-That-There Bedienkonzept (Bolt, 1980)

In diesem Ansatz wird eine kontaktlose Interaktion durch die Kombination von Zeigegesten und Sprachsteuerung ermöglicht. Wie in Abbildung 1 zu erkennen ist, sitzt man vor einer Wand, worauf der Computer das Bild projiziert. Die Anwender können durch Sprachbefehle und Fingerzeige verschiedene geometrische Figuren erstellen, verändern und verschieben. Deuten sie beispielsweise auf ein Objekt und sagen gleichzeitig *„Move*

that to the right of the green square", so wird die identifizierte Figur rechts vom grünen Quadrat platziert. Zur aktiven Gestenerkennung musste man allerdings noch einen magnetischen Sensor am Handgelenk oder als Ring am Finger tragen, da das damals verwendete Trackingsystem *ROPAMS* (Remote Object Position Attitude Measurement System) auf elektromagnetischen Feldern basierte. Das bedeutet, dass der Anwender örtlich gebunden war und sich immer innerhalb eines gewissen Bereiches der Sensoren aufhalten musste.

Eine weitere, allerdings wesentlich spätere Arbeit umgeht das Problem zusätzlicher tragbarer Geräte, indem die Sprachsteuerung mit der Erkennung von Kopfhaltung kombiniert wird. Ausgehend der Beobachtungen von Vertegaal et al. (2001), die belegen, dass Personen, mit denen zu sprechen versucht wird, wahrscheinlicher angeschaut werden, als alle anderen im Raum, hat Darrell et al. (2002) das *Look-to-Talk*-Experiment durchgeführt. Er besagt, dass es daher natürlich ist zu glauben, dass die Verwendung von Kopfhaltung als Schnittstelle zur Aktivierung der automatischen Spracherkennung eine natürliche Mensch-Computer-Interaktion ermöglicht. Anhand drei unterschiedlicher Paradigmen wurde evaluiert, ob natürliche Interaktion die konventionellen Mittel zur Initiierung der Kommunikation mit interaktiven Agenten ersetzen kann. Diese sind:

- *Push-To-Talk (PTT)* - ein haptischer, Knopf-basierter Ansatz
- *Look-To-Talk (LTT)* - ein blickbasierter Ansatz
- *Talk-To-Talk (TTT)* - ein gesprochener, Keyword-basierter Ansatz

Dazu wurde im Experiment ein animierter Charakter namens *Sam* verwendet, der den Softwareagenten repräsentiert. Er bestand aus einfachen Formen, die ein Gesicht darstellten und hatte genau zwei mögliche Gesichtsausdrücke: zuhörend und abwesend.

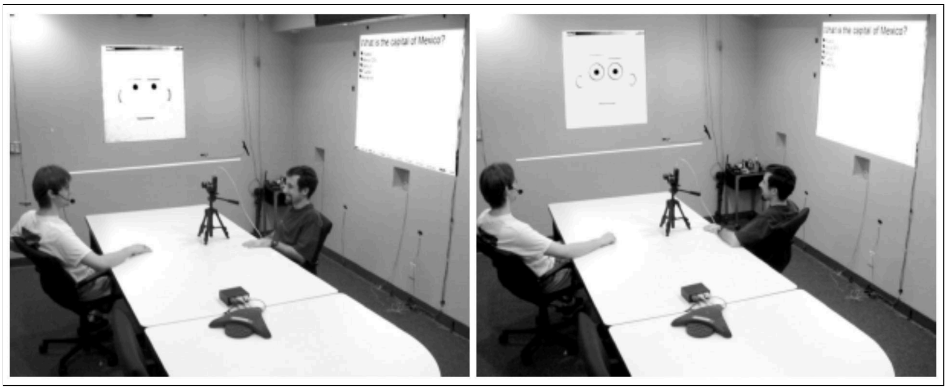


Abbildung 2: Agent-Dialog-Prototyp (Darrell et al., 2002)

Die Probanden sollten dann versuchen, den Agenten anhand der drei Paradigmen PTT, LTT und TTT anzusteuern, der anschließend über den integrierten Sprachsynthesizer einfache Fragen vorlas. In Abbildung 2 kann man erkennen, wie Sam seinen Gesichtsausdruck auf zuhörend ändert, sobald er erkennt, dass der Proband versucht mit ihm zu kommunizieren.

Das Ergebnis des Experiments veranschaulicht, dass sich im ersten Teil keine signifikanten Unterschiede zwischen allen drei Paradigmen ergaben und die Probanden überwiegend den Talk-To-Talk-Ansatz bevorzugten, was allerdings daran lag, dass dieser akkurater zur funktionieren schien, als die blickbasierte Variante. In einem zweiten Versuch unter idealeren Bedingungen (Wizard of Oz) gibt es allerdings einen signifikanten Unterschied zwischen Push-To-Talk und den anderen zwei Ansätzen. Dies zeigt, dass mit besseren Technologien sowohl LTT als auch TTT die bessere Wahl für die Mensch-Computer-Interaktion sind (Darrell et al., 2002).

Tabelle 2 zeigt die in dem Experiment verwendeten Paradigmen mit ihrer jeweiligen Aktivierung und Deaktivierung des Sprachinterfaces sowie deren Feedback auf. So wird beispielsweise die Sprachsteuerung aktiviert, sobald der Kopf in Richtung Softwareagenten gedreht wird und wieder deaktiviert, wenn man sich wieder weggedreht.

Modus	Activate Command	Active Feedback	Deactivate Command	Deactivate Feedback
PTT	Switch the microphone to „on“	Physical status of the switch	Switch the microphone to „mute“	Physical status of the switch
LTT	Turn head towards Sam	Sam shows listening expression	Turn head away from Sam	Sam shows normal expression
TTT	Say „computer“	Special beep	Automatic (after 5 sec)	None

Tabelle 2: Paradigmen der Sprachschnittstelle (Darrell et al., 2002)

Die Erkenntnisse von Darrell et al. (2002) werden somit als fundamentale Grundlage dieser Arbeit verwendet, da dieser Ansatz für eine kontaktlose Interaktion mit den Geräten im Smart Home von Interesse sein kann.

2.3 Natural User Interfaces im Smart Home

Durch die Kombination von Natural User Interfaces mit Produkten aus dem Smart Home eröffnen sich ganz neue Möglichkeiten für eine natürliche Benutzerinteraktion in den eigenen vier Wänden. Die Hersteller bieten zunehmend immer mehr Produkte mit beispielsweise einer komplett sprachbasierten Schnittstelle an, wie etwa *Google Now* auf Android, *Siri* auf Apple-Geräten oder *Alexa* auf dem *Amazon Echo* (Tucker et al., 2016).

Aber auch in der Literatur lassen sich bereits Ideen zur natürlichen Interaktion mit Heimgeräten durch reine Sprachbedienung finden. So stellt Potamitis et al. (2003) unter anderem ein integriertes System vor, das Sprache als eine natürliche Eingabemodalität verwendet, um einen benutzerfreundlichen Zugang zu Informations- und Unterhaltungsgeräten bereitzustellen. Es basiert auf einer Kombination von *Beamforming*, einem Verfahren zur Positionsbestimmung von Quellen in Schallfeldern durch Mikrofonarrays, sowie Spracherkennung.

In seiner Arbeit adressiert Potamitis et al. (2003) das Problem der freihändigen Spracherkennung in einem durch hausspezifische Nebengeräusche (z.B. Fernseher, Radio, Klimaanlage) gestörten Raum. Er verwendet Band-

breitenfilter zur adaptiven Geräuschunterdrückung, um eine unbeabsichtigte Aktivierung der Geräte zu vermeiden. In einer zweiten Stufe extrahiert er die Geräuschquellen anhand von Beamforming mit Mikrofonarrays in einem Winkel von $\pm 60°$. Der so gefilterte Frequenzbereich wird dann an einen *Word Spotter* weitergegeben, der nach spezifischen Schlüsselwörtern sucht. Beispielsweise wird er nur aktiv, wenn das Wort *Agent* im Satz vorkommt, wie etwa in *„Agent open the light please"*. Anschließend wird versucht, den Sprecher zu identifizieren, um bei Erfolg den Datenstrom an das Spracherkennungsmodul weiterzuleiten, was schließlich die Heimgeräte ansteuert. Abbildung 3 zeigt nochmals eine schematische Darstellung des Systems.

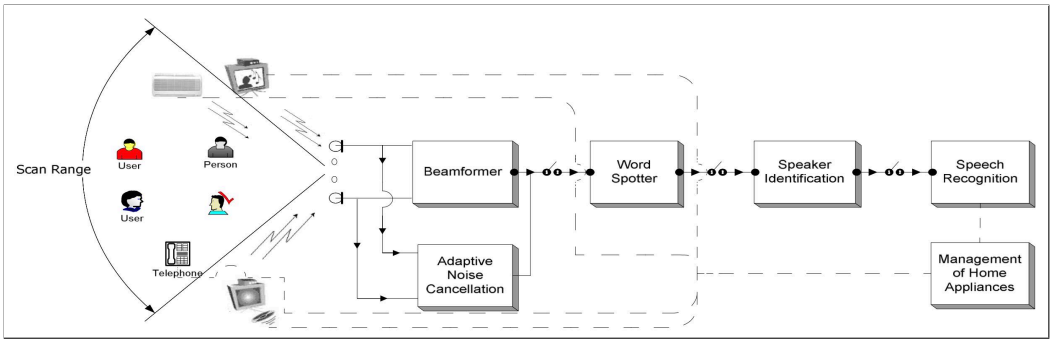


Abbildung 3: Schematisches Diagramm (Potamitis et al., 2003)

Allerdings erläutert Potamitis et al. (2003) zudem, dass diese Technologie auch physikalischen Grenzen unterliegt. So kann der Spracherkennungsalgorithmus des Systems das eigentliche Sprechsignal nicht von dem anderer gleichzeitig sprechender Personen oder Hintergrundmusik unterscheiden, falls die Signale zu sehr interferieren. Das würde beispielsweise passieren, wenn das Radio zu laut aufgedreht wäre.

Die anschließende Evaluation des integrierten Systems mit drei unterschiedlichen Szenarien zeigt ein vielversprechendes Ergebnis. Im ersten Szenario diskutieren vier Personen im Sitzen in einem Raum mit ruhigen Bedingungen, im Zweiten laufen drei Probanden währenddessen herum

und im letzten werden Nebengeräusche in Form von Musik hinzugefügt.

Anhand von Tabelle 3 ist deutlich zu erkennt, dass in allen drei Fällen die durchschnittlichen Erfüllungsquoten bei jeweils über 75 Prozent liegen. Beachtet man, dass im dritten Szenario Nebengeräusche hinzugefügt wurden, liegen die Differenzen zu Szenario 2 trotzdem nur bei rund 6-8 Prozent. Somit zeigt sich, dass die Verwendung von Bandbreitenfilter in Kombination mit Mikrofonarrays als ein durchaus interessanter Ansatz zur Extraktion von Sprachquellen gilt.

Scenarios	**Task Completion Rate (%)**		
	1st Trial	2nd Trial	3rd Trial
1st	96	100	-
2nd	85	92	96
3rd	79	84	90

Tabelle 3: Durchschn. Task Completion Rates (Potamitis et al., 2003)

Einen etwas anderen Ansatz zur natürlichen Interaktion mit Geräten im Smart Home verfolgt Kim et al. (2011) mit der vollständig gestenbasierten *Ambient Wall*. Diese Schnittstelle ermöglicht dem Benutzer seine Umgebung durch einfache Gesten zu bedienen und alle diesbezüglichen Informationen an einer zentralen Stelle anzeigen zu lassen.

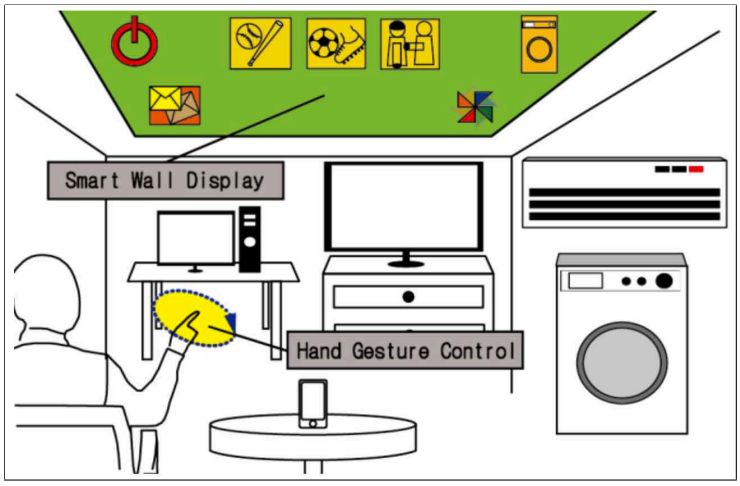


Abbildung 4: Ambient Wall (Kim et al., 2011)

Hierzu wird das Kontrollinterface nicht mehr an einem Bildschirm ausgegeben, sondern wie in Abbildung 4 zu sehen, mit einem Projektor an die Wand oder die Decke gestrahlt. Die Anwender können dann selbst entscheiden, was, wann und wo sie etwas auf dem großen Bild sehen möchten.

Durch ein kontextsensitives Menü wird dem Benutzer eine übersichtliche Oberfläche angeboten und man kann alle verbundenen Geräte bequem über eine Stelle bedienen, was bei Geräten, die in unterschiedlichen Räumen stehen, sicherlich eine bequeme Art und Weise bietet, diese anzusteuern. Die Gesten an sich werden mit einer Microsoft Kinect erfasst, was zusätzliche Hardware wie etwa Magnetarmbänder unnötig macht.

2.4 Natural User Interfaces im Smart Kitchen

Als *Smart Kitchen* bzw. *intelligente Küche* ist ein Unterbereich des Smart Home zu sehen, der Haushaltsgeräte wie Herd, Kühlschrank oder auch Kaffeemaschine vernetzt und teilweise automatisiert. Durch die Kommunikation miteinander entstehen immer mehr Szenarien, die den Benutzer in seiner alltäglichen Situation unterstützen können. Weiterhin kann dieser Bereich stark von Natural User Interfaces profitieren, da gerade in der Küche die Hände sehr oft durch Arbeitsprozesse blockiert sind.

Es existieren mittlerweile viele verschiedene Ansätze, die den Benutzer während des Koch- oder Backvorgangs unterstützen. So auch das blick- und sprachbasierte Kochbuch *eyeCOOK* von Bradbury et al. (2003). Mit seinem multimodalen Ansatz umgeht er das Problem, dass gerade für unerfahrene Anwender konventionelle Kochbücher eher eine Herausforderung darstellen, da sie stark vom eigentlichen Kochprozess ablenken.

Das interaktive Kochbuch lässt sich ganz einfach über Sprachbefehle und dem integrierten Eye-Tracker bedienen. Sobald ein Rezept aus der Datenbank ausgewählt wurde, wird es auf einem Monitor in zwei getrennten Ansichten abgebildet und die einzelnen Schritte farblich nach bereits abgeschlossenen und noch offenen Aufgaben markiert. Zudem kann man sich

Zutaten, Anweisungen, Begriffe, Nährwertangaben oder Küchenutensilien erklären und laut vorlesen lassen. Dazu schaut man den Begriff entweder an und spricht *„Define this“* oder man nennt den Begriff explizit, da man beim Kochen nicht immer in Reichweite des Eye-Trackers stehen kann. Weiterhin setzt das System automatisch Erinnerungen und Timer, sobald das Kochen begonnen wird (Bradbury et al., 2003).

Einen Schritt weiter geht allerdings Alexandersson et al. (2015) mit dem *Kochbot in the Intelligent Kitchen*, indem er die Technologien eine sprachbasierten Rezepte-App mit denen einer vernetzten Küche erweitert. Die so miteinander verbundenen Küchengeräte unterstützen den Koch, indem sie selbständig, entweder durch die App oder von Benutzerkommandos gesteuert, gleichzeitige oder zeitkritische Vorgänge abarbeiten. Dieser prototypische Ansatz ist vor allem für Kochanfänger oder Menschen mit körperlichen oder geistigen Einschränkungen von Vorteil.

Zu den Basisfunktionen der für Smartphones und Tablets verfügbaren App gehören unter anderem eine große Sammlung an Rezepten, das etappenweise Vorlesen der Kochanweisungen sowie das Beantworten von Fragen während dem Kochprozess. Durch die Erweiterung und Vernetzung der App mit einer Steuereinheit für das Smart Home können nun sukzessiv Küchengeräte wie etwa die Dunstabzugshaube, der Wasserhahn oder das Induktionsfeld des Herdes durch die App angesteuert werden. Dazu wurde die *Universal Remote Console (URC)* verwendet, das die Schnittstellen verschiedener Hersteller unterstützt. Der Fokus dieser Technologie ist die Integration von verschiedenen Geräten und Diensten um dem Endnutzer auf ihn zugeschnittene Benutzerschnittstellen zur Verfügung zu stellen.

Die Kochbot-App läuft als native Anwendung direkt auf den mobilen Endgeräten und die Spracherkennung geschieht via *Google API* über das Internet. Sollte das jedoch nicht zur Verfügbar stehen, wird auf ein lokales Spracherkennungsmodul zurückgegriffen, das allerdings nicht so zuverlässig funktioniert wie die Internetbasierte Variante. Dies ermöglicht zu-

sätzlich eine offline Verwendung, wenn das Rezept einmal vom öffentlich zugänglichen Volltextserver herunterladen wurde.

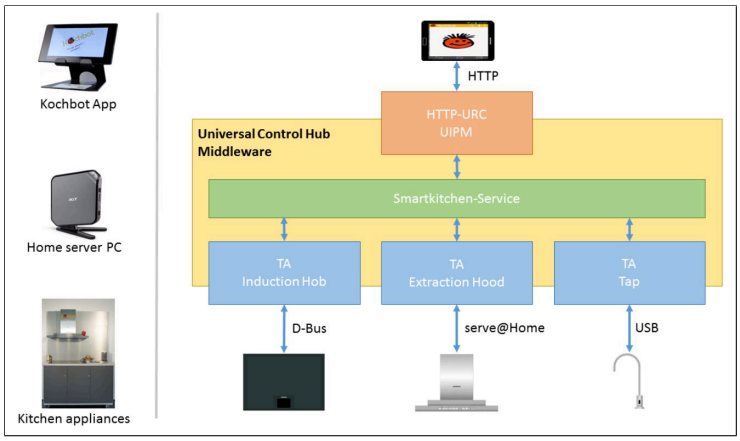


Abbildung 5: Kochbot Systemarchitektur (Alexandersson et al., 2015)

Die Middleware ist auf einem Windows-Rechner in einem Staufach unterhalb der Küche installiert und die Geräte sind jeweils entweder mit *USB*, *D-BUS* oder der *Siemens serve@Home*-Technologie über einen sogenannten *Target Adapter (TA)* an ihm angeschlossen, wie in Abbildung 5 dargestellt wird. Ein spezieller Service ist dann in der Lage, die rezeptspezifischen Anweisungen, wie etwa 800ml Wasser aus dem Wasserhahn zu lassen, in einen Satz von Einstellungen für die einzelnen Target Adapter zu übersetzen und umgekehrt. Die Konzeption und der modulare Aufbau des Prototypen von Alexandersson et al. (2015) sind somit eine wertvolle Grundlage für diese Masterarbeit.

3 Analyse und Konzeption

Der zu entwickelnde Prototyp soll eine intuitive Aktivierung natürlicher, sprachbasierter Mensch-Maschine-Kommunikation in pervasiven Umgebungen ermöglichen, ohne hierfür haptische Auslöser wie etwa Bedienknöpfe oder mobile Endgeräte einbeziehen zu müssen. Das hierfür verwendete Prinzip, das stark auf dem *Look-to-Talk*-Ansatz von Darrell et al. (2002) basiert, soll im Rahmen dieser Masterabeit prototypisch umgesetzt werden. Ausgehend der Anforderungen, die sich somit an dieses System stellen, werden in diesem Kapitel nun das Konzept, die Architektur sowie die zugrundeliegenden Funktionalitäten der prototypischen Umsetzung definiert.

3.1 Lösungsansatz

In einem zunehmend moderner werdenden Haushalt hält auch die natürliche Interaktion in Gestalt von Sprachsteuerung immer weiter Einzug. So ist der Anwender inzwischen daran gewöhnt, dass manche Geräte mit Sprachbefehlen wie beispielsweise *„Hey Siri“* oder *„OK Google“* ohne vorherigem Betätigen eines haptischen Auslösers bedient werden können. Das erlaubt den Bewohnern praktisch die Hände für andere Dinge frei zu haben, was gerade im Küchenumfeld sehr von Vorteil sein kann.

Überträgt man dieses Konzept nun direkt auf die Küchengeräte, könnte man sich ein Szenario vorstellen, indem die Interaktion mit dem Kühlschrank durch *„Hey Kühlschrank“* und die des Backofens mit *„Hey Backofen“* gestartet wird. Doch in einem Umfeld, in dem viele, vor allem aber laute Nebengeräusche produziert werden, greift diese Erkennung nicht oder nur bedingt, da sich die Schallwellen überlagern (Interferenzen). Ein solches Umfeld ist nunmal die Küche, wenn eben Geräte wie die Dunstabzugshaube, Kaffeemaschine, Wasserkocher oder Küchenradio Lärm erzeugen.

Natürlich kann man mit Ansätzen wie der Kombination von *Beamforming* mit Bandbreitenfilterung (Potamitis et al., 2003) die Sprachquellen identifizieren und dadurch eine bessere Erkennung erzielen, doch auch das funktioniert nur, wenn man einerseits das Equipment hat und andererseits die Störquellen nicht all zu laut sind. Läuft nebenbei noch das Radio, kann es außerdem passieren, dass der Gesang aus den Lautsprechern mit dem Gesagten des Anwenders verwechselt wird.

Eine mögliche Alternative hierzu wäre die Erkennung von Zeigegesten auf das Haushaltsgerät, ähnlich der bereits beschriebenen Funktionalität der *Ambient Wall* von Kim et al. (2011). Bedenkt man aber den Fall, dass man während des Kochens oder Backens auch nasse und tropfende Hände haben kann, ist dieser Ansatz wohl eher ungeeignet, da es durchaus passieren kann, dass die Küche dabei zusätzlich verdreckt wird.

Abbildung 6: Konzept (eigene Abbildung)

Daher wird in dieser Masterarbeit ein neues Konzept entwickelt, das sich stark am *Look-to-Talk*-Ansatz von Darrell et al. (2002) orientiert, indem anhand von Gesichtserkennung und Kopflagebestimmung überprüft wird, ob der Anwender in eine angeschlossene Kamera schaut (Abbildung 6). Wird er erkannt, werden zunächst die Gesichtserkennung deaktiviert und alle Haushaltsgeräte in einen Ruhemodus versetzt. Danach wird die eigent-

liche Sprachsteuerung gestartet, die erkannten Sprachbefehle ausgewertet und das Gerät über die bereitgestellte API angesteuert. Abschließend wird der Ursprungszustand der Geräte wiederhergestellt und die Gesichtserkennung wieder aktiviert. Um dies zu erreichen muss zunächst die Systemarchitektur definiert werden.

3.2 Systemarchitektur

> „A distributed system is a collection of independent computers that appears to its users as a single coherent system.“ (Tanenbaum & Steen, 2006, S. 2)

Wie schon Tanenbaum & Steen (2006) definiert haben, ist ein verteiltes System eine Sammlung unabhängiger Computer, die dem Anwender als ein einziges, zusammenhängendes System erscheint. Die einzelnen Haushaltsgeräte, die aus Sicht des Benutzers als voneinander unabhängig wahrgenommen werden, sollen jedoch später einen Geräteverbund bilden und miteinander kommunizieren, um sich gegenseitig Befehle, wie etwa dem Ruhemodus, schicken zu können. Ferner soll das System funktionsfähig bleiben, wenn neue Geräte hinzu oder bestehende entfernt werden. Aus diesem Grund bietet sich ein *verteiltes System* perfekt als Grundlage für die Softwarearchitektur dieser Masterarbeit an.

Weiterhin sollen die Haushaltsgeräte autark von einem zentralen Server betrieben werden, weshalb hier keine klassische *Client-Server*-Architektur in Frage kommt. Die einzelnen Geräte sollen sich ähnlich dem *Plug-and-Play*-Prinzip selbständig finden und kommunizieren können, sodass es keiner manuellen Konfiguration seitens der Benutzer bedarf.

Eine solche Funktion wird von den verschiedenen Herstellern von Haus aus nicht angeboten, weswegen diese über zusätzlich angebaute Komponenten angesteuert werden müssen. Die Software auf diesen Komponenten, auch *Middleware* genannt, steuert dann zusätzlich auch die Gesichtser-

kennung und Sprachsteuerung.

Nach der Literaturrecherche von Badica et al. (2013) sind die wichtigsten offenen Standards für die Entwicklung von Smart Home-Middleware:

- **Web Standards (WS)** - Eine Sammlung an Standards und standarisierter Praktiken für das Entwickeln von Webapplikationen. Es enthält einerseits *Web Service*-Standards aber auch Praktiken wie den *REpresentational State Transfer (REST)* Architekturstil. Badica et al. (2013) sieht diese Standards jedoch am sinnvollsten zur Anbindung der Smart Home Umgebung mit der Außenwelt.

- **Foundation for Intelligent Physical Agents (FIPA)** - Ein offener Standard für agentbasierte Komponenten-Middleware, die eine dynamische Unterstützung für das Agentenmanagement und standardisierte Interaktionsprotokolle bietet.

- **Open Services Gateway initiative (OSGi)** - Ein offener Standard für ein serviceorientiertes, dynamisches Komponentenmodell, das denn Lebenszyklus von Softwareentwicklung vereinfacht. Insbesondere bietet es eine erweiterte Unterstützung zum Installieren, Entfernen, Updaten, Starten und Stoppen von Softwarekomponenten in sogenannten *Bundles* an.

Somit sind schon drei verschiedene Plattformen für eine Middleware im Bereich Smart Home identifiziert worden. Web Services bieten vor allem eine einfache und standardisierte Grundlage an, um über das *Hypertext Transfer Protocol (HTTP)* zu kommunizieren, was gerade für eine Anbindung vom Heimnetzwerk nach außen nützlich ist. Mit *FIPA* werden insbesondere Lösungen und Protokolle angeboten, mit dem die Konzeption und Umsetzung von Softwareagenten bzw. agentenbasierten Komponenten und deren Verwaltung vereinfacht wird. Der Terminus Softwareagent bzw. Agent ist in der Literatur bisher allerdings nicht eindeutig definiert. Wooldridge (2002) beschreibt dies jedoch mit:

> „An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.“ (Wooldridge, 2002, S. 5)

Ein Softwareagent arbeitet demnach autonom in seiner Umgebung, um die ihm gesetzten Ziele zu erreichen, wonach das in dieser Arbeit zu entwickelnde System keinen Softwareagenten in diesem Sinn darstellt. Es handelt nicht autonom, sondern reagiert lediglich auf Ereignisse von außen.

Mit *OSGi* ist jedoch eine Plattform gegeben, die sich durch ihr serviceorientiertes und dynamisches Komponentenmodell besonders gut für die hier gegebene Problemstellung anbietet. Durch das integrierte Bundle-Konzept ermöglicht es das Installieren neuer, sowie das Entfernen alter Module, was gerade bei der Registrierung neuer Haushaltsgeräte von Vorteil ist.

3.2.1 OSGi

Die OSGi Alliance (2014) beschreibt das Java-basierte Framework als eine modulare Serviceplattform, die ein dynamisches Komponentenmodell in Form von sogenannten *Bundles* implementiert. Hierbei bildet ein Bundle eine modulare Einheit, die aus Java-Klassen und anderen Ressourcen besteht, die zusammen dem Anwender Funktionen bereitstellen. Weiterhin können sie Java-Pakete (Packages) zwischen einem *Exporter*-Bundle sowie einem *Importer*-Bundle teilen. Daraus ergeben sich zudem Abhängigkeiten untereinander, sodass Bundle B von Bundle A abhängig ist, wenn es beispielsweise die Pakete importieren möchte, welche Bundle A exportiert.

Die Funktionalität des Frameworks wird in verschiedene Schichten unterteilt, wie es Abbildung 7 genauer veranschaulicht. Die *Execution Environment* beschränkt die Ausführung der Bundles auf die deklarativ angegebene Java-Version (z.B. JavaSE-1.8). Die *Module*-Schicht definiert ein Modularisierungsmodell und den Austausch von Packages zwischen den Bundles. Die *Life Cycle*-Ebene ist für das Installieren, Entfernen sowie Starten

und Stoppen der Bundles verantwortlich. Der *Service*-Layer bietet schließlich ein dynamisches Servicemodell zum registrieren und exportieren von Diensten an das Framework an.

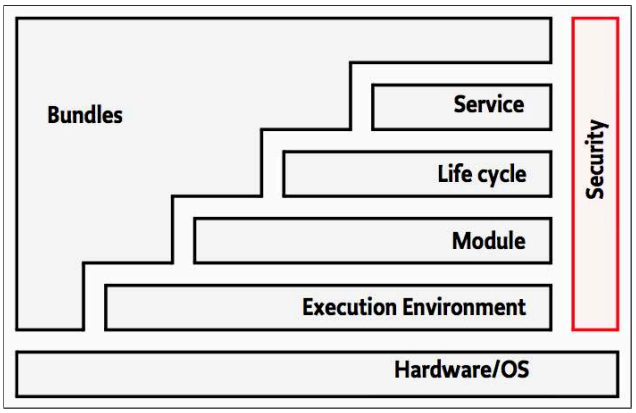


Abbildung 7: OSGi Ebenen (OSGi Alliance, 2014)

Die Bundles unterliegen einer genauen Versionierung und besitzen einen eigenen Lebenszyklus. Somit wird ermöglicht, dass zur Laufzeit mehrere Versionen des gleichen Bundles bereitgestellt werden können, was vor allem sehr nützlich ist, wenn man etwa Abhängigkeiten zu unterschiedlichen Versionen eines Frameworks hat.

Abbildung 8 zeigt die sechs Zustände, die ein Bundle annehmen kann. Befindet es sich im Zustand *INSTALLED*, so wurde das Bundle erfolgreich in der Laufzeitumgebung installiert und ist verfügbar. Der Zustand *RESOLVED* sagt aus, dass alle abhängigen Klassen vorhanden sind und das Bundle bereit ist, gestartet zu werden - oder es wurde gerade gestoppt. *STARTING* bedeutet, dass das Bundle gerade startet und es so lange in diesem Zustand verweilt, bis es aktiviert ist. Im *ACTIVE*-Zustand wurde das Bundle erfolgreich aktiviert. Befindet es sich im Zustand *UNINSTALLED*, dann wurde es deinstalliert und kann keine weiteren Zustände mehr annehmen (OSGi Alliance, 2014). Ferner kann man sich anhand sogenannter *BundleActivator* zudem in den Lebenszyklus der einzelnen Bundles einhängen und eigene Funktionen während dem Start und Stop ausführen lassen.

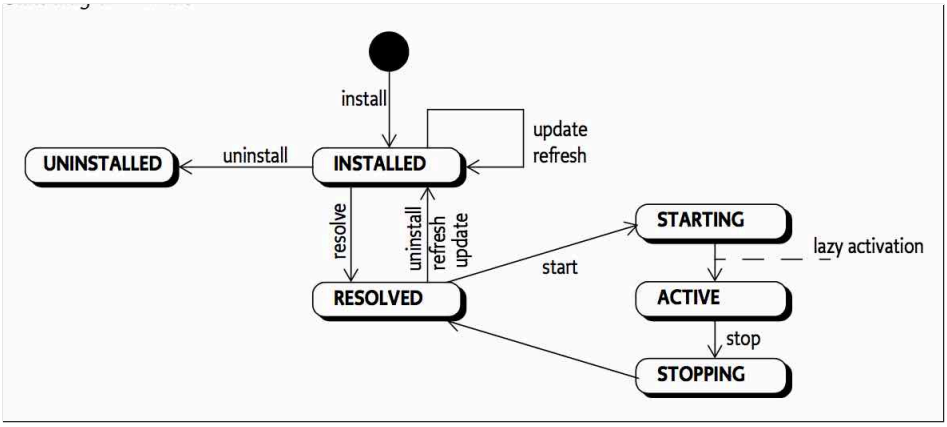


Abbildung 8: Zustandsdiagramm eines Bundles (OSGi Alliance, 2014)

Da OSGi durch seinen modularen Ansatz und dem dynamischen Komponentenmodell immer populärer wird, haben sich auch bereits andere Forscher dieser Thematik angenommen. Unter anderem haben Flotyński et al. (2013) das Framework im Kontext modularer Middleware für das Internet der Dinge näher analysiert. Neben dem bereits erwähnten modularen Bundle-Konzept wurden noch andere Mechanismen identifiziert, die für das Smart Home wichtig sind:

- **Laufzeitkonfiguration** - Die OSGi Runtime Configuration legt anhand gewisser Start Level fest, wann welches Bundle gestartet wird. Dabei werden alle Bundles berücksichtigt, die kleiner oder gleich der Runtime Configuration.

- **Bundlekonfiguration** - Die Konfigurationen in den Implementierungen der Bundles können durch den *Configuration Admin Service* in Konfigurationsdateien oder Datenbanken entkoppelt und so dynamisch zur Laufzeit angepasst werden.

- **Ereignisbasierte Kommunikation** - OSGi unterstützt eine flexible, ereignisbasierte Kommunikation zwischen den Bundles, die einen gemeinsamen Datenaustausch bereitstellt. Sie können einen Filter registrieren, der nur auf gewisse Nachrichten auf dem Bus anspringt, was

den Vorteil hat, dass man Abhängigkeiten vermeiden kann, da sich die Bundles nicht gegenseitig zu kennen brauchen.

- **Lokale Services** - Services sind Klassen, die ein Interface implementieren und können veröffentlicht, entdeckt und aufgerufen werden. Auf die Dienste wird synchron und lokal von derselben Maschine zugegriffen. Ähnlich der ereignisbasierten Kommunikation erfordert das Aufrufen von Diensten keine Verweise auf Bundles, die diese enthalten. Lediglich das Bundle mit der implementierten Schnittstelle muss bekannt sein. Angepasste Instanzen von Services werden von der ServiceFactory erstellt und im Framework zwischengespeichert. Bundles, die an bestimmten Diensten interessiert sind, können über entsprechende *ServiceEvents* im *ServiceListener* registriert, modifiziert und deregistriert werden.

- **Konsole** - Das OSGi Framework bietet eine integrierte Konsole an, mit der es möglich ist über SSH die einzelnen Bundles und ihre Services zu verwalten (z.B. starten und stoppen).

- **Web Services** - Anhand von Web Services können durch den *Remote Service Admin* lokale Services für andere OSGi-Laufzeitumgebungen exportiert werden und somit auch auf entfernten Maschinen angesprochen werden.

Zusätzlich der von Flotyński et al. (2013) aufgelisteten Mechanismen bietet die OSGi Serviceplattform noch eine weitere wichtige Funktionalität an: das *Hot Deployment*. Durch den modularen Aufbau und dem Lebenszyklus der Bundles können mit der integrierten Konsole ganz einfach neue Bundles über das Netzwerk installiert werden, ohne das bereits laufende System herunterfahren zu müssen (OSGi Alliance, 2014). Somit bildet das OSGi-Framework ein interessantes serviceorientiertes Konzept zur Modularisierung in Java und wird deswegen als Basisplattform für die prototypische Umsetzung in dieser Masterarbeit verwendet.

3.2.2 Laufzeitumgebungen

Die OSGi Serviceplattform findet zunehmend Verbreitung in kommerziellen und Open-Source Anwendungen. Auch große Konzerne wie IBM setzen mit ihrem WebSphere Applikationsserver mittlerweile auf diese Technologie (IBM, 2013). Dabei gibt es laut Hall et al. (2011) verschiedene Implementierungen dieser Spezifikation, unter anderem:

- Eclipse Equinox
- Apache Felix
- Knopflerfish

Eclipse Equinox ist der wohl am weitesten verbreitete Vertreter der OSGi Implementierungen, da er die Grundlage für die gleichnamige Entwicklungsumgebung bildet und durch die *Eclipse Rich Client Platform* in vielen weiteren Desktop-Applikationen verbreitet ist. *Apache Felix* ist ein Toplevel-Projekt der *Apache Software Foundation* und dient unter anderem als Basis für den *GlassFish* Applikationsserver (Sahoo, 2012).

Eine weitere Implementierung der OSGi Spezifikation is *Concierge*, das als Projekt an der Eidgenössische Technische Hochschule Zürich (ETHZ) entstand und als OSGi-Implementierung für Geräte mit beschränkten Ressourcen geplant und daher für mobile sowie eingebettete Geräte optimiert ist (Rellermeyer & Alonso, 2007). Mittlerweile wurde es der Eclipse Foundation übertragen, befindet sich aber noch immer in der Entwicklung.

Ein Laufzeitcontainer, der die beiden Implementierungen *Eclipse Equinox* und *Apache Felix* unterstützt, ist *Apache Karaf*. Er ist besonders interessant, da er zusätzlich ein integriertes Provisioning System besitzt, das auf dem Buildtool *Apache Maven* basiert und abhängige Bundles selbständig aus dem Maven-Repository beziehen kann (Apache Software Foundation, 2014). Da sich *Concierge* noch in der Entwicklung befindet und *Apache Karaf* zusätzliche Werkzeuge für das Deployment und Management der Bundles bereit-

stellt, wird *Apache Karaf* als Laufzeitcontainer und *Apache Felix* als OSGi-Laufzeitumgebung für die prototypische Umsetzung gewählt.

3.2.3 Apache Aries Blueprint

Mit *Apache Aries Blueprint* wird ein Dependency Injection-Framework für OSGi bereitgestellt, mit dessen Hilfe es möglich ist, reine Java Objekte sowie OSGi-Services deklarativ über eine XML-Datei zu registrieren und exportieren zu lassen (Apache Software Foundation, 2017). Ein Bundle wird dann als Blueprint-Bundle gesehen, wenn es ein oder mehrere Blueprint-XML-Dateien in einem speziell vorgesehenen Ordner (OSGI-INF/blueprint) enthält. Wird ein Bundle als solches erkannt, wird ein spezieller Blueprint-Container erstellt, der danach folgende Mechanik ausführt:

1. Parsen der Blueprint-XML-Datei
2. Instanziieren der Komponenten
3. Verdrahten der Komponenten miteinander
4. Registrieren der Dienste
5. Service-Referenzen nachschlagen

Während der Initialisierung stellt der Blueprint-Container sicher, dass obligatorische Service-Referenzen erfüllt sind, registriert alle Services in die Service-Registry und erstellt erste Komponenteninstanzen. Der Blueprint-Container und die Dienste werden dann zerstört, wenn das Bundle gestoppt wird (Apache Software Foundation, 2017). Der Vorteil von Apache Aries Blueprint ist, dass man die Services nicht mehr im BundleActivator erstellen muss, sondern sie von außen in den OSGi-Lebenszyklus injizieren lassen und alle Services an einer zentralen Stelle verwalten kann.

3.2.4 Device Discovery

Da es möglich sein soll, dass sich die Haushaltsgeräte gegenseitig im Netzwerk ohne vorherige Konfiguration durch den Anwender selbständig fin-

den können, benötigt die Middleware eine Komponente, die eine automatische Geräteerkennung (*Device Discovery*) im Netzwerk anbietet. Eine gängige Definition dafür lautet:

> „Discovery is the process by which an entity on a network (a client) is spontaneously notified of the availability of desirable services or devices on the network (resources). More precisely, discovery is a mechanism for dynamically referencing a resource on the network. These references are handles or other information that the client can subsequently use to contact the resource.“ (Edwards, 2006, S. 70)

Edwards (2006) beschreibt Discovery demnach als einen Prozess, durch den Entitäten in einem Netzwerk spontan über die Verfügbarkeit gewünschter Dienste oder Geräte (Ressourcen) informiert werden. Durch Verweise sind sie später in der Lage, diese Ressourcen zu kontaktieren. Gängige Technologien hierfür sind unter anderem:

- Service Location Protocol (SLP)
- Apache Jini / Apache River
- Apple Bonjour / multicast DNS

Für das *Service Location Protocol* wurde mit *jSLP* von Rellermeyer (2005) eine Java-Implementierung geschrieben, die durch ihren geringen Speicherbedarf besonders für kleine und eingebettete Geräte geeignet ist. Es kann in einem unmanaged Modus ohne zusätzliche Verwendung eines dedizierten Lookup-Servers betrieben werden.

Apache Jini, das mittlerweile durch *Apache River* ersetzt wurde, ist ein rein Java-basiertes Framework, das eine Infrastruktur anbietet, mit der Dienste im Netzwerk bereitgestellt werden können, die sich an Veränderungen anpassen. So können beispielsweise Clients und Dienste ohne gemeinsames Wissen über das Transportprotokoll kommunizieren. Allerdings kommt

dieses Framework für diese Arbeit nicht in Frage, da es zum Auffinden der Dienste einen zentralen Lookup-Server voraussetzt (Edwards, 2006).

Mit *JmDNS* gibt es eine Java-Implementierung, die zur Registrierung sowie dem Auffinden von Diensten in lokalen Netzwerken verwendet werden kann und komplett kompatibel zu Apple's *Bonjour* ist, da es auf dem *multicast DNS*-Prinzip basiert (van Hoff, 2002). Da es weitestgehend auf einen zentralen Lookup- und Registrierungsmechanismus verzichtet, ist dies ein geeigneter Ansatz für diese Arbeit.

Die Entscheidung, welches Framework nun in der Umsetzung verwendet wird, fiel zunächst auf *jSLP*. In den ersten Tests stellte sich allerdings schnell heraus, dass die Geräte nur in eine Richtung erkannt wurden. Somit wird für die Umsetzung schließlich *JmDNS* verwendet, da es sich wesentlich stabiler verhielt und sich alle Geräte gegenseitig erkannt haben.

3.2.5 Remote Connections

Damit die Haushaltsgeräte schließlich miteinander Kommunizieren können, müssen sie einen Verbindungskanal zueinander aufbauen, um die Informationen austauschen zu können. Hierzu gibt es verschiedene Ansätze und Möglichkeiten, die in diesem Kapitel kurz angeschnitten werden.

Ein möglicher Ansatz wäre das Bereitstellen von *Web Services*, wie es Badica et al. (2013) bereits erwähnt hat. Dies setzt allerdings einen Webserver voraus, der permanent auf eingehende Verbindungen horcht. Zudem muss eine Adressierung implementiert werden, die mit Mechanismen wie etwa *SOAP* oder *REST* aber relativ schnell umgesetzt werden kann.

Eine weitaus leichtgewichtigerer Ansatz ist die Verwendung von OSGi-Bundles, die über TCP mit den Laufzeitumgebungen anderer Maschinen kommunizieren können. Eine solche Implementierung bietet Rellermeyer et al. (2007) mit *R-OSGi* an. Abbildung 9 zeigt einen groben Überblick über den Architekturaufbau von *R-OSGi*.

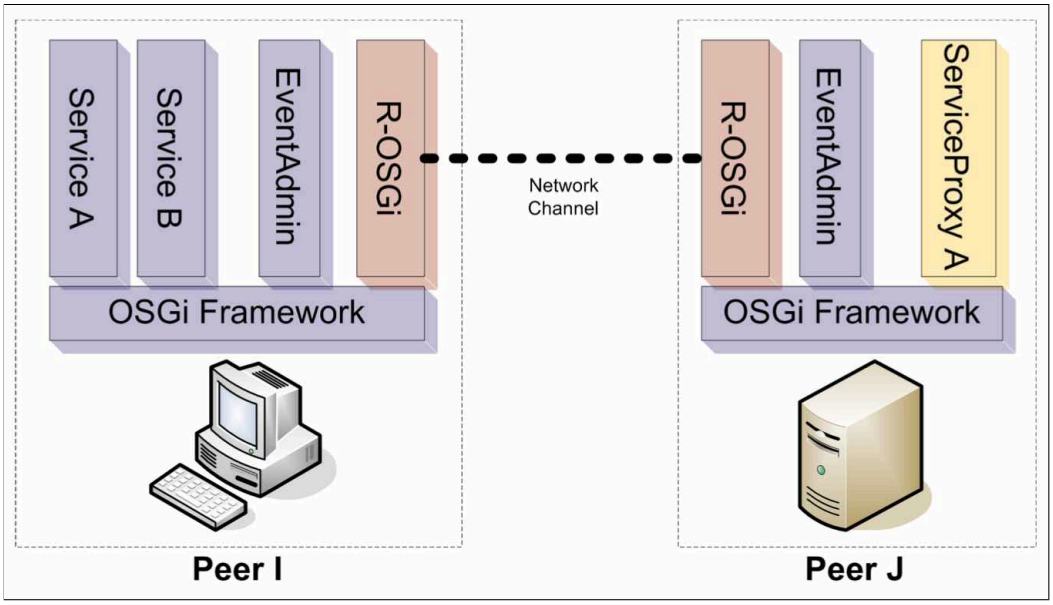


Abbildung 9: Übersicht der R-OSGi Architektur (Rellermeyer et al., 2007)

R-OSGi wird als eigenes Bundle in der lokalen OSGi-Laufzeitumgebung installiert und kann als *Remote Service* deklarierte Dienste einer entfernten Laufzeitumgebung über einen Proxy-Mechanismus in der lokalen Umgebung bereitstellen. Diese Remote Services können dann genau wie lokale Services auch über die eigene Service-Registry aufgerufen werden. Zusätzlich ermöglicht R-OSGi auch das Erweitern der ereignisbasierten Kommunikation, indem es die Events an die entfernte Laufzeitumgebung über das Proxy-Bundle weiterleitet.

Dieser leichtgewichtige Mechanismus, der besonders für eingebettete Geräte im Umfeld von OSGi entwickelt wurde, wird daher für den Aufbau der entfernten Verbindung im Prototyp verwendet.

3.3 Hardware

Da die Haushaltsgeräte später mit technischen Komponenten erweitert werden sollen, muss man sich zunächst Gedanken machen, welche Hardware für die gegebene Problemstellung überhaupt in Frage kommt. Mittlerweile gibt es viele verschiedene Kleincomputer und Platinen auf dem Markt, die

wegen ihrer geringen Abmessungen ideal zur Erweiterung dienen können.

Maksimović et al. (2014) hat verschiedene Hardware untersucht, die für eine Anwendung im Gebiet des Internet der Dinge und somit auch dem Smart Home in Frage kommen. Dazu wurden fünf verschiedene Einplatinenrechner miteinander verglichen:

Name	**Board operating system**	**Programming language**
Raspberry Pi	Raspbian, Ubuntu, Android, Arch-Linux, FreeBSD, Fedora, RISC OS	C, C++, Java, Python
Arduino	/	Arduino
BeagleBone Black	Linux Angstrom	Arduino
Phidgets	Linux	Visual Basic, VB.NET, C#, C/C++, Flash/Flex, Java, Labview, Matlab, ActionScript 3.0, Cocoa
Udoo	Ubuntu, Android, Linux, ArchLinux	Arduino, C, C++, Java

Tabelle 4: Betriebssysteme und Programmiersprachen der Einplatinenrechner (Maksimović et al., 2014)

Da wegen der OSGi Serviceplattform eine Java-Umgebung vorausgesetzt wird, fallen das *Arduino* sowie das *BeagleBone Black* aus dem Rahmen, da sie Java nicht unterstützen. Zusammenfassend erläutert Maksimović et al. (2014) zwar, dass das *Udoo*-Board die beste Leistung unter allen getesteten Einplatinenrechnern erzielt hat, es allerdings auch einen sehr hohen Marktpreis besitzt. Auf der anderen Seite hat die Detailanalyse des *Raspberry Pi* gezeigt, dass es zwar ein sehr günstiges Board ist, aber die Vorteile eines PCs in die Domäne von Sensor-Netzwerken bringt, was es zu einer perfekten Plattform für die Anbindung einer Vielzahl von externen Peripheriegeräten macht.

Gleichzeitig besitzen die meisten Geräte die Möglichkeit, ein Linux als Betriebssystem zu installieren, weshalb man sie wie herkömmliche Rechner konfigurieren und betreiben kann. Die Untersuchung ist zum Stand dieser Masterarbeit allerdings schon zwei Jahre alt, weshalb zwischenzeitlich auch neuere Modelle auf dem Markt erschienen sind. Basierend auf

den Erkenntnissen von Maksimović et al. (2014) wird jedoch für die Umsetzung des Prototypen das *Raspberry Pi 3 Modell B* als Einplatinenrechner verwendet, da es im Gegensatz zu seinen Vorgängern ein integriertes WiFi- sowie Bluetooth-Modul und auch einen wesentlich schnelleren Prozessor besitzt. Als Betriebssystem kommt das weit verbreitete *Raspbian Jessie* zum Einsatz, das speziell an das *Raspberry Pi* angepasst wurde.

3.3.1 Steuergeräte

Wie schon erwähnt, wird für die Steuergeräte das *Raspberry Pi 3 Modell B* verwendet. Doch es wird für die jeweiligen Teilaufgaben des Steuergerätes noch weitere Hardware benötigt:

- **Webcam** - Zur Gesichtserkennung muss eine Kamera oder Webcam permanent die Umgebung filmen.
- **Mikrofon** - Zur Spracherkennung müssen die gesprochenen Schallwellen über ein Mikrofon als Stream aufgezeichnet werden.
- **Lautsprecher** - Zur synthetischen Sprachausgabe wird ein Lautsprecher benötigt, der die erzeugten Sätze wiedergibt.

Als Kamera wird eine *Logitech C920* verwendet, da sie hochauflösende Bilder im Format 1920x1080 Pixeln aufzeichnet und einen integrierten Prozessor besitzt, der die Frames im H.264-Format vorberechnen kann. Gleichzeitig enthält sie ein eingebautes Mikrofon, das eine integrierte Rauschunterdrückung besitzt, weshalb hier keine extra Hardware mehr beschafft werden muss.

Da die Kamera zudem auch über eine blau leuchtende LED verfügt, wird diese als zusätzlicher visueller Indikator für eine aktive Gesichtserkennung genutzt. Zur Sprachausgabe wird ein einfacher Lautsprecher verwendet, der an die integrierte Soundkarte des *Raspberry Pi* angeschlossen wird. Der

komplette Hardwareaufbau wird in Abbildung 10 dargestellt.

Abbildung 10: Hardware des Prototypen (eigene Abbildung)

Der Vorteil dieser Hardware liegt in seinen geringen Abmessungen und dem verhältnismäßig günstigen Anschaffungspreis. Man könnte den Einplatinenrechner und den Lautsprecher so hinter dem Haushaltsgerät verstecken, dass nur noch die Kamera sichtbar ist und das System somit kaum noch auffällt.

3.3.2 Haushaltsgeräte

Da zum Zeitpunkt dieser Masterarbeit keine realen Smart Home Geräte zur Verfügung standen, wurden Eigenimplementierungen verwendet, welche die Funktionalität herkömmlicher Haushaltsgeräte imitieren.

Angesprochen werden sie durch einen Webservice, weshalb die Steuergeräte über HTTP mit den Haushaltsgerätimitaten kommunizieren können. Dieser Ansatz ist stellvertretend für alle Smart Home Geräte, die mit festdefinierten Webservices wie beispielsweise der *Smart Home Cloud API* von Samsung (2017) angesprochen werden.

Als Hardware wird wieder ein *Raspberry Pi* mit angeschlossenem Lautsprecher verwendet. So gibt es unter anderem einen Musikplayer, der zwischen den Liedern wechseln kann, ein Dunstabzugshaubenimitat, das die Lautstärke des Absauggeräusches in drei Stufen reguliert und eine Kaffeemaschine, welche die Geräusche des Mahl- und Brühvorganges eines Kaffeevollautomaten wiedergibt. Die Projekte und deren Quelltexte lassen sich im Ordner *Appliances* auf dem beigefügten Datenträger entnehmen.

3.4 Eingabe- und Ausgabemodalitäten

> „Wenn zur Nutzereingabe oder Systemausgabe gleichzeitig und kombiniert mehrere Wege der Kommunikation genutzt werden, spricht man von einem multimodalen Medium.“ (Malaka et al., 2009, S. 259)

Im Fokus dieser Masterarbeit steht die Konzeption eines multimodalen Systems, das eine intuitive Kontaktaufnahme vernetzter Haushaltsgeräte ermöglicht, indem die Eingabemodalitäten Gesichtserkennung und Sprachsteuerung miteinander kombiniert werden. In solch einem Fall spricht man auch von *multimodaler Fusion*. Aber auch zur Systemausgabe werden mehrere Modalitäten miteinander verbunden, was dann als *multimodale Fission* bezeichnet wird.

Die nachfolgende Abbildung 11 stellt das multimodale Prinzip des Konzeptes genauer dar. So wird ähnlich dem Talk-To-Talk-Ansatz von Darrell et al. (2002) das Abspielen von Pieptönen als Indikator für die Spracheingabe verwendet und das Erlöschen der Kamera-LED zeigt an, dass ein Gesicht erkannt wurde. Schließlich gibt ein Synthesizer noch Sprache als natürliche Form der Kommunikation zurück.

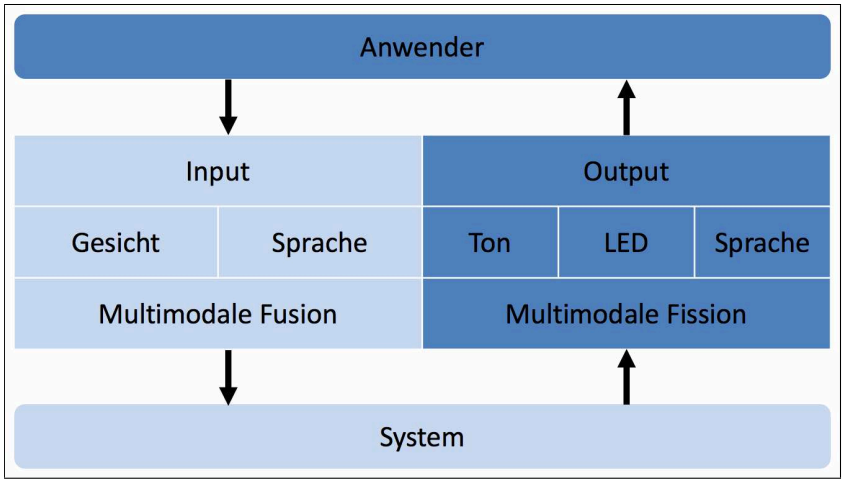


Abbildung 11: Multimodales Prinzip des Systems (eigene Abbildung)

Für die prototypische Umsetzung müssen daher bereits existierende Technologien und Ansätze für die einzelnen Modalitäten betrachtet werden, die in den nachfolgenden Unterkapiteln näher erklärt werden.

3.4.1 Sprachausgabe

Für die Ausgabe künstlicher Sprache anhand von geschriebenem Text benötigt man ein sogenanntes *Text-To-Speech*-System (TTS), dessen Ziel darin besteht, aus Texten automatische Sprachsignale in Form akustischer Wellen zu erzeugen (Klatt, 1987). Dabei unterscheidet man nach Rashad et al. (2010) zwei Phasen:

- **Natural Language Processing**: Analyse und übertragen des Eingabetextes in eine phonetische Darstellungsform und Berechnung der Prosodie (Eigenschaften wie Betonung und Sprechpausen).
- **Signal Processing**: Erzeugung akustischer Sprachwellen anhand der phonetischen und prosodischen Information, auch als *Sprachsynthese* bezeichnet.

Es gibt drei Hauptansätze für die Sprachsynthese: artikulatorische, konkatenative und Formant-Synthese. Artikulatorische Synthese erzeugt Spra-

che durch direkte Modellierung des Verhaltens der menschlichen Artikulation, Formant-Synthese-Modelle anhand der Polfrequenzen von Sprachsignalen. Formanten sind die Resonanzfrequenzen des Stimmtraktes. Da die Formanten die Hauptfrequenzen bilden, die Klänge unterscheiden, wird Sprache unter Verwendung dieser geschätzten Frequenzen synthetisiert. Die konkatenative Sprachsynthese erzeugt die Sprache durch Verketten von kleinen, voraufgezeichneten Spracheinheiten, wie etwa Phonemen und Diphonen (Rashad et al., 2010).

Abbildung 12 stellt beispielsweise die generelle Arbeitsweise von konkatenativen TTS-Systemen schematisch dar. So wird der Text zuerst normalisiert, dann die Phonetik analysiert und die Prosodie erzeugt. Beides wird dann als Eingabe an die Sprachsynthese weitergegeben, die anhand von *Prosody Matching* schließlich die Wortketten in ein Audiosignal überführt.

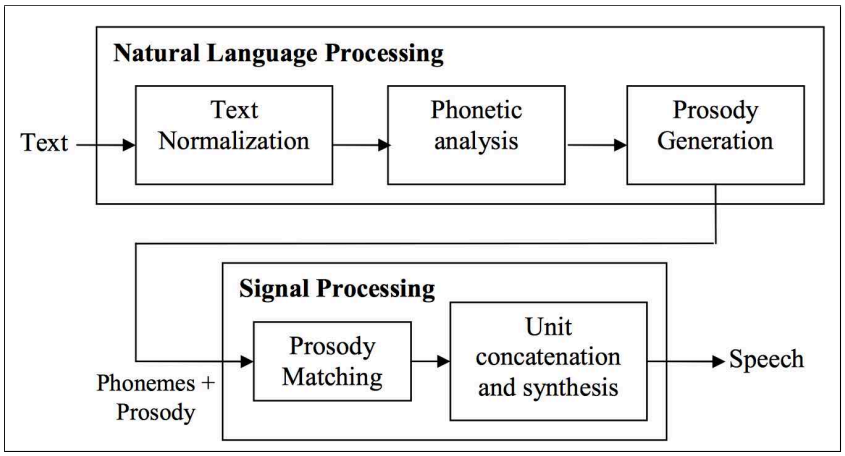


Abbildung 12: Konkatenativer TTS Ansatz (Rashad et al., 2010)

Als konkatenative TTS-Systeme werden *MaryTTS* von Schröder & Trouvain (2003) sowie *freeTTS* von Walker et al. (2002) für diese Arbeit in Betracht gezogen, da beide in der Programmiersprache Java entwickelt wurden. Während der Machbarkeitsanalyse stellte sich schnell heraus, dass *freeTTS* eine sehr abgehakte und metallische Stimme erzeugt. Da die Betonung der Stimme in *MaryTTS* subjektiv angenehmer wahrgenommen wird

und das System zusätzlich die deutsche Sprache unterstützt, wird *MaryTTS* für die Sprachausgabe im Prototyp eingesetzt.

3.4.2 Spracherkennung

> „Natürliche Sprache ist sicher das Kommunikationsmedium, das Menschen am häufigsten nutzen, wenn sie mit anderen Menschen Informationen austauschen [...] Es liegt also nahe, menschliche Sprache auch in Computersystemen zu nutzen.“ (Malaka et al., 2009, S. 265)

Nach Malaka et al. (2009) wird Spracherkennung benötigt, wenn man gesprochene Sprache mit dem Computer analysieren und verstehen will. Dazu muss ein Audiosignal analysiert werden und Phoneme sowie Wörter müssen erkannt werden, so dass dabei ganze Sätze entstehen. Gaida et al. (2014) hat dazu eine umfangreiche Evaluation von Open-Source-Toolkits durchgeführt, mit denen eine automatische Spracherkennung umgesetzt werden kann. Er vergleicht dabei folgende Implementierungen:

- **Hidden Markov Model Toolkit (HTK)**
  - **HDecode v3.4.1**
  - **Julius v4.3**
- **Sphinx Toolkit**
  - **pocketSphinx v0.8**
  - **Sphinx-4**
- **Kaldi**

Die Frameworks wurden vor allem in Bezug auf Usability und Aufwand der Erkennungsgenauigkeit verglichen und die Evaluation erfolgte in deutscher und englischer Sprache mit dem Verbmobil 1 bzw. dem Wall Street Journal 1 Korpus. Zusammenfassend kam heraus, dass *Kaldi* herausragende Ergebnisse erzielt, das Sphinx-Toolkit aber auch gute Ergebnisse in kurzer Zeit liefert. HTK bietet die geringste Unterstützung und erfordert viel

mehr Zeit, Kenntnisse und Anstrengungen, um die gleichen Ergebnisse wie die anderen zwei zu erzielen.

Da das Sphinx-Toolkit im Gegensatz zu Kaldi eine Java-Schnittstelle anbietet, wird *poketSphinx* als Spracherkenner für die Umsetzung verwendet. Sphinx-4 ist zwar komplett in Java programmiert, doch Huggins-Daines et al. (2006) hat gezeigt, dass pocketSphinx auf einem tragbaren Gerät mit 206 MHz im Schnitt 8 mal schneller als das Grundsystem arbeitet und somit viel performanter ist.

Als Sprachkorpus und Akustikmodell kommt der frei verfügbare Korpus für deutsche Spracherkennung von Radeck-Arneth et al. (2015) zum Einsatz, der an der TU Darmstadt entstanden ist. Es wurde in einer kontrollierten Umgebung mit drei verschiedenen Mikrofonen im Abstand von einem Meter aufgenommen und umfasst 180 verschiedene Sprecher mit insgesamt 36 Stunden Audio-Aufnahmen (Radeck-Arneth et al., 2015).

Dieser Korpus enthält eine Wörterbuchdatei, die Wörter mit ihren phonologische Repräsentationen registriert, wie nachfolgend exemplarisch aufgelistet ist. Diese können verwendet werden, um anschließend eine Grammatik zu definieren. Dabei können jedoch nur die Wörter erkannt werden, die zuvor in der Wörterbuchdatei registriert wurden.

```
1 abbruch      a p b r uu x
2 abspielen    a p ss p i: l @ n
3 ja           j aa:
4 nein         n ai n
5 kaffee       k aa: f @
6 stopp        ss t oo p
7 weiter       v ai t ei
8 zurück       t s u: r yy k
```

Algorithmus 1: Beispielhafte Wörterbuchdatei (Radeck-Arneth et al., 2015)

3.4.3 Gesichtserkennung

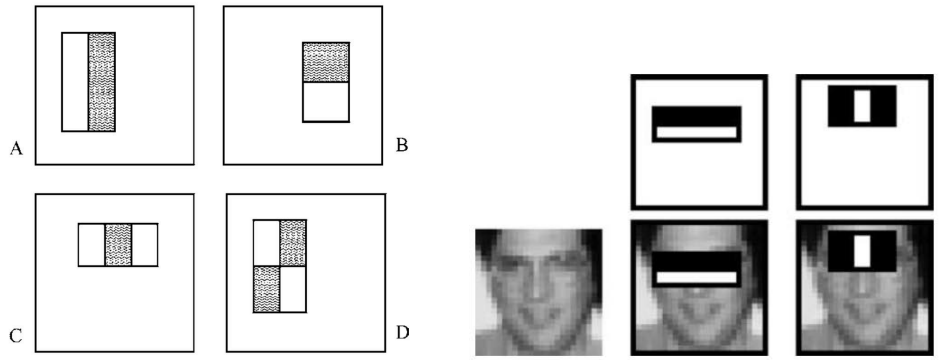
Die Bezeichnung *Gesichtserkennung* wird in der deutschen Sprache mehrdeutig verwendet. Einerseits versteht man darunter das Auffinden eines gesichtsähnlichen Objektes, im Englischen als *Face Detection* bezeichnet, auf der anderen Seite die Identifizierung eines bestimmten Gesichts, auch *Face Recognition* genannt (Zhao et al., 2003).

Diese Masterarbeit verwendet den Begriff *Gesichtserkennung* im Sinne des Auffinden von Gesichtern in digitalen Bildern. Der Prototyp nimmt keine Klassifizierung verschiedener Anwender im Sinne der *Face Recognition* vor, womit es durchaus passieren kann, dass die Geräte ungewollt aktiv werden, wenn beispielsweise während dem Kochen das Kind ständig die Kaffeemaschine anschaut.

Ein weit verbreitetes Verfahren zur Erkennung von Gesichtern in Echtzeit ist die sogenannte *Viola-Jones-Methode*, die Bilder sehr schnell verarbeiten kann und dabei gleichzeitig sehr hohe Erkennungsraten erzielt. Anhand eines sogenannten *Integralbildes* können die für die Erkennung benötigten Haar-Features schnell berechnet werden. Ein einfacher, aber effizienter Klassifizierer wird dann anschließend durch den *AdaBoost* Lernalgorithmus von Freund & Schapire (1995) trainiert, der eine kleine Anzahl kritischer visueller Features aus einer sehr großen Menge potentieller Features auswählt. Durch die Verkettung mehrerer solcher Klassifizierer wird erreicht, dass Hintergrundbereiche des Bildes schnell verworfen werden, während auf gesichtsähnlichen Regionen mehr Berechnungen angewendet werden können (Viola & Jones, 2004).

Haar-Features sind rechteckige Merkmale, die einen einzelnen Wert repräsentieren, der sich durch die Subtraktion der Summen von Pixeln in weißen und schwarzen Bereichen des Untersuchungsfensters bildet (Viola & Jones, 2004). in der nachfolgenden Abbildung 13a werden die unterschiedlichen Typen der verwendeten Haar-Features dargestellt. In (A) und

(B) werden Zwei-Rechteck-Features gezeigt, die beispielsweise zur Kantenerkennung benutzt werden. Für die Erkennung von Linien kommen Drei-Rechteck-Features (C) zum Einsatz und (D) zeigt ein Feature, das aus vier Rechtecken besteht.



(a) Haar Features **(b)** Anwendung der Features

Abbildung 13: Viola-Jones-Methode (Viola & Jones, 2004)

Das Bild wird dann typischerweise herunterskaliert und die Haar-Features in den jeweiligen *Subwindows* über das Bild gelegt. Abbildung 13b zeigt das erste und zweite Feature, das vom AdaBoost-Lernalgorithmus ausgewählt wurde. Das erste Feature misst den Intensitätsunterschied im Bereich zwischen den Augen und oberen Wangen, da es auf der Annahme beruht, dass der Bereich um die Augen oft dunkler ist als die der Wangen. Das zweite Feature vergleicht die Intensitäten im Augenbereichen mit der Intensität über der Nase (Viola & Jones, 2004).

Dieser Ansatz wird auch standardmäßig von Frameworks wie beispielsweise *OpenCV* bereitgestellt. *OpenCV* ist eine quelloffene Bibliothek für Bildverarbeitung und *Computer Vision*. Es wurde speziell für effiziente Rechenleistung und mit dem Fokus auf Echtzeit-Anwendungen entwickelt. Da es in optimiertem C/C++ geschrieben ist, kann die Bibliothek die Vorteile des Multi Core Processing nutzen (OpenCV, 2017). Das ist gerade für Geräte mit beschränkten Ressourcen, wie etwa dem *Raspberry Pi*, von Vorteil.

Weiterhin bietet das Framework eine Schnittstelle zur Programmierung in Java an, was eine Grundvoraussetzung in dieser Arbeit ist. Zusätzlich bietet es für die Erkennung von Gesichtern verschiedene Klassifikationsdateien an, die auf der Viola-Jones-Methode basieren und mit dem Lernalgorithmus bereits vortrainiert wurden und somit nur noch eingelesen werden müssen.

Es gibt allerdings noch weitere Bildverabeitungs-Frameworks, wie etwa *OpenIMAJ* von Hare et al. (2011). Diese Bibliothek bietet auch eine Gesichtserkennung anhand der Viola-Jones-Methode an und hat gleichzeitig den Vorteil, dass durch seine reine Java-Implementierung keine vorherige Kompilierung des Frameworks nötig ist. Mit *JavaCV* kommt ein weiterer Kandidat hinzu, da es eine alternative Java-Schnittstelle für das native *OpenCV* anbietet und dies zusätzlich als vorkompilierte Module mitbringt, unter anderem auch für ARM-basierte Systeme wie dem *Raspberry Pi*. Um die Entscheidungsfindung zu erleichtern, wurden alle drei Frameworks einem Leistungsvergleich auf dem Einplatinenrechner unterzogen, die in der nachfolgenden Tabelle 5 aufgelistet ist. Dazu wurde ein Bild vor der Kamera platziert und jedes Framework für jeweils 5 Minuten bei einer Abtastrate von 200 Millisekunden laufen gelassen.

Framework	**Min**	**Max**	**Ø**	**Median**	**Varianz**	**StdDev**
OpenCV	71	152	90	88	144	12
JavaCV	140	208	150	148	95	9
OpenIMAJ	1725	1899	1759	1761	374	19

Tabelle 5: Leistungsvergleich Gesichtserkennung in ms (Eigene Abbildung)

Das Ergebnis zeigt eindeutig, dass *OpenIMAJ* mit über 1.7 Sekunden im Schnitt am langsamsten abgeschnitten hat, was durch die reine Implementierung in Java bedingt ist. Am schnellsten hingegen lief mit rund 90 Millisekunden das C-basierte *OpenCV* mit seinen nativen Java-Aufrufen. Die al-

ternativen Java-Schnittstellen von *JavaCV* brauchen allerdings fast doppelt so lange, weshalb für die Gesichtserkennung des Prototypen nur *OpenCV* mit seinen eigenen Java-Wrappern in Betracht kommt.

3.5 Head Pose Estimation

Da die frontalen Klassifikatoren der Viola-Jones-Methode nicht nur gerade, sondern auch leicht seitlich blickende Gesichter erkennen, würde die Sprachsteuerung bereits aktiviert werden, wenn der Anwender gar nicht richtig in die Kamera schaut. Daher muss eine weitere Mechanik implementiert werden, die das Gesicht auf dessen Kopfhaltung analysiert und die Sprachsteuerung erst aktiviert, wenn der Kopf gerade zur Kamera steht. Dieser Vorgang wird auch als *Head Pose Estimation* bezeichnet.

Laut Nöll et al. (2011) zeigt die menschliche Wahrnehmung, dass eine korrekte Interpretation einer 3D-Szene auf Basis eines 2D-Bildes durchaus möglich ist, indem anhand der Identifizierung natürlicher Merkmale die Orte und Ausrichtungen verschiedener Objekte auf dem Bild identifiziert werden können. Der Schlüsselaspekt für diese Interpretation ist allerdings die korrekte Schätzung der Kameraposition, wofür er in seiner Arbeit drei verschiedene Ansatzklassen identifiziert hat:

- **Direct Linear Transformation (DLT)**: Diese Algorithmen schätzen die Kameraposition, indem Sie bestimmte Einschränkungen hinsichtlich des Lösungsraums ignorieren.
- **Perspective n-Point Problem (PnP)**: Diese Algorithmen beabsichtigen das direkte Abschätzen von Werten, die den Lösungsraum aller gültigen Kamerapositionen parametrisieren.
- **A Priori information estimators**: Neben den Punktkorrespondenzen gibt es vor der Berechnung häufig zusätzliche Informationen zur Kameraposition.

Die Algorithmen dieser Klasse verwenden diese Informationen, um zuverlässigere oder schnellere Ergebnisse zu liefern.

Nöll et al. (2011) zeigt weiterhin auf, dass die Schwäche von DLT in seiner unbeschränkten Minimierung der Parameter liegt und gerade bei einer geringen Anzahl an Merkmalen ein schlechtes Ergebnis liefert. In dieser Situation arbeiten PnP-Algorithmen signifikant robuster als DLT, da sie die Lösung üblicherweise nur in einem gültigen Lösungsraum suchen. Da keine zusätzlichen A Priori-Informationen zur Kameraposition bereitgestellt werden, wird dieser Ansatz für die Head Pose Estimation gewählt.

Das Ziel des *Perspective n-Point*-Problems ist das Auffinden relativer Abbildungen zwischen Objekten und einer Kamera anhand von *n* Paaren zwischen 3D-Punkten und ihren dazugehörigen 2D-Projektionen auf der Brennebene (Valeiras et al., 2016). Anhand eines Satzes von Übereinstimmungen zwischen den in einem weltlichen Bezugsrahmen ausgedrückten 3D-Punkten p_i und ihren 2D-Projektionen u_i auf das Bild wird schließlich die Rotationsmatrix R und der Translationsvektor t der Kamera in Bezug auf die Welt und der Brennweite f bestimmt (OpenCV, 2016). Dabei wird das *Lochkameramodell* verwendet, das eine Szenenansicht durch Projektion von 3D-Punkten in die Bildebene unter der Verwendung einer perspektivischen Transformation bildet und nachfolgender Formel zu Grunde liegt:

$$sm' = A[R|t]M' \qquad \Rightarrow \qquad s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Hierbei sind m' sowie M' die 2D bzw. 3D-Koordinaten der zusammengehörigen Punkte des Objektes. A bildet die Kameramatrix der intrinsischen Parameter, s die Skalierung der Tiefe (was bei monokularen Kameras ver-

nachlässigt werden kann), R die Rotationsmatrix und t den Translationsvektor. Das Zentrum der Bildebene wird durch c_x und c_y gebildet und die Brennweite durch f_x und f_y gegeben. Kennt man die zusammengehörigen Punkte (u,v), (X,Y,Z) sowie die Parameter der Kamera, können Rotationsmatrix und Translationsvektor berechnet werden. Diese werden benötigt um die eulerschen Winkel zu bestimmen, die die Lage eines Objektes im Raum anhand von Roll, Pitch und Yaw beschreiben.

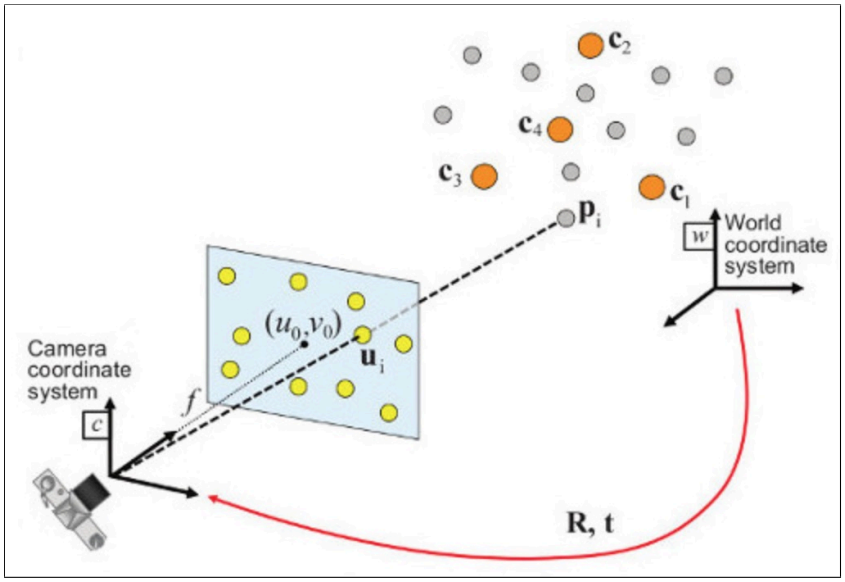


Abbildung 14: Funktionsweise PnP im Lochkameramodell (OpenCV, 2016)

Abbildung 14 veranschaulicht nochmals die Funktionsweise des PnP-Problems und den Zusammenhang von Kamera, 3D-Punkten sowie ihren dazugehörigen 2D-Projektionen auf der Brennebene. Das Bildzentrum, das vom Punkt (u_0, v_0) gebildet wird, sowie die Brennweite f sind hierbei die Parameter der Kamerakalibration. Die Rotationsmatrix R und der Translationsvektor t bestimmen sodann die Lage der Kamera, sodass sich die 3D-Koordinaten auf den 2D-Punkte perspektivisch überlagern.

Ein mögliches Framework, das zur Bestimmung der Kopflage in monokularen Bildern und Videos anhand des PnP-Ansatz verwendet werden kann, ist *Openface* von Baltrušaitis et al. (2016). Dazu berechnet es markante

Gesichtspunkte wie etwa Augen, Nase, Mund und Gesichtskonturen, die man auch als sogenannte *Facial Landmarks* bezeichnet. Diese Punkte werden anhand von *Conditional Local Neural Fields (CLNF)* berechnet und haben eine interne 3D-Representation, die über orthographische Kameraprojektion auf ein zweidimensionales Bild projiziert werden. Dadurch kann die Lage des Kopfes durch das Lösen des PnP-Problems bestimmt werden. Dem Framework sollten dafür aber die Parameter zur Kamerakalibrierung (wie etwa Brennweite und Bildzentrum) bereitgestellt werden, da es sonst eine grobe Schätzung anhand der Bildgröße durchführt. Abbildung 15 zeigt die Analyse-Pipeline von *Openface* inklusive *Facial Landmark*-Erkennung und Kopflagebestimmung.

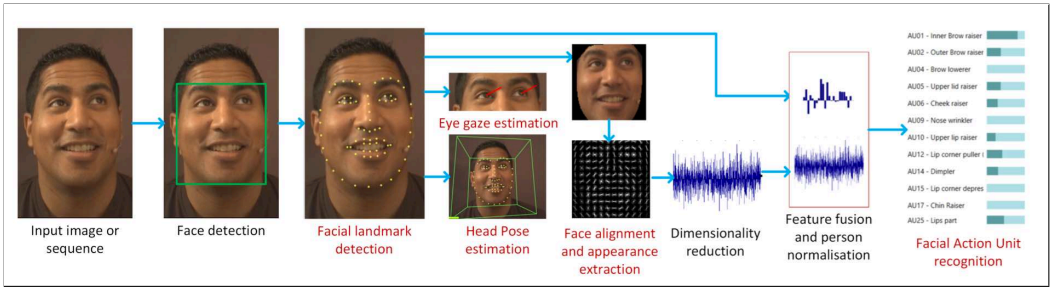
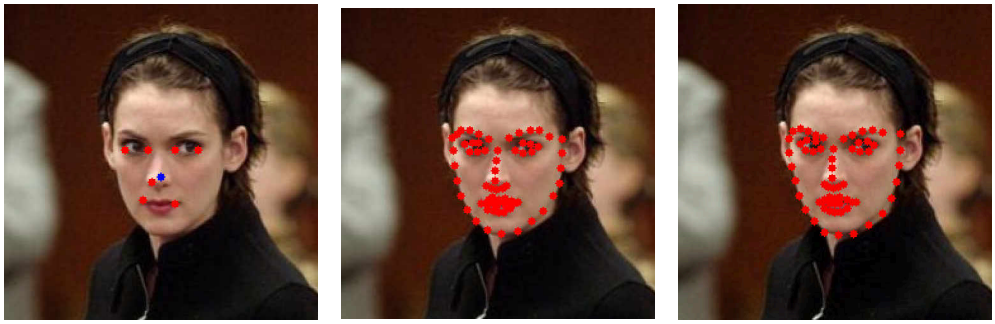


Abbildung 15: Openface Pipeline (Baltrušaitis et al., 2016)

In der ersten Testphase des Prototypen hat sich allerdings schnell herausgestellt, dass sich *Openface* auf dem *Raspberry Pi* nicht installieren lässt, da der Kompiliervorgang aufgrund des geringen Arbeitsspeichers immer wieder abgebrochen ist. Alternativ hierzu stellt Cech et al. (2014) in seiner Arbeit einen ähnlichen Algorithmus zur Bestimmung der Kopflage vor, der *Facial Landmarks* in monokularen Bildern lokalisiert und durch perspektivische Projektion mittels Lösung des PnP-Problems auf ein generisches 3D-Modell projiziert. Dabei wird jedoch nicht das gesamte Modell benötigt, sondern lediglich mindestens drei Punkte, die denen der *Facial Landmarks* entsprechen.

Zur Berechnung dieser Gesichtspunkte in zweidimensionalen Bildern bieten sich die Frameworks *FLandmark* (Uřičář et al., 2012) sowie dessen Nachfolger *CLandmark* (Uřičář et al., 2015) an. *FLandmark* berechnet in acht Punkten die Position des Mundes, der Augen, sowie der Nase. Im Gegensatz dazu kann *CLandmark* mit 68 Punkten weitaus mehr Features, wie zusätzlich Augenbrauen und die Umrandung des Gesichts erkennen. Ferner bietet es aber auch die Vorgängervariante mit acht Punkten an. Als weitere Alternative hierzu gibt es noch *Dlib* von King (2009), das auch in *Openface* zum Einsatz kommt (Baltrušaitis et al., 2016) und ebenso wie *CLandmark* eine Erkennung mit 68 Gesichtspunkten anbietet. Darüber hinaus stellt es weitere Funktionen zur Bildanalyse und Machine-Learning bereit. In der nachfolgenden Abbildung 16 werden die Gesichtspunkte der drei Frameworks gegenübergestellt.



(a) FLandmark (8P) **(b)** CLandmark (68P) **(c)** Dlib (68P)

Abbildung 16: Facial Landmark Frameworks (eigene Abbildung)

Da die Gesichtserkennung mit *OpenCV* bereits rund 90 Millisekunden auf dem *Raspberry Pi* benötigt, wird im nächsten Schritt versucht herauszufinden, wie lange dies in Kombination mit der Berechnung der *Facial Landmarks* dauert. Der Prototyp sollte so schnell wie möglich auf die Gesichtserkennung reagieren, um die Sprachsteuerung zeitnah aktivieren zu können. Für den Gesamtprozess sollte der Algorithmus jedoch nicht länger als zwei Sekunden benötigen. Um ein unbeabsichtigtes Auslösen durch kurzzeitigen Blickwechsel zu vermeiden, müssen außerdem mindestens zwei

Bilder hintereinander mit Kopfhaltung zur Kamera erkannt werden, was bedeutet, dass die gesamte Verarbeitungszeit pro Bild nicht höher als 1000 ms betragen darf, wenn die Erkennungsrate bei 100% liegt.

Um die Bearbeitungszeiten zu analysieren wird der Leistungsvergleich aus Kapitel 3.4.3 wieder herangezogen und mit den Frameworks erweitert. *JavaCV* bietet *FLandmark* direkt schon als vorkompiliertes Modul an, wogegen für *OpenCV* jeweils erst noch native Java-Wrapper für *FLandmark*, *CLandmark* und *Dlib* geschrieben werden mussten. Da auch *OpenIMAJ* eine Funktion zur Berechnung von 68 Gesichtspunkten bereitstellt, wird auch dies wieder mituntersucht. Die Ausführungszeit pro Framework beträgt wieder jeweils 5 Minuten und die Abtastrate wieder 200 Millisekunden. Als Klassifizierer wird hier für *JavaCV* und *OpenCV* die Datei *haarcascade_frontalface_default.xml* verwendet.

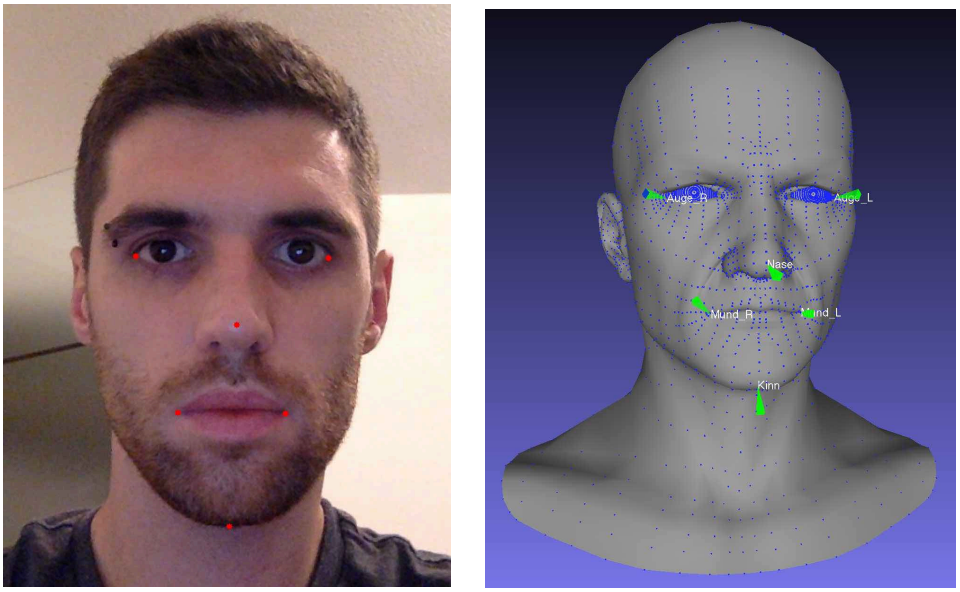
Framework	**Min**	**Max**	**Ø**	**Median**	**Var.**	**StdDev**
JavaCV [FLandmark 8]	253	491	341	339	663	25
OpenCV [FLandmark 8]	172	315	216	213	509	22
OpenCV [CLandmark 8]	169	244	183	182	123	11
OpenCV [CLandmark 68]	1393	1493	1424	1422	364	19
OpenCV [Dlib 68]	269	505	348	347	875	29
OpenIMAJ 68	5550	6831	6074	6038	67466	259

Tabelle 6: Leistungsvergleich Gesichtserkennung und Facial Landmarks in ms (Eigene Abbildung)

Die Ergebnisse aus Tabelle 6 machen deutlich, dass *OpenIMAJ* auf dem *Raspberry Pi* überhaupt nicht verwendet werden kann, da Erkennungszeiten von über sechs Sekunden pro Bild viel zu lange sind. Dies würde das System sehr träge erscheinen lassen und sich negativ auf die Nutzbarkeit auswirken. Die Berechnung der 8 Gesichtspunkte mit *OpenCV* in Kombination mit *FLandmark* und *CLandmark* geht hingegen mit durchschnittlich 216 bzw. 183 Millisekunden äußerst schnell. Für die Lokalisierung der vollen

68 Gesichtspunkte benötigt *CLandmark* jedoch mit rund 1,4 Sekunden pro Bild viel zu lange, um eine brauchbare *Head Pose Estimation* zu realisieren.

Mit *Dlib* hat sich aber schließlich ein Framework gefunden, das diese Punkte in einer maximalen Ausführungszeit von 505 Millisekunden liefern kann und somit für das weitere Vorgehen zur Berechnung der Gesichtspunkte verwendet wird. Die Bestimmung der Kopflage wird umso genauer, je mehr *Facial Landmarks* für die Projektion auf das 3D-Modell zur Verfügung stehen, weshalb Cech et al. (2014) in seinem Algorithmus beispielsweise zusätzlich auch das Kinn verwendet. Abbildung 17 zeigt die Gegenüberstellung der in dieser Arbeit verwendeten 2D *Facial Landmarks* durch *Dlib* sowie die Annotation der selben Punkte im dreidimensionalen Raum anhand eines generischen 3D-Modells[1].



(a) Facial Landmarks (Dlib) **(b)** Punkte im 3D Modell

Abbildung 17: Facial Landmarks und 3D Modell (eigene Abbildung)

[1] *Middle Aged Man Head 3d Model* by WindTrees - `https://www.turbosquid.com/3d-models/3d-polygonal-male-head/293761`

Weiterhin bietet *OpenCV* mit seinem *Calib3D*-Modul bereits Algorithmen zur Lösung des PnP-Problems an, die lediglich die 2D sowie 3D Koordinaten der *Facial Landmarks*, die Kameramatrix sowie den Verzerrungskoeffizienten benötigen (*solvePnP* und *solvePnPRansac*). Als Ergebnis bekommt man sodann die Rotationsmatrix und den Translationsvektor zurück, der gleichzeitig auch die Eulerschen Winkel darstellt. Projiziert man dann einen weit entfernten dreidimensionalen Punkt auf der Z-Achse mit den Gesichtspunkten, kann man wie in Abbildung 18 die Kopflage sichtbar machen. Fällt der Winkel der Drehung (*Yaw*) in einen gewissen Bereich, wird angenommen, dass der Kopf gerade zur Kamera steht.

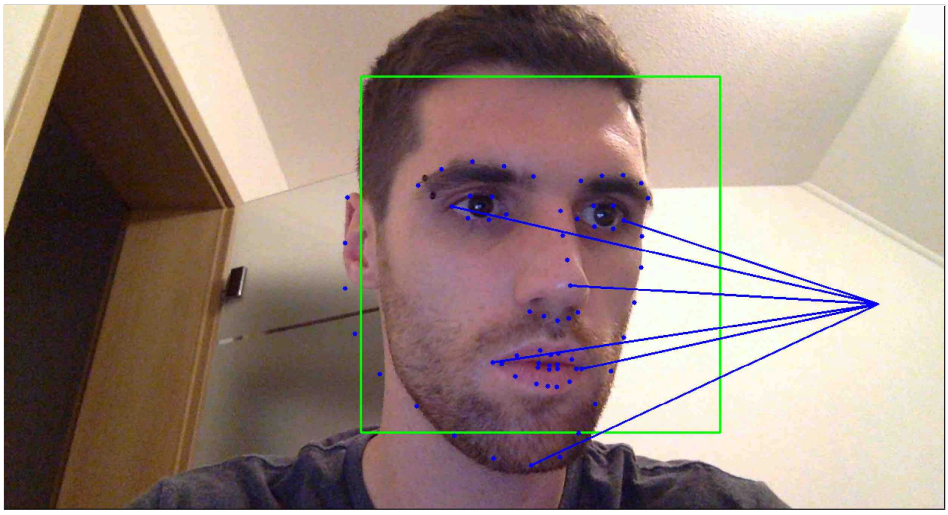


Abbildung 18: Kopflageerkennung des Prototyps (eigene Abbildung)

Mallik (2016) setzt den Ansatz von Cech et al. (2014) ebenfalls durch das Lösen des PnP-Problems um, geht aber davon aus, dass die Kamera nicht kalibriert ist und es keinen Verzerrungskoeffizienten gibt. Die Kameramatrix der intrinsischen Parameter A bestimmt er durch die Größe des Bildes. Da für die Logitech C920 der Winkel des Sichtfelds bekannt ist, wird die Kamera anhand ihrer Brennweite kalibriert. Da im ersten Testversuch das eigene generische Modell eine schlechtere Genauigkeit erreichte, werden zudem die 3D-Koordinaten von seinem Modell übernommen.

Abschließend wurde der Algorithmus wieder einem Leistungstest auf dem Einplatinenrechner unterzogen. Wie sich aus Tabelle 7 entnehmen lässt, benötigt er zur Berechnung der Kopflage im Schnitt rund 400 Millisekunden pro Bild. Dies ist für die Anforderungen dieser Arbeit somit ausreichend, um die Sprachsteuerung zeitnah aktivieren zu können.

Head Pose Estimation	**Min**	**Max**	**Ø**	**Median**	**Var.**	**StdDev**
OpenCV + Dlib	304	596	404	395	1792	42

Tabelle 7: Leistungstest der Head Pose Estimation in ms (Eigene Abbildung)

3.6 Limitationen

Der Hauptfokus dieser Masterarbeit liegt in der Konzeption sowie der prototypischen Umsetzung eines modularen Systems zur intuitiven Aktivierung natürlichsprachlicher Mensch-Maschine-Kommunikation in pervasiven Umgebungen. Sie deckt dabei aber nicht die Modellierung eines eingängigen Dialogsystems ab, sondern erkennt nur ausgewählte Schlüsselwörter wie etwa *Weiter*, *Zurück* oder *Abbruch*.

Auch die technische Vernetzung und Ansteuerung klassischer Geräte im Smart Home wird nicht weiter betrachtet. Die hier verwendeten Haushaltsgeräte sind Eigenimplementierungen, die zur Vereinfachung über Webschnittstellen bedient werden und Funktionen herkömmlicher Haushaltsgeräte imitieren, indem sie Geräusche wiedergeben.

Des Weiteren wird keine Gesichtserkennung im Sinne der Klassifizierung verschiedener Benutzer (*Face Recognition*) vorgenommen, sondern lediglich die Erkennung eines willkürlichen Gesichtes (*Face Detection*). Dadurch können die Geräte auch von Gästen verwendet werden, ohne sie vorher im System registrieren zu müssen.

Zusätzlich muss beim Aufbau des Gesamtsystems darauf geachtet werden, dass die Kameras der Haushaltsgeräte nicht direkt nebeneinander,

sondern in einem bestimmten Winkel und Abstand zueinander stehen, um eine zeitgleiche Erkennung zu vermeiden. Ist an einem Steuergerät eine Spracherkennung bereits aktiv, können die anderen jedoch erst verwendet werden, wenn der Dialog beendet ist.

Abschließend muss noch erwähnt werden, dass der Prototyp keine Möglichkeit anbietet, neue Sprachbefehle selbständig zu erlernen, wie es beispielsweise durch neuronale Netze möglich wäre.

3.7 Zusammenfassung

Dieses Kapitel hat die Analyse und das Design für eine prototypische Umsetzung zur natürlichen Interaktion mit Haushaltsgeräten näher betrachtet. Es wird für jedes Haushaltsgerät ein eigenes *Raspberry Pi* als Steuergerät verwendet, die sich selbständig im Netzwerk finden und ansprechen können. An allen Steuergeräten wird ein Mikrofon, ein Lautsprecher und eine Webcam angeschlossen, damit bei Erkennung eines Benutzers die Sprachsteuerung aktiviert wird. Durch modale Fission gibt das Systems jeweils Feedback in Form von gesprochener Sprache, einer leuchtenden LED sowie akustische Töne zurück. Zur Erkennung des Benutzers werden vortrainierte Klassifizierer der Viola-Jones-Methode verwendet und einer *Head Pose Estimation* unterzogen, um zu berechnen, ob der Anwender auch den Kopf gerade zur Kamera wendet.

Wird ein Benutzer schließlich erfolgreich an einem Steuergerät erkannt, werden alle anderen Steuergeräte über *Remote Services* mit R-OSGi in einen Ruhemodus versetzt. Mit *pocketSphinx* werden die gesprochenen Wörter sodann erkannt und durch eine Programmlogik interpretiert, die schließlich das imitierende Haushaltsgerät über HTTP anspricht. Abschließend wird der Ruhemodus auf allen Steuergeräten wieder zurückgesetzt, sobald die Interaktion beendet ist.

4 Implementierung

Die in Kapitel 3 analysierten Grundbausteine und das darauf aufbauende Konzept werden nun im Rahmen einer prototypischen Implementierung (Projektname *SmartKitchen*) umgesetzt und in den folgenden Unterkapiteln näher erklärt.

4.1 Allgemeiner Aufbau

In der nachfolgenden Abbildung 19 wird der Aufbau der einzelnen Komponenten und die Verbindungen zueinander illustriert. An jedem Steuergerät, für das als Basis ein *Raspberry Pi* dient, sind Webcam und Mikrofon über USB angeschlossen. Der Lautsprecher wird mit einem Kabel durch die integrierte Klinkebuchse verbunden. Die Haushaltsgeräte werden über HTTP durch einen *Representational State Transfer*-Webservice angesprochen, weshalb sie im gleichen Netzwerk angeschlossen sein müssen. Die dynamischen Hostnamen bzw. IP-Adressen der Haushaltsgeräte können im Steuergerät in einer Konfigurationsdatei eingetragen werden. Im prototypischen Aufbau wird alles über ein Funknetzwerk angeschlossen, womit die Remoteverbindung zu den anderen Steuergeräten über eine TCP-Verbindung im gleichen Netz geschieht.

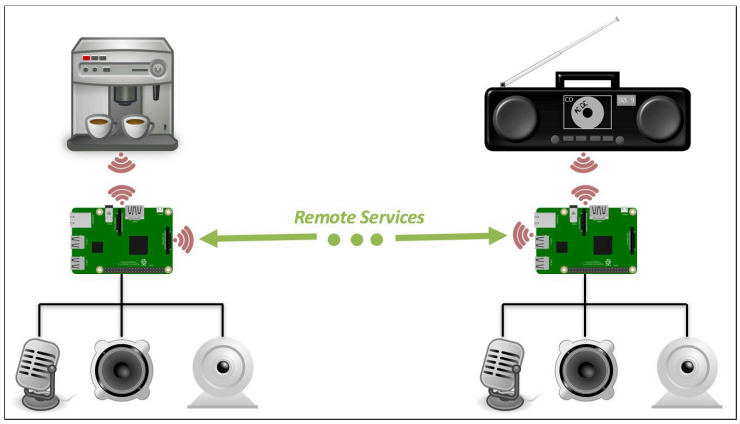


Abbildung 19: Aufbau der Hardware des Prototypen (eigene Abbildung)

4.2 Verwendete Werkzeuge

Da der Prototyp größtenteils in der Programmiersprache Java entwickelt wurde, kommt als Entwicklungsumgebung die *Eclipse IDE* in Version 4.6.1 (Codename Neon) zum Einsatz. Durch nachinstallierbare Plugins kann die IDE um neue Funktionen erweitert werden, wie beispielsweise dem *C/C++ Development Tooling (CDT)*, das verwendet wird, um die C++-basierten Teile des Prototypen umzusetzen.

Als Build-Management-Tool wird *Apache Maven* v3.3.9 in Kombination mit dem *JDK1.8* verwendet, da es selbständig Abhängigkeiten zu *Third-Party*-Bibliotheken auflöst und herunterlädt, sowie durch Plugins wie etwa dem *Apache Felix Maven Bundle Plugin* erweitert werden kann, um OSGi-Bundles zu generieren. Anhand selbstgeschriebener Shell-Skripte wird der Maven-Bauprozess dann automatisch durchgeführt

Zum Deployment der OSGi-Anwendung in den *Apache Karaf*-Laufzeit-Container werden sogenannte *Features* verwendet. Ein Feature ist eine logische Struktur, die die Teile der Anwendung sowie ihren Abhängigkeiten auflistet und daher die Bundles von selbst installieren kann. Dadurch dass Apache Karaf einen Netzwerkzugriff über eine Konsole anbietet, kann dies alles im Shell-Skript durchgeführt werden.

4.3 Frameworks

Das System setzt gewisse Frameworks voraus, die vor der Inbetriebnahme der Prototypen auf dem Einplatinenrechner installiert sein müssen. Da *OpenCV*, *Dlib* und *pocketSphinx* in C++ entwickelt wurden, müssen diese zuvor allerdings erst noch auf dem *Raspberry Pi* kompiliert werden. Glücklicherweise bieten alle drei Frameworks jeweils ein Makefile an, das die größte Arbeit übernimmt, sodass nur minimale Konfigurationen über *cmake* vorgenommen werden müssen. Die detaillierten Anleitungen hierzu sind im Anhang C beigefügt.

Ist Java auf dem Steuergerät installiert, generieren die Kompiliervorgänge von *OpenCV* und *pocketSphinx* jeweils eine Schnittstelle, mit der über das *Java Native Interface (JNI)* auf die nativen Funktionen zugegriffen werden kann. Für *Dlib* gibt es allerdings keine solche vorgefertigte API, weshalb in einem Unterprojekt eine eigene JNI-Lösung entwickelt wird. Dazu wird eine Java-Klasse erstellt, die mit dem Schlüsselwort *native* eine Brücke zu JNI herstellt.

```
1  public class LandmarkDetector {
2    private final long nativeObj;
3    public LandmarkDetector(String model) {
4      nativeObj = n_LandmarkDetector(model);
5    }
6    public double[] detect(Mat mat, int[] bbox) {
7      return n_detect(nativeObj, mat.nativeObj, bbox);
8    }
9    public void release() {
10     n_release(nativeObj);
11   }
12   // --- Native methods
13   private native long n_LandmarkDetector(String model);
14   private native void n_release(long nativeObj);
15   private native double[] n_detect(long nativeObj, long nativeImg, int[] bbox);
16 }
```

Algorithmus 2: Eigener Java-C-Wrapper für Dlib

Wie man in Algorithmus 2 erkennen kann, enthält die Klasse *LandmarkDetector* einen öffentlichen Konstruktor sowie eine Release- und Detect-Methode. Diese delegieren dann an die nativen Methoden (Zeile 13-15) weiter. Die Long-Variablen sind dabei die Pointer-Adressen der C-Objekte. Anhand dieser Klasse kann man sodann mit JNI eine Headerdatei generieren lassen, die man mit C++ ausimplementiert. Danach werden diese Dateien kompiliert und mit abhängigen Bibliotheken verlinkt und man erhält eine dynamische Bibliothek.

Algorithmus 3 zeigt das eigene Shell-Script, das anhand der Java-Datei zuerst die Class-Dateien erstellt, danach mit JNI die Headerdatei generiert und die CPP-Datei kompiliert und verlinkt.

```
1 #!/bin/bash
2 JDK_INCLUDE="/usr/lib/jvm/jdk-8-oracle-arm32-vfp-hflt/include"
3 LINUX_INCLUDE="/usr/lib/jvm/jdk-8-oracle-arm32-vfp-hflt/include/linux"
4 LOCAL_INCLUDE="/usr/local/include"
5 LOCAL_LIB="/usr/local/lib"
6 javac -cp libs/opencv-320.jar:./java java/dlib/LandmarkDetector.java
7 javah -jni -cp libs/opencv-320.jar:./java -d src dlib.LandmarkDetector
8 g++ -std=c++11 -I$JDK_INCLUDE -I$LINUX_INCLUDE -I$LOCAL_INCLUDE -c -Wall -Werror -fpic src/
      dlib_LandmarkDetector.cpp
9 g++ -std=c++11 -shared -o libdlib_java.so -L$LOCAL_LIB -ldlib -lopencv_core dlib_LandmarkDetector.o
```

Algorithmus 3: Shell Script zum Bauen des Dlib Wrappers

Die hiermit erzeugte dynamische Bibliothek sowie die Java-Klasse können schließlich direkt im Prototypen eingebunden werden. Man muss allerdings bedenken, dass jede kleine Änderung einen erneuten Bauvorgang voraussetzt.

4.4 Interner Aufbau

Der Aufbau des Prototypen kann von zwei Seiten betrachtet werden: Einerseits die Konstruktion der Bundles, die das fertige System bilden, andererseits den Aufbau der Projektstruktur woraus die Bundles generiert werden. Die nächsten Unterkapitel erläutern beide Aufbauten näher.

4.4.1 Bundles

Die eigentliche Implementierung des Prototypen besteht aus mehreren modularen Einheiten (OSGi Bundles), von denen jede Funktionen und Services eines gewissen Aufgabenbereiches bereitstellen. Sie werden daher in Bundles gruppiert, welche einerseits die Basisfunktionen bilden (*Platform Bundles*) und ferner die Haushaltsgeräte beschreiben und steuern (*Appliance Bundles*). Sie werden nachfolgend aufgelistet:

- **Platform Commons** - Gemeinsam genutzte Interfaces, Klassen und Utilities, die von anderen Bundles verwendet werden

- **Platform Connection** - Bereitstellung von Services zur Anbindung von Haushaltsgeräten über beispielsweise HTTP
- **Platform Audio** - Services zum Abspielen von Audiodateien und Aufzeichnen von Audiosignalen über das Mikrofon
- **Platform CV** - Implementiert den Service zur Gesichtserkennung anhand von OpenCV und Dlib sowie die Head Pose Estimation
- **Platform TTS** - Stellt einen Sprachsynthesizer bereit, der anhand von MaryTTS Text in Audiosignale verwandelt
- **Platform STT** - Dienste zum Erkennen von gesprochener Sprache mit dem pocketSphinx-Framework
- **Platform Discovery** - Untersucht das Netzwerk auf andere Steuergeräte und verwaltet ihre IP-Adressen
- **Platform Remote** - Stellt die Remote Verbindung zu den Steuergeräten bereit
- **Platform Evaluation** - Ein Service, der für die Evaluation verwendet wird
- **Appliance Core** - Die Kernkomponente, die alle Services steuert und die Logik abbildet
- **Appliance Coffeemaker** - Konfiguration und Implementierung zur entfernten Ansteuerung der Kaffeemaschine
- **Appliance Extractor** - Konfiguration und Implementierung zur entfernten Ansteuerung der Dunstabzugshaube
- **Appliance Musicplayer** - Konfiguration und Implementierung zur entfernten Ansteuerung des Music Players

Die einzelnen Bundles stehen in Beziehung zueinander und bilden damit ein modulares System. Das Appliance-Core-Bundle bildet die eigentliche Programmlogik, da es die Services wie Gesichtserkennung, Audio oder Remoteverbindung steuert. Durch ein *Callback*-Konstrukt wird der *ApplianceManager* von den einzelnen Services benachrichtigt, wie beispielsweise durch das Erkennen eines Benutzers, das Hinzukommen eines neuen Steuergeräts im Netzwerk oder das Ergebnis der Spracherkennung. Da die Interfaces der Services alle im Platform-Commons Bundle stecken, können die verschiedenen Implementierungen modular ausgetauscht werden. So

könnte man das Spracherkennungsmodul (Platform-STT) durch ein anderes ersetzen, das beispielsweise die Spracherkennung anhand der *Google API* über das Internet vornimmt. Durch das *Hot Deployment* der OSGI-Serviceplattform könnte man dies sogar im laufenden Betrieb vornehmen, ohne eine lange Ausfallzeit zu riskieren.

In Abbildung 20 werden die Beziehungen der Bundles untereinander nochmal bildlich dargestellt. Der besseren Übersicht wegen werden die weiteren Abhängigkeiten (wie etwa *pocketSphinx*, *JmDNS*, *OpenCV*) nicht dargestellt. Man erkennt hier ganz deutlich, dass die meisten Platform-Bundles ihre Services dem Appliance-Core Bundle zur Verfügung stellen, er aber auch die Konfigurationen und Steuerungsinstruktionen der drei Appliance-Bundles Musicplayer, Coffeemaker und Extraktor erhält. Diese Bundles stehen stellvertretend für viele weitere Haushaltsgeräte, die so angesteuert werden könnten. Das Discovery-Bundle stellt seine Dienste dem Remote-Bundle zur Verfügung, da es zum Aufbau der Remote Verbindungen daran interessiert ist, welche weiteren Steuergeräte im Netzwerk noch verfügbar sind.

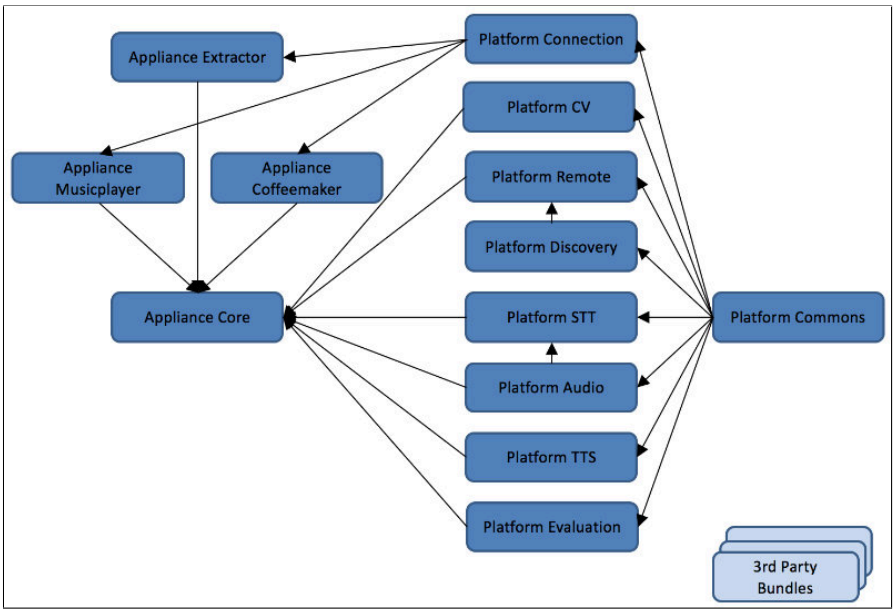


Abbildung 20: Aufbau der Bundles des Prototypen (eigene Abbildung)

Für die Abhängigkeiten *OpenCV*, *Dlib*, *JmDNS*, *R-OSGi* sowie *pocketSphinx* wurden eigene, sogenannte *Library Bundles* gebaut, die im Grunde genommen einfache OSGi-Bundles sind, die die Klassen und Ressourcen von Java-Bibliotheken bündeln und exportieren. In den jeweiligen *BundleActivators* wurde für die Bibliotheken sodann eine Mechanik implementiert, die beim Starten des Bundles die dynamischen Bibliotheken laden und der Laufzeit zur Verfügung stellen.

```
1  public class LibraryLoader {
2    public static void loadOpenCV() throws Exception{
3      String os = System.getProperty("os.name").toUpperCase();
4      String arch = System.getProperty("os.arch").toUpperCase();
5      String library = getLibrary(os, arch);
6      NativeUtils.loadLibraryFromJar(library);
7    }
8    private static String getLibrary(String os, String arch) {
9      if ("MAC OS X".equals(os) && "X86_64".equals(arch)) {
10       return "/jniLibs/macosx-x86_64/libopencv_java320.dylib";
11     }
12     if ("LINUX".equals(os) && "ARM".equals(arch)) {
13       return "/jniLibs/linux-armhf/libopencv_java320.so";
14     }
15     throw new UnsupportedOperationException("Unsupported OS/Architecture: " + os + " [" + arch + "]");
16   }
17 }
```

Algorithmus 4: Laden der dynamischen Bibliothek (OpenCV)

Algorithmus 4 zeigt einen vereinfachten Auszug der dynamischen Ladefunktion am Beispiel von *OpenCV*. Hier wird zwischen den Betriebssystemen *MAC OS X* und *LINUX* unterschieden, da auf einem Mac entwickelt, die Anwendung aber unter einem Linux ausgeführt wird. Der Algorithmus entscheidet anhand des Betriebssystems und der Prozessorarchitektur sodann welche Datei er für die Umgebung laden muss.

4.4.2 Maven Struktur

Der komplette Prototyp ist als ein *Multi-Module-Maven-Project* angelegt worden. Darunter versteht man eine hierarchische Struktur der einzelnen Teilprojekte, was den Vorteil bringt, dass man die verschiedenen Bundles jeweils in eigene Maven-Projekte auslagert, aber gleichzeitig alle Module bau-

en kann, in dem man den Bauprozess im Vaterknoten anstößt. Zusätzlich kann man sämtliche Einstellungen wie etwa Versionsnummern oder Zeichenkodierung in der POM-Datei des Vaterknotens (*parent pom*) eintragen, so dass alle an einer zentralen Stelle stehen. Hier werden auch die benötigten Abhängigkeiten verwaltet, die an die Kindprojekte weitervererbt werden. Das Projekt ist folgendermaßen aufgebaut:

- **smartkitchen**
  - **smartkitchen-appliance**
    - **smartkitchen-appliance-core**
    - **smartkitchen-appliance-coffeemaker**
    - **smartkitchen-appliance-extractor**
    - **smartkitchen-appliance-musicplayer**
  - **smartkitchen-config**
  - **smartkitchen-dependency**
    - **clandmark-api**
    - **flandmark-api**
    - **dlib-api**
    - **jmdns**
    - **opencv-api**
    - **pocketsphinx-api**
    - **rosgi**
  - **smartkitchen-platform**
    - **smartkitchen-platform-audio**
    - **smartkitchen-platform-commons**
    - **smartkitchen-platform-connection**
    - **smartkitchen-platform-cv**
    - **smartkitchen-platform-discovery**
    - **smartkitchen-platform-evaluation**
    - **smartkitchen-platform-remote**
    - **smartkitchen-platform-stt**
    - **smartkitchen-platform-tts**
  - **smartkitchen-executor**
  - **smartkitchen-feature**

Das Modul *smartkitchen-config* enthält die Konfigurationsdateien für Apache Karaf sowie die selbstgeschriebenen Buildskripte für lokales und entferntes Deployment. Das Projekt *smartkitchen-executor* dient zu Testzwecken zum Ausführen einzelner Services außerhalb der OSGi-Umgebung.

Das Untermodul *smartkitchen-feature* enthält die Feature-Datei und die Anweisungen zum bauen, kopieren und erstellen der Artefakte, die sodann im OSGi-Laufzeitcontainer installiert werden können. Die Aggregator-Projekte *smartkitchen-appliance*, *smartkitchen-platform* sowie *smartkitchen-dependency* bilden die Vaterknoten der in Kapitel 4.4.1 bereits vorgestellten Bundles.

4.5 Grundfunktionen

Die wichtigsten Funktionen und Services des Prototypen werden in den nachfolgenden Unterkapiteln näher erklärt. Um die Komplexität der Algorithmen vereinfacht darstellen zu können, werden sie als Aktivitätsdiagramme abgebildet und anhand dessen beschrieben.

4.5.1 ApplianceManager

Das *Appliance-Core* Bundle enthält die *ApplianceManager*-Klasse, die für die eigentliche Steuerung und den Ablauf verantwortlich ist. Sie ist der zentrale Kernpunkt, an dem alle Grundfunktionen zusammenlaufen und miteinander verwoben werden. Beispielsweise wird ein *ConversationTriggerService*-Interface verwendet, das vom Platform-CV-Bundle für die Gesichtserkennung ausimplementiert wird und den *ApplianceManager* über einen *Callback* benachrichtig, sobald ein gerade zur Kamera gerichtetes Gesicht erkannt wurde. Dabei deaktiviert der *ApplianceManager* sodann die Gesichtserkennung, was die LED zum erlöschen bringt und startet anschließend die Sprachsteuerung, wie man dem Aktivitätsdiagramm in Abbildung 21 entnehmen kann. Ist ein valides Wort erkannt worden, wird der Ruhemodus am lokalen Gerät sowie an allen anderen verbundenen Steuergeräten aktiviert und der Befehl an die Appliance (der Ansteuerung des Haushaltsgerätes) weitergeleitet. Anschließend wird der Ruhemodus wieder deaktiviert und die Gesichtserkennung von neuem gestartet.

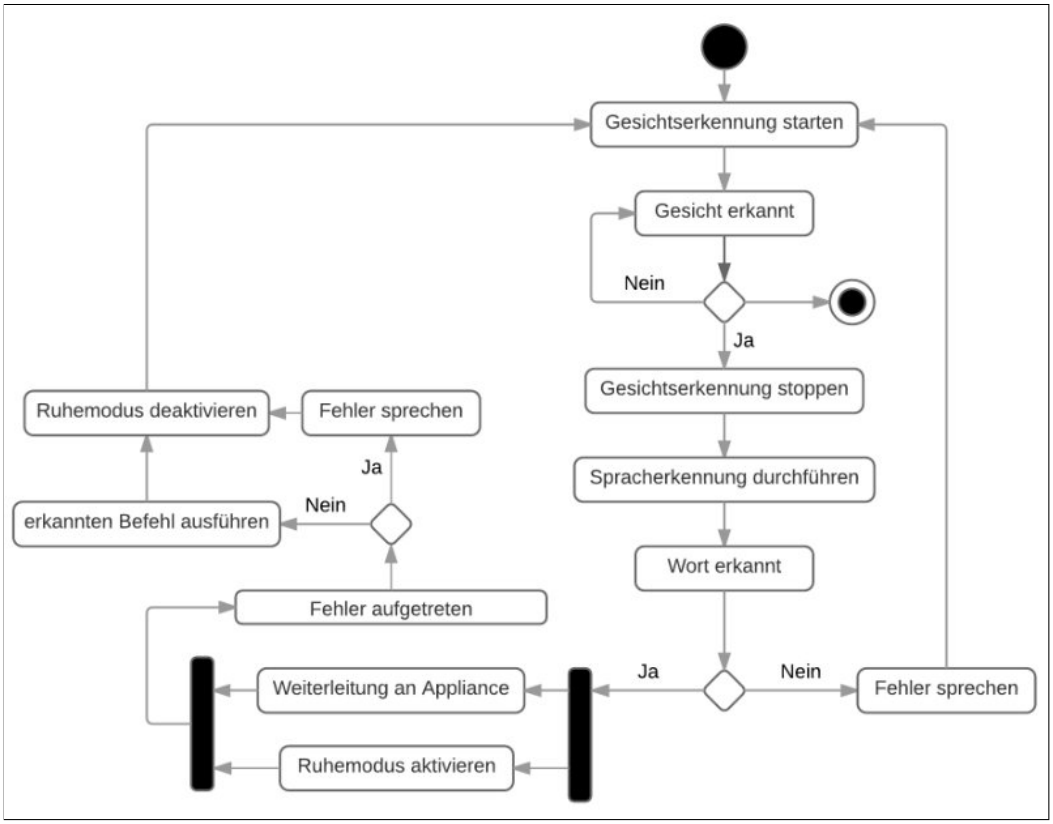


Abbildung 21: Funktionsweise ApplianceManager (eigene Abbildung)

4.5.2 Gesichtserkennung

Der eigentliche Gesichtserkenner ist eine Kombination aus einem frontalen Viola-Jones-Gesichtsklassifikator und einer Head Pose Estimation. Zuerst wird das generische 3D-Modell initialisiert und danach eine Schleife gestartet, die alle 200 Millisekunden ein Bild mit einer Auflösung von 1280x720 Pixeln von der Kamera abgreift. Anschließend wird dies in Graustufen konvertiert und mit Hilfe des Klassifikators überprüft, ob ein Gesicht erkannt wird. Trifft dies nicht zu, bricht die Sequenz hier ab und startet eine neue Schleifeniteration. Wurde ein Gesicht erkannt, werden sodann die *Facial Landmarks* bestimmt und das PnP-Problem gelöst. Anhand der Rotationsmatrix und des Translationsvektors wird dann ein entfernter Punkt auf der Z-Achse projiziert. Fällt dieser Punkt in einen gewissen Grenzbereich, wird ein Zähler inkrementiert. Ist dieser größer gleich 2, wird er wieder auf 0 ge-

setzt und der registrierte Service (in diesem Fall der *ApplianceManager*) über einen *Callback* benachrichtigt und die Schleife beginnt nach 200 Millisekunden von vorne.

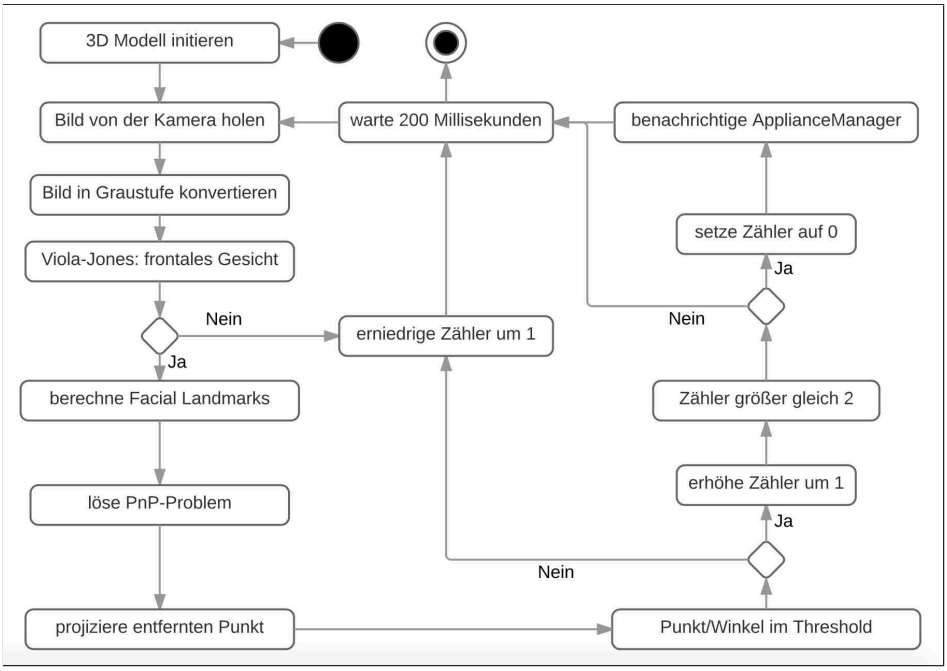


Abbildung 22: Funktionsweise Gesichtserkenner (eigene Abbildung)

4.5.3 RemoteServices

Auch das Aufbauen entfernter Verbindungen zu den anderen Steuergeräten sowie das Empfangen und Senden von Befehlen, wie etwa dem Ruhemodus, gehören zu den wichtigsten Funktionen und werden daher näher erläutert. Durchgeführt wird dies alles von einer *RemoteService*-Klasse, die ähnlich der Gesichtserkennung einen registrierten Service (hier der *ApplianceManager*) besitzt und über *Callbacks* benachrichtig. Kommt ein Befehl von einem entfernten Steuergerät an, wird der Service benachrichtigt, um das angeschlossene Haushaltsgeräte anzusprechen und den Befehl auszuführen. Dies kann einerseits der Ruhemodus sein, andererseits beispielsweise der Befehl *weiter* zum Abspielen des nächsten Liedes des Musikplayers. In einem anderen parallelen Prozess horcht der *RemoteService* auf lokale

Befehle, die er anhand gesammelter IP-Adressen des *DiscoveryService* über eine TCP-Verbindung an die entfernten *RemoteServices* weiterleitet. Danach werden die Verbindungen wieder geschlossen und der *ApplianceManager* im Erfolgs- sowie im Fehlerfall benachrichtigt. Das Aktivitätsdiagramm in Abbildung 23 stellt dies nochmals bildlich dar.

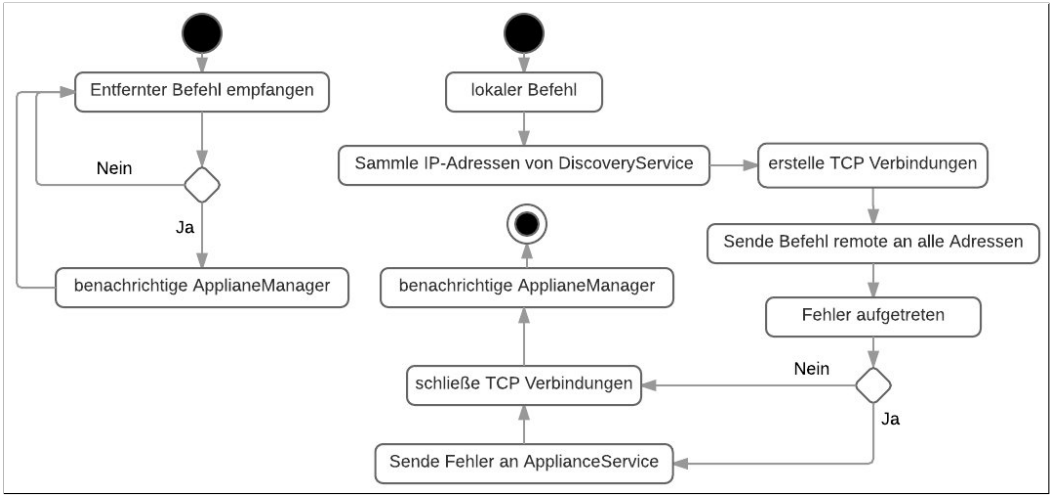


Abbildung 23: Funktionsweise Remote Service (eigene Abbildung)

4.5.4 Sprachsteuerung

Die Sprachsteuerung in der prototypischen Umsetzung setzt sich aus Sprachausgabe und Spracherkennung zusammen. Zu diesem Zweck wurden einerseits ein *VoiceResponseService* implementiert, der mit Hilfe von *MaryTTS* Text in gesprochene Sprache als AudioSignal umwandelt, die über einen *AudioService* wiedergegeben werden kann. Auf der anderen Seite konvertiert ein auf *pocketSphinx* basierender *VoiceRecognitionService* gesprochene Sprache in Text, indem er das Akkustikmodell und den Sprachkorpus von Radeck-Arneth et al. (2015) anwendet. Der *AudioService* spricht hierfür zusätzlich das Mikrofon an, um die Sprache digital aufzuzeichnen und dem *VoiceRecognitionService* im richtigen Format zur Verfügung zu stellen. Der *ApplianceManager* steuert schließlich alle Services zu einer gesamten Sprachsteuerung, wie in Abbildung 24 zu sehen ist.

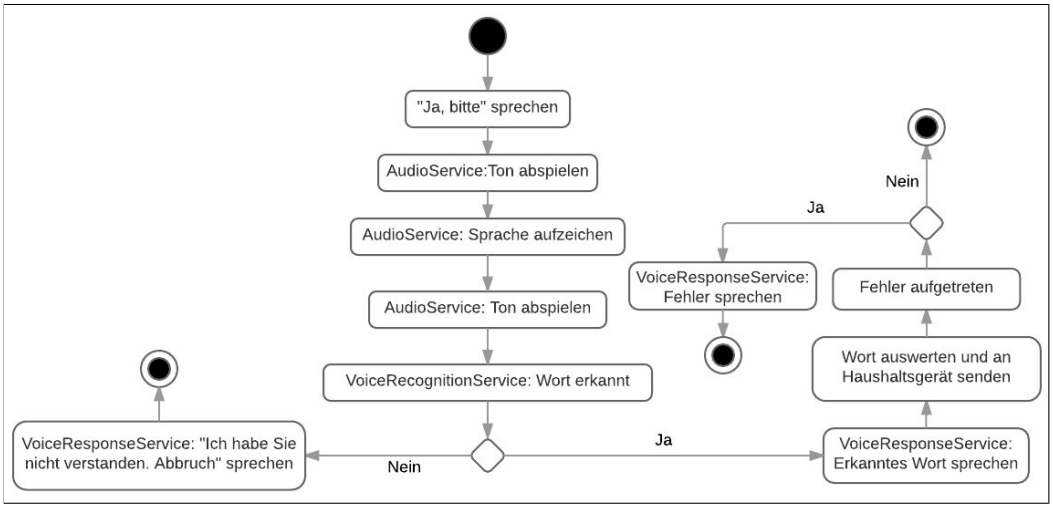


Abbildung 24: Funktionsweise Sprachsteuerung (eigene Abbildung)

4.6 Zusammenfassung

Der Prototyp dieser Arbeit ist in einer modularen Bauweise durch OSGi-Bundles umgesetzt worden, die es durch eine strikte Trennung von Schnittstellen und Implementierungen erlaubt, einzelne Implementierungen einfach zur Laufzeit austauschen bzw. erweitern zu können. Man könnte beispielsweise das Spracherkennungsmodul (Platform-STT) durch ein Bundle ersetzen, das anstatt eine lokale Spracherkennung von Signalwörtern, komplexe Sätze über ein internetbasiertes Dialogsystem erkennt. Hierfür muss schließlich nur das Bundle über die Konsole deinstalliert und das neue Bundle installiert werden.

Des Weiteren wurden die Module in Platform- sowie Appliance-Bundles aufgeteilt, um eine Trennung der Grundfunktionen und -services von den gerätespezifischen Ansteuerungen zu trennen. Man könnte zum Beispiel in einem nächsten Schritt reale Geräte der Hausautomation ähnlich dem Ansatz von Alexandersson et al. (2015) über einen Target Adapter mit USB oder KNX anschließen.

Schlussendlich arbeitet das Zusammenspiel aller Komponenten zudem schnell und stabil auf dem *Raspberry Pi 3 Modell B*.

5 Evaluation

In diesem Kapitel wird anhand eines Experiments untersucht, wie intuitiv eine Kombination von Gesichtserkennung und Sprachsteuerung zur natürlichen Interaktion mit Haushaltsgeräten von den Probanden wahrgenommen wird. Hierfür bildet eine intuitive Aktivierung der natürlichsprachlichen Mensch-Maschine-Kommunikation die Grundlage, weshalb im ersten Schritt eine technische Evaluation der Gesichtserkennung vorgenommen wird. Es soll herausgefunden werden, wie zuverlässig die Erkennung eines gerade zur Kamera gerichteten Kopfes bei unterschiedlichen Entfernungen arbeitet und wie lange das System im Schnitt benötigt, bis es die Probanden erkennt und die Sprachsteuerung initiiert. Im Anschluss daran wird die intuitive Wahrnehmung des gesamten Systems gemessen, indem die Teilnehmer einer Abschlussbefragung unterzogen werden.

5.1 Design des Experiments

Im ersten Teil der Evaluation soll die Effektivität des Gesichtserkennungsalgorithmus untersucht werden. Da dieser die erkannten Gesichter nach gerader oder seitlicher Lage klassifiziert, kann man hier die aus dem binären Klassifikationsproblem bekannten Leistungsmesswerte *Precision*, *Accuracy*, *Recall* und *False Alarm Rate* übertragen (Olson & Delen, 2008). In einem binären Entscheidungsproblem werden bestimmte Muster von einem Klassifikator entweder als positiv oder negativ gekennzeichnet und können in einer Tabelle dargestellt werden, die man auch als Konfusionsmatrix bezeichnet (Davis & Goadrich, 2006).

	Nutzer erkannt	**Nutzer nicht erkannt**
Kopf gerade	Richtig Positiv (TP)	Falsch Negativ (FN)
Kopf seitlich	Falsch Positiv (FP)	Richtig Negativ (TN)

Tabelle 8: Konfusionsmatrix der Gesichtserkennung (Eigene Abbildung)

Tabelle 8 zeigt die Konfusionsmatrix, die sich für den Anwendungsfall dieser Arbeit aufstellen lässt. Als *Richtig Positiv (TP)* oder *Richtig Negativ (TN)* werden Gesichter bezeichnet, die vom Algorithmus korrekt als gerade oder seitlich klassifiziert wurden. Werden Gesichter als seitlich gekennzeichnet, obwohl sie gerade schauen, sind sie *Falsch Positiv (FP)*. Sind sie seitlich und werden fälschlicher Weise als gerade eingestuft, spricht man von *Falsch Negativ (FN)*. Die Leistungsmesswerte lassen sich schließlich anhand dieser Variablen wie folgt ableiten (Olson & Delen, 2008, S.138) :

$$Recall = \frac{TP}{TP + FN} \qquad Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP} \qquad False\ Alarm\ Rate = \frac{FP}{FP + TN}$$

Um nun ein statistisch repräsentatives Ergebnis erzielen zu können, wird eine Stichprobengröße von 30 Teilnehmern gewählt. Da zusätzlich herausgefunden werden soll, wie sich die Gesichtserkennung auf unterschiedliche Distanzen verhält, müssen die Probanden jeweils fünf Aufgaben in einem Abstand von 100, 150 und 200 Zentimetern zur Kamera durchführen, deren Reihenfolge für jeden Teilnehmer randomisiert wird. Somit erhält man insgesamt 450 Datensätze, deren Videosequenzen dabei aufgezeichnet und durch ein Expertenrating den Variablen der Konfusionsmatrix zugeordnet werden. Weiterhin sind die Probanden durch das *Between-Subject-Design* in zwei Räumen mit unterschiedlichem Hintergrundbereich eingeteilt, da das System später auch in unterschiedlichen Umgebungen verwendet werden soll. Dies liegt darin begründet, weil beispielsweise nicht jeder Anwender die gleiche Küche besitzt. Die Mittelwerte der Durchführungszeiten werden dann schließlich durch Zweistichproben-T-Tests miteinander verglichen. Anhand der ANOVA-Varianzanalyse werden zudem die Durchführungszeiten der verschiedenen Abstände zur Kamera untersucht, da die Gruppengröße hier ≥ 3 ist (Sauro & Lewis, 2012, S.3).

Im zweiten Teil der Evaluation soll die intuitive Wahrnehmung des gesamten Systems gemessen werden, weshalb zwei Haushaltsgeräte miteinander gekoppelt werden. Für das Experiment kommen ein Musikplayer sowie eine Dunstabzugshaube zum Einsatz, wobei letztere nur zur Demonstration des Ruhemodus verwendet wird. Wie schon in Kapitel 3.3.2 beschrieben wurde, imitieren diese Geräte nur die Funktion von Haushaltsgeräten, indem sie Geräusche wiedergeben. Der Musikplayer kann jedoch mit den Schlüsselwörtern *„abspielen“*, *„weiter“*, *„zurück“*, *„stopp“* sowie *„Abbruch“* verwendet werden und den Probanden steht in den einzelnen Tasks frei, welchen Befehl sie verwenden möchten.

Um nun die subjektive Wahrnehmung der intuitiven Nutzung des Systems messen zu können, kann man den *Questionnaire for the subjective consequences of intuitive use (QUESI)* von Hurtienne & Naumann (2010) verwenden. Er besteht aus insgesamt 14 Fragen, die mit einer fünfstufigen Lickert-Skala beantwortet werden. Hierbei steht 1 für das schlechteste und 5 für das beste Ergebnis. Weiterhin werden die Fragen in fünf Unterskalen gruppiert:

- **W** - Wahrgenommene Mentale Beanspruchung (Fragen 1, 6, 11)
- **G** - Wahrgenommene Zielerreichung (Fragen 2, 7, 12)
- **L** - Wahrgenommener Lernaufwand (Fragen 3, 8, 13)
- **F** - Vertrautheit (Fragen 4, 9, 14)
- **E** - Wahrgenommene Fehlerrate (Fragen 5, 10)

Die Punktzahl jeder einzelnen Unterskala ergibt sich durch den Mittelwert der Antworten auf die jeweiligen Fragen in dieser Kategorie. Die Gesamtpunktzahl berechnet sich schließlich aus dem Mittelwert aller fünf Unterskalen (Naumann & Hurtienne, 2010).

Für die Abschlussbefragung wird der QUESI-Fragebogen noch um gängige demographischen Fragen und der Häufigkeit des Kochens/Backens im privaten Bereich erweitert. Darüber hinaus wird gezielt nachgefragt, ob man das Produkt auch selbst verwenden würde und kann in einer offenen

Antwort sein Pro und Contra nennen. Der gesamte Fragebogen kann im Anhang A eingesehen werden.

5.2 Durchführung

Für das Experiment konnten insgesamt 30 Teilnehmer (17 m, 13 w) gewonnen werden, die in zwei Räumlichkeiten mit unterschiedlichem Hintergrund aufgeteilt wurden. Der erste Raum hatte eine hellblau-gemusterte Tapete und eine dunkle Tür im Hintergrund, wohingegen der zweite Raum weiße Wände und Möbel aufwies. Beide Räume waren mit Rollläden abgedunkelt und durch Deckenlampen künstlich beleuchtet worden.

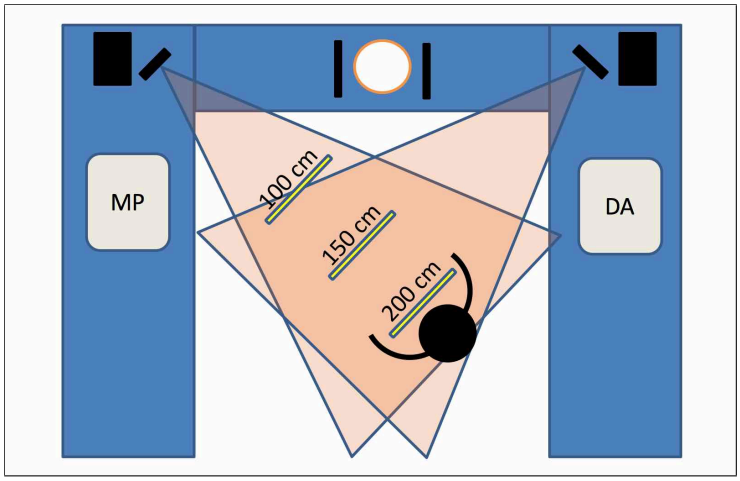


Abbildung 25: Schematischer Experimentaufbau (eigene Abbildung)

Abbildung 25 zeigt den schematischen Aufbau des Testsettings. Die Probanden mussten sich vorstellen, gerade einen Knödelteig mit den Händen geknetet zu haben und sollten dann den Musikplayer (MP) bedienen, während zeitgleich die Dunstabzugshaube (DA) auf höchster Stufe lief. Nach einer kurzen mündlichen Vorstellung des Experiments folgte eine kurze Testphase, in der die Probanden das System ausprobieren und sich mit der grundlegenden Funktion vertraut machen konnten. Die Schlüsselwörter für die Nutzung des Musikplayers wurden dabei mündlich erklärt.

Für die jeweiligen Tasks wurden drei Klebestreifen am Boden platziert, die die Entfernungen von 100, 150 und 200 Zentimetern zur Kamera markierten. Die Probanden stellten sich zu Beginn des Tasks auf die jeweiligen Markierungen und schauten von der Kamera weg, bis sie das Signal zum Start erhielten. Die Videosequenzen und die Dauer der Tasks wurden sodann bis zur Gesichtserkennung oder nach einem Timeout von 15 Sekunden aufgezeichnet.

5.2.1 Struktur der Stichprobe

Als Basis für das Experiment wird eine Stichprobe von 30 Personen in unterschiedlichen Berufs- und Altersgruppen mit einem Durchschnittsalter von etwa 34,23 Jahren (SD 14,18) herangezogen. Der Altersbereich liegt bei 12 bis 61 Jahren und abgesehen von 3 Schülern haben alle bereits einen Abschluss erworben. Der größte Teil wird hierbei mit etwa zwei Drittel von Mittlerer Reife und Hochschulabschlüssen gebildet. Weiterhin ist das Verhältnis von technischen zu nichttechnischen Berufen mit jeweils 15 Probanden sehr gut ausgeglichen.

Von der gesamten Stichprobe haben 12 Personen eine Brille und 8 Personen einen Bart, bei 3 Personen wurde sogar beides gleichzeitig getragen. Dies ist ein interessanter Faktor, da die Gesichtserkennung Merkmale wie Augenpaare und Kinn berechnet. Weiterhin kocht oder backt über die Hälfte aller Teilnehmer privat mindestens ein bis zwei mal pro Woche selbst, was eine potentielle Affinität zu Küchen- oder Haushaltsgeräten noch weiter unterstreicht.

5.2.2 Beobachtungen

Während der Durchführung des Experiments konnte man schnell beobachten, dass sich die Teilnehmer stark auf das Erlöschen der LED an der Kamera fokussierten und gespannt auf das folgende *„Ja, bitte?“* warteten. Da zwischen der Sprachausgabe und dem dreimaligen Piepsen (Indikator

für die Sprachaufzeichnung) eine kurze Pause liegt, hat der Großteil der Probanden anfänglich schon unmittelbar danach angefangen zu sprechen. Viele begründeten dies mit Erfahrungen von Call-Centern und Bandansagen, bei denen man direkt losreden kann.

Bei drei Probanden ist das System wegen Memory Leaks durch das Aufzeichnen des Videostreams abgestürzt, bei allen anderen lief es hingegen jedoch durchgehend.

5.2.3 Auswertung

Nachdem die Daten durch das Experiment gesammelt wurden und alle 450 Datensätze den Variablen der Konfusionsmatrix zugeordnet wurden, konnte mit der Auswertung begonnen werden. In einem Zweistichproben-T-Test stellte sich heraus, dass mit einem p-Wert von 0,9756 kein signifikanter Unterschied zwischen den beiden Räumen existiert. Daher wird die Annahme verworfen und alle Datensätze zusammen betrachtet.

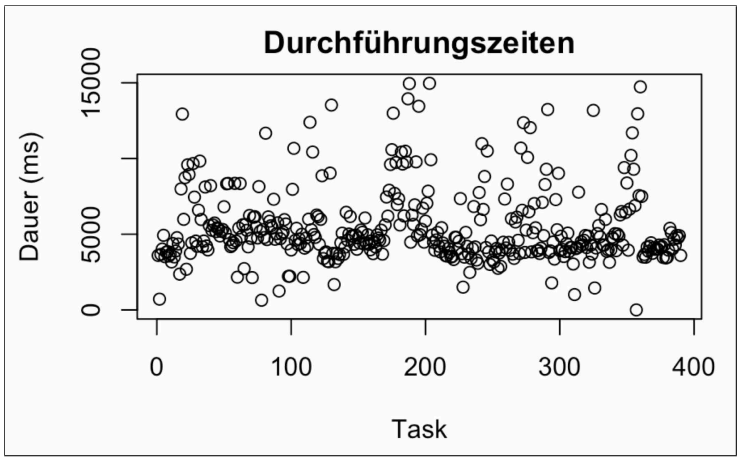


Abbildung 26: Durchführungszeiten (eigene Abbildung)

Der Plot in Abbildung 26 zeigt die Durchführungszeiten aller Tasks. Die durchschnittliche Bearbeitungsdauer beträgt hier 5429,167 Millisekunden, der Median liegt bei 4702,5 Millisekunden und die Standardabweichung bei 2482,354 Millisekunden. Weiterhin gibt es keine statistische Signifikanz

zwischen den Geschlechtern ($p = 0,7744$) sowie zwischen Personen mit und ohne Bart ($p = 0,9503$). Bei den Brillenträgern lässt sich allerdings ein deutlicher Unterschied nachweisen ($p = 0,01033$). Im Schnitt brauchen diese rund 687 Millisekunden länger, als Personen ohne Brille. Das kann man darin begründen, da der Gesichtserkenner unter anderem die Augenpaare analysiert und Brillen das Licht reflektieren können, was die Erkennung beeinträchtigt.

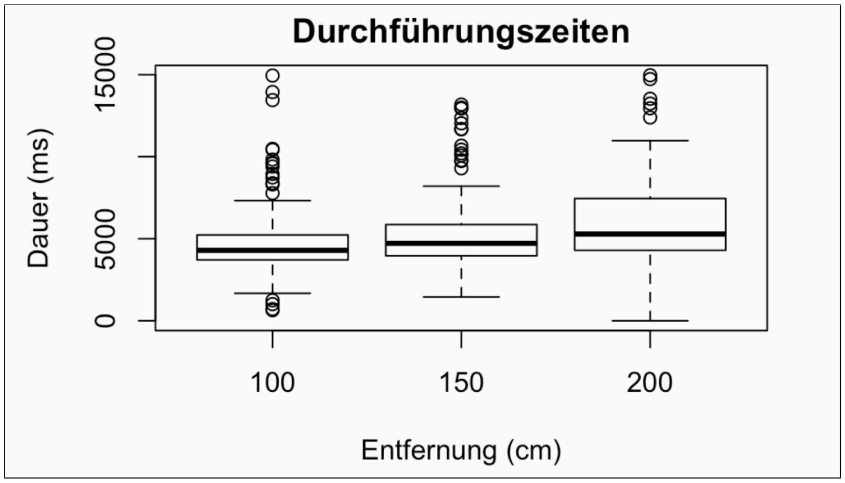


Abbildung 27: Durchführungszeiten pro Entfernung (eigene Abbildung)

Gruppiert man die durchschnittlichen Bearbeitungszeiten der Tasks nun nach deren Abstand zur Kamera, kann man in Abbildung 27 eine leichte Tendenz erkennen, die die Annahme aufstellt, dass mit steigendem Abstand zur Kamera auch die Erkennung schlechter wird. Um diese Annahme zu untersuchen, wird eine Varianzanalyse mit der ANOVA-Methode vorgenommen, um signifikante Unterschiede festzustellen.

	Df	**Sum Sq**	**Mean Sq**	**F value**	**Pr(>F)**
Distance	2	9,543e+07	47714229	8,023	0,000386
Residuals	387	2,302e+09	5947343		

Tabelle 9: ANOVA Zusammenfassung (eigene Abbildung)

Tabelle 9 listet eine Zusammenfassung der ANOVA-Analyse der drei Entfernungsgruppen auf. Der p-Wert in Abhängigkeit zu F ist hoch signifikant, weshalb es einen starken Unterschied zwischen den Gruppen gibt. Um nun herauszufinden, welche Gruppen von der Signifikanz betroffen sind, kann man auf die ANOVA-Tabelle den *Tukey HSD*-Test anwenden, dessen Ergebnisse in der nachfolgenden Tabelle 10 aufgelistet sind:

	Diff	**LWR**	**UPR**	**P Adj**
150 - 100	516,0902	-177,186429	1209,367	0,1875920
200 - 100	1233,1085	507,934781	1958,282	0,0002230
200 - 150	717,0184	5,732411	1439,769	0,0523807

Tabelle 10: Tukey HSD Auswertung (eigene Abbildung)

Man erkennt deutlich, dass ein hochsignifikanter Unterschied zwischen den Gruppen 100 und 200 existiert. Für den Gruppenvergleich 150 zu 200 ist ein p-Wert mit 0,052 Nahe der statistischen Signifikanz gegeben, weswegen man hier im Allgemeinen sagen kann, dass mit steigender Entfernung die Erkennung schlechter wird.

Da weiterhin die Effektivität des Gesichtserkennungsalgorithmus untersucht wird, werden die Datensätze nach deren Typ (*TP, TN, FP, FN*) gruppiert und gezählt. Anhand dieser Variablen ergibt sich somit eine Precision von 97%, einem Recall von 88%, eine Accuracy von 86% sowie eine False Alarm Rate von 75%. Die Summen sind zudem in der nachfolgenden Konfusionsmatrix dargestellt:

	Nutzer erkannt	**Nutzer nicht erkannt**
Kopf gerade	381 (TP)	53 (FN)
Kopf seitlich	12 (FP)	4 (TN)
Summe	393	57

Tabelle 11: Konfusionsmatrix Ergebnisse (Eigene Abbildung)

Die Antworten der Teilnehmer aus dem QUESI-Fragebogen wurden anschließend mit der prozentualen Trefferquote (Richtig Positiv) der zugehörigen Tasks erweitert, sodass man sie in zwei Gruppen einteilen kann. Die erste Gruppe besteht aus den Antworten der Personen, die eine hundertprozentige Erfolgsrate hatten, die zweite aus denen der Personen, bei denen mindestens eine Fehlerkennung oder Timeout aufgetreten ist. Die Teilergebnisse der Kategorien sowie der darauf berechnete QUESI-Score werden in der nachfolgenden Tabelle 12 zusammengetragen:

	100%	**< 100%**	**Wilcoxon**	**Gesamt**
Mentale Belastung	4,48	4,3	0,692	4,36
Zielerreichung	4,44	4,06	0,04649	4,18
Lernaufwand	4,44	4,43	0,7195	4,43
Vertrautheit	4,48	4,24	0,1124	4,31
Fehlerrate	4,44	3,21	0,001219	3,78
QUESI Score	4,46	4,11	0,01091	**4,21**
Teilnehmer	9	21	-	30

Tabelle 12: QUESI Scores nach Trefferquote (Eigene Abbildung)

Die Mittelwerte der einzelnen Unterkategorien zeigen deutlich, dass bei der mentalen Belastung, dem Lernaufwand, sowie der Vertrautheit keine großen Unterschiede existieren. Dies wird auch anhand eines anschließenden Wilcoxon-Rangsummentests bestätigt, da die p-Werte für den Gruppenvergleich oberhalb dem Schwellwert von 5% liegen. Der signifikante Unterschied in den beiden anderen Kategorien lässt sich allerdings daran erklären, dass das System die Probanden dieser Gruppe weniger gut erkannt hat, als die in der anderen. Dementsprechend wird die subjektive Wahrnehmung der Fehlerrate und der Zielerreichung negativ beeinflusst, was zu obigem Ergebnis führt und weswegen auch der QUESI-Score einen signifikanten Unterschied aufweist. Betrachtet man den gesamten QUESI-

Score, der einem Wert von 4,21 erzielt, so kann man darauf schließen, dass das System vom Großteil der Probanden akzeptiert und als intuitiv befunden wird. Das spiegelt sich auch mit den Angaben in der offenen Antwort von Pro und Contra des Systems wieder, da beispielsweise 10 Probanden es toll fanden, dass man die Hände frei hat und 7, dass man eine erleichterte Bedienung hat. Zusätzlich würden über drei Viertel aller Teilnehmer das System privat zu Hause verwenden wollen.

Auf der anderen Seite gab es auch Argumente gegen die Nutzung des Systems, wie beispielsweise Messungenauigkeiten bei der Gesichtserkennung oder dass es zu langsam für einen produktiven Betrieb sei. Weiterhin machten sich einige Probanden auch Gedanken bezüglich des Datenschutzes und zwei Teilnehmer fühlten sich wegen der ständig laufenden Kamera sogar überwacht. Die Antworten können im Anhang B eingesehen werden.

5.3 Fazit

Schlussendlich kann man sagen, dass die Gesichtserkennung mit der integrierten Head Pose Estimation ein sehr gutes Ergebnis bezüglich der Effektivität abliefert (95% Precision), diese allerdings im Durchschnitt rund 5 Sekunden für die Erkennung benötigt. Dazu muss an dieser Stelle noch erwähnt werden, dass während den Tasks die Videobilder annotiert, skaliert und gespeichert wurden, weshalb auf der CPU und dem internen Speicher mehr Last erzeugt wurde. Für den Produktivbetrieb kann man daher einen Teil der Gesamtdauer abziehen, da diese Funktion später nicht benötigt wird. Die durchschnittliche Obergrenze von rund 5 Sekunden ist daher ein akzeptabler Wert für das System.

Von insgesamt 450 Versuchen der Kontaktaufnahme gab es 381 richtige Erkennungen, was einer Trefferquote von rund 85% entspricht. Dies ist ein sehr gutes Ergebnis, wenn man berücksichtigt, dass man den Algorithmus zur Gesichtserkennung und der Head Pose Estimation später noch mit weiteren Parametern feinjustieren kann.

6 Ausblick

Die allgemeine Akzeptanz der Testpersonen ist sehr hoch, was sich auch in den errechneten QUESI-Scores wiederspiegelt. Somit ist belegt, dass eine Kombination aus Gesichtserkennung und Sprachsteuerung zur natürlichen Interaktion mit intelligenten Haushaltsgeräten dienen kann.

Da der prototypische Ansatz nicht nur für ein Küchenszenario geeignet ist, kann man das Einsatzgebiet später beispielsweise auch auf andere Zimmer im Haushalt oder sogar Werkstätten und Fabriken ausweiten. Durch den modularen Kern lassen sich die Funktionalitäten ganz einfach erweitern, so dass man beispielsweise ähnlich dem Kochbot-Ansatz von Alexandersson et al. (2015) durch das Connectivity-Bundle mit Target Adaptern richtige Geräte der Hausautomatisierung ansteuern kann. Möglich wäre dann etwa das Öffnen des Wasserhahns durch anschauen und sprechen.

Während der Entwicklung gab es allerdings auch viele Hürden, die es zu überwältigen galt. Beispielsweise ist das Debuggen durch die Verwendung mehrerer Technologien, die auf unterschiedlichen Programmiersprachen basieren, sehr umständlich. Da die hier verwendeten C++-basierten Frameworks über das Java Native Interface angesprochen werden, gab es zudem Probleme durch Memory Leaks, die den Arbeitsspeicher überfüllten und schließlich zum Absturz des gesamten Systems führten. Durch das freigeben der C++-Objekte und einem anschließenden Aufruf des GarbageCollectors bekam man die Memory Leaks allerdings gut in den Griff. Weiterhin nutzte die erste Version des Prototyps einen REST-basierten Webservice, der insgesamt gesehen zu viel CPU-Last erzeugte und schließlich genauso zu Systemabstürzen führte. Dieser Ansatz wurde aber erfolgreich durch R-OSGi von Rellermeyer et al. (2007) abgelöst.

Um im nächsten Schritt eine schnellere Gesichtserkennung bzw. Head Pose Estimation zu bekommen, könnte man die Grundplattform durch einen schnelleren Einplatinenrechner wie beispielsweise das Udoo-Board er-

setzen. Dies hat mehr Leistung und Arbeitsspeicher und kann somit die Prozesse der Bildverarbeitung schneller verarbeiten.

Weiterhin ist es auch denkbar, dass man die recht simple und auf vordefinierte Schlüsselwörter basierte Sprachsteuerung durch ein komplexeres Dialogsystem ersetzt, so dass zum Beispiel das Steuergerät einer Kaffeemaschine in Unterdialogen gezielt nachfragen kann, welche Art von Kaffee man gerne haben möchte, wenn der Anwender *„Einen Kaffee, bitte!“* spricht.

Durch die event-gesteuerte und auf *Callbacks*-basierte Struktur wäre es zudem auch denkbar, dass Ereignisse der intelligenten Haushaltsgeräte vom Steuergerät empfangen bzw. periodisch abgefragt werden, so dass die Sprachausgabe direkt darauf hinweisen könnte, dass für den nächsten Kaffee erst das Wasser aufgefüllt oder der Trester-Behälter geleert werden muss.

Literaturverzeichnis

Alexandersson, J., Schäfer, U., Britz, J., Rekrut, M., Arnold, F. & Reifers, S. (2015). Kochbot in the Intelligent Kitchen–Speech-enabled Assistance and Cooking Control in a Smart Home. In *8. aal-kongress* (Bd. 8, S. 396-405). VDE VERLAG GMBH.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. & Ayyash, M. (2015, 10). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys and Tutorials*, *17* (4), 2347–2376.

Apache Software Foundation. (2014). Apache Karaf Users' Guide [Software-Handbuch]. Zugriff am 17. 01. 2017 auf http://www-us.apache.org/dist/karaf/documentation/4_x.pdf

Apache Software Foundation. (2017, Januar). *Apache Aries Blueprint.* Zugriff am 18. 01. 2017 auf http://aries.apache.org/modules/blueprint.html

Badica, C., Brezovan, M. & Badica, A. (2013). An Overview of Smart Home Environments: Architectures, Technologies and Applications. In C. K. Georgiadis, P. Kefalas & D. Stamatis (Hrsg.), *Bci (local)* (Bd. 1036, S. 78). CEUR-WS.org.

Baltrušaitis, T., Robinson, P. & Morency, L.-P. (2016). Openface: an open source facial behavior analysis toolkit. In *Ieee winter conference on applications of computer vision.*

Blake, J. (2012). The natural user interface revolution. In *Natural User Interfaces in .NET* (S. 1–43). Manning Publications Co.

Bollhoefer, K. W., Meyer, K. & Witzsche, R. (2009, Februar). *Microsoft Surface und das Natural User Interface (NUI)* (Bericht). Berlin: Pixelpark.

Bolt, R. A. (1980, Juli). "Put-that-there": Voice and Gesture at the Graphics Interface. *SIGGRAPH Comput. Graph.*, *14* (3), 262–270.

Bradbury, J. S., Shell, J. S. & Knowles, C. B. (2003). Hands on Cooking: Towards an Attentive Kitchen. In *Chi '03 extended abstracts on human factors in computing systems* (S. 996–997). New York, NY, USA: ACM.

Cech, J., Franc, V. & Matas, J. (2014). A 3D Approach to Facial Landmarks: Detection, Refinement, and Tracking. In *Proceedings of the 2014 22nd international conference on pattern recognition* (S. 2173–2178). Washington, DC, USA: IEEE Computer Society.

Commission of The European Communities. (2008, September). Early challenges regarding the "Internet of Things". *Future networks and the Internet*.

Darrell, T., Tollmar, K., Bentley, F., Checka, N., Morency, L.-P., Rahimi, A. & Oh, A. (2002). Face-responsive interfaces: from direct manipulation to perceptive presence. In *International conference on ubiquitous computing* (S. 135–151).

Davis, J. & Goadrich, M. (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on machine learning* (S. 233–240).

Edwards, W. K. (2006, April). Discovery systems in ubiquitous computing. *IEEE Pervasive Computing*, *5* (2), 70-77.

Flotyński, J., Krysztofiak, K. & Wilusz, D. (2013). Building Modular Middlewares for the Internet of Things with OSGi. In A. Galis & A. Gavras (Hrsg.), *The future internet: Future internet assembly 2013: Validated results and new horizons* (S. 200–213). Berlin, Heidelberg: Springer Berlin Heidelberg.

Fokusgruppe Connected Home. (2014, Oktober). *Vor dem Boom - Marktaussichten für Smart Home.* Ergebnisdokument der Fokusgruppe Connected Home. Bundesministerium für Wirtschaft und Energie. Zugriff am 06. 01. 2017 auf http://www.bmwi.de/BMWi/Redaktion/PDF/IT-Gipfel/it-gipfel-2014-ergebnisdokument-ag-8-connected-home

Freund, Y. & Schapire, R. E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory* (S. 23–37).

Gaida, C., Lange, P., Petrick, R., Proba, P., Malatawy, A. & Suendermann-Oeft, D. (2014). *Comparing open-source speech recognition toolkits.*

Gartner. (2014, November). *Gartner Says 4.9 Billion Connected "Things" Will Be in Use in 2015.* Pressemitteilung. Zugriff am 03. 01. 2017 auf http://www.gartner.com/newsroom/id/2905717

Gartner. (2015, Dezember). *Gartner Says Smart Cities Will Use 1.6 Billion Connected Things in 2016.* Pressemitteilung. Zugriff am 03. 01. 2017 auf http://www.gartner.com/newsroom/id/3175418

Georgieff, P. (2008). Ambient Assisted Living. *Marktpotenziale IT-unterstützter Pflege für ein selbstbestimmtes Altern, FAZIT Forschungsbericht, 17*, 9–10.

Hall, R., Pauls, K., McCulloch, S. & Savage, D. (2011). *OSGi in Action: Creating Modular Applications in Java*. Manning.

Hare, J. S., Samangooei, S. & Dupplaw, D. P. (2011). OpenIMAJ and ImageTerrier: Java Libraries and Tools for Scalable Multimedia Analysis and Indexing of Images. In *Proceedings of the 19th acm international conference on multimedia* (S. 691–694). New York, NY, USA: ACM.

Henseler, W. (2011). Von GUI zu NUI - Die nächste Generation des User-Interface-Designs. *Konturen 2011*. Zugriff am 07.

01. 2017 auf https://www.hs-pforzheim.de/fileadmin/user_upload/uploads_redakteur/Presse-_und_Oeffentlichkeitsarbeit/Publikationen/Konturen/2011/Aus_Forschung_und_Lehre/Von_GUI_Zu_NUI.pdf

Huggins-Daines, D., Kumar, M., Chan, A., Black, A. W., Ravishankar, M. & Rudnicky, A. I. (2006, May). Pocketsphinx: A Free, Real-Time Continuous Speech Recognition System for Hand-Held Devices. In *2006 ieee international conference on acoustics speech and signal processing proceedings* (Bd. 1, S. I-I).

Hurtienne, J. & Naumann, A. (2010). QUESI – A questionnaire for measuring the subjective consequences of intuitive use. In R. Porzel, N. Sebanz & M. Spitzer (Hrsg.), *Interdisciplinary college 2010. focus theme: Play, act and learn* (S. 536+).

IBM. (2013). *IBM® WebSphere® Application Server and Intel Technologies* (Bericht). USA: Autor. Zugriff auf ftp://public.dhe.ibm.com/software/webservers/appserv/was/high-performance-computing-xeon-ibm-websphere-paper.pdf

Jain, J., Lund, A. & Wixon, D. (2011). The Future of Natural User Interfaces. In *Chi '11 extended abstracts on human factors in computing systems* (S. 211–214). New York, NY, USA: ACM.

Kim, H.-J., Jeong, K.-H., Kim, S.-K. & Han, T.-D. (2011). Ambient Wall: Smart Wall Display Interface Which Can Be Controlled by Simple Gesture for Smart Home. In *Siggraph asia 2011 sketches* (S. 1:1–1:2). New York, NY, USA: ACM.

King, D. E. (2009). Dlib-ml: A Machine Learning Toolkit. *Journal of Machine Learning Research*, *10*, 1755-1758.

Klatt, D. H. (1987). Review of text-to-speech conversion for English. *The Journal of the Acoustical Society of America*, *82* (3), 737–793.

Kreuzer, K. & Eichstädt-Engelen, T. (2015). *openHAB: Automatisiertes Heim - Teil 1*. entwickler.Press.

Maksimović, M., Vujović, V., Davidović, N., Milošević, V. & Perišić, B. (2014). Raspberry pi as internet of things hardware: performances and constraints. *design issues*, *3*, 8.

Malaka, R., Butz, A. & Hußmann, H. (2009). *Medieninformatik: eine Einführung*. Pearson Studium.

Mallik, S. (2016, September). *Head Pose Estimation using OpenCV and Dlib.* Blog. Zugriff am 28. 01. 2017 auf http://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/

Naumann, A. & Hurtienne, J. (2010). Benchmarks for Intuitive Interaction with Mobile Devices. In *Proceedings of the 12th international conference on human computer interaction with mobile devices and services* (S. 401–402). New York, NY, USA: ACM.

Nöll, T., Pagani, A. & Stricker, D. (2011). Markerless camera pose estimation–an overview. In *Oasics-openaccess series in informatics* (Bd. 19).

Olson, D. L. & Delen, D. (2008). *Advanced data mining techniques* (1st Aufl.). Springer Publishing Company, Incorporated.

OpenCV. (2016, Dezember). The OpenCV Reference Manual [Software-Handbuch]. Zugriff am 29. 01. 2017 auf http://docs.opencv.org/3.2.0/ (3.2.0)

OpenCV. (2017). *OpenCV Website.* Zugriff am 15. 01. 2017 auf http://opencv.org

OSGi Alliance. (2014, Juni). *OSGi Service Platform, Core Specification, Release 6, Version 1.8* (techreport). San Ramon, CA: Autor.

Potamitis, I., Georgila, K., Fakotakis, N. & Kokkinakis, G. K. (2003). An integrated system for smart-home control of appliances based on remote speech interaction. In *Interspeech.*

Radeck-Arneth, S., Milde, B., Lange, A., Gouvêa, E., Radomski, S., Mühlhäuser, M. & Biemann, C. (2015). Open Source German Distant Speech Recognition: Corpus and Acoustic Model. In *Proceedings of the 18th international conference on text, speech, and dialogue - volume 9302* (S. 480–488). New York, NY, USA: Springer-Verlag New York, Inc.

Rashad, M. Z., El-Bakry, H. M., Isma'il, I. R. & Mastorakis, N. (2010). An Overview of Text-to-Speech Synthesis Techniques. In *Proceedings of the 4th international conference on communications and information technology* (S. 84–89). Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS).

Rellermeyer, J. S. (2005). *jSLP project, Java Service Location Protocol (2005).* Zugriff am 07. 01. 2017 auf http://jslp.sourceforge.net

Rellermeyer, J. S. & Alonso, G. (2007, März). Concierge: A Service Platform for Resource-constrained Devices. *SIGOPS Oper. Syst. Rev., 41* (3), 245–258.

Rellermeyer, J. S., Alonso, G. & Roscoe, T. (2007). R-OSGi: Distributed Applications Through Software Modularization. In *Proceedings of the 8th acm/ifip/usenix international conference on middleware* (S. 1–20). Berlin, Heidelberg: Springer-Verlag.

Robert Bosch Hausgeräte GmbH. (2016, September). *Die Gestaltung prägt das Markenerlebnis.* Pressemitteilung. Zugriff am 04. 01. 2017 auf http://www.bosch-home.com/Files/Bosch2/De/de/Press/IFA%202016/PI_8729_0916_Bosch_Design.pdf

Sahoo, S. (2012, Oktober). OSGi Application Development using GlassFish Server (Software-Handbuch Nr. 1.5). Zugriff am 28. 01. 2017 auf https://glassfish.java.net/public/GF-OSGi-Features.pdf

Samsung. (2017). *Smart Home Cloud API.* Zugriff am 22. 01. 2017 auf http://developer.samsung.com/smart-home

Sauro, J. & Lewis, J. R. (2012). *Quantifying the user experience: Practical statistics for user research* (1st Aufl.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Schröder, M. & Trouvain, J. (2003). The German Text-to-Speech Synthesis System MARY: A Tool for Research, Development and Teaching. *International Journal of Speech Technology*, *6*, 365-377.

Sietmann, R. (2016). Standards im Wettstreit. *Das große Smart Home Kompendium*, 12-19.

Stabit, A. (2016). Haushaltsgeräte mit Netzfunktionen. *Das große Smart Home Kompendium*, 82-84.

Strese, H., Seidel, U., Knape, T. & Botthof, A. (2010). *Smart Home in Deutschland: Untersuchung im Rahmen der wissenschaftlichen Begleitung zum Programm Next Generation Media (NGM) des Bundesministeriums für Wirtschaft und Technologie*. VDI.

Tanenbaum, A. S. & Steen, M. v. (2006). *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Tucker, G., Wu, M., Sun, M., Panchapagesan, S., Fu, G. & Vitaladevuni, S. (2016). Model compression applied to small-footprint keyword spotting. In *Proc. interspeech.*

Uřičář, M., Franc, V. & Hlaváč, V. (2012, 24-26. February). Detector of Facial Landmarks Learned by the Structured Output SVM. In G. Csurka &

J. Braz (Hrsg.), *Visapp '12: Proceedings of the 7th international conference on computer vision theory and applications* (Bd. 1, S. 547-556). Portugal: SciTePress — Science and Technology Publications.

Uřičář, M., Franc, V., Thomas, D., Akihiro, S. & Hlaváč, V. (2015, May). Real-time Multi-view Facial Landmark Detector Learned by the Structured Output SVM. In *11th ieee international conference and workshops on automatic face and gesture recognition (fg), 2015* (Bd. 02, S. 1-8).

Valeiras, D. R., Kime, S., Ieng, S.-H. & Benosman, R. B. (2016). An event-based solution to the perspective-n-point problem. *Frontiers in neuroscience*, *10*.

van Hoff, A. (2002). *JmDNS.* Zugriff am 07. 01. 2017 auf http://jmdns.sourceforge.net

Vertegaal, R., Slagter, R., van der Veer, G. & Nijholt, A. (2001). Eye Gaze Patterns in Conversations: There is More to Conversational Agents Than Meets the Eyes. In *Proceedings of the sigchi conference on human factors in computing systems* (S. 301–308). New York, NY, USA: ACM.

Viola, P. & Jones, M. J. (2004, Mai). Robust Real-Time Face Detection. *Int. J. Comput. Vision*, *57* (2), 137–154.

Walker, W., Lamere, P. & Kwok, P. (2002, August). *FreeTTS: A Performance Case Study* (Bericht). Mountain View, CA, USA: Sun Microsystems, Inc.

Wooldridge, M. (2002). Intelligent Agents: The Key Concepts. In *Proceedings of the 9th eccai-acai/easss 2001, aemas 2001, holomas 2001 on multi-agent-systems and applications ii-selected revised papers* (S. 3–43). London, UK, UK: Springer-Verlag.

Zhao, W., Chellappa, R., Phillips, P. J. & Rosenfeld, A. (2003). Face recognition: A literature survey. *ACM computing surveys (CSUR)*, *35* (4), 399–458.

Anhang A Fragebogen

Smart Kitchen

* Erforderlich

1. **Teilnehmer-ID** *

QUESI - Questionnaire for Intuitive Use

Hinweis: Versuchen Sie Ihre Einschätzung des Systems ausschließlich auf die Benutzung des Systems zu beziehen (und nicht z.B. auf die Schwierigkeit der Aufgabe an sich). Es gibt keine richtigen oder falschen Antworten. Bitte antworten Sie spontan und lassen Sie keine Fragen aus.

2. *

Markieren Sie nur ein Oval pro Zeile.

	trifft gar nicht zu	trifft wenig zu	trifft teils-teils zu	trifft ziemlich zu	trifft völlig zu
Es gelang mir, das System ohne Nachdenken zu benutzen.	◯	◯	◯	◯	◯
Ich habe erreicht, was ich mit dem System erreichen wollte.	◯	◯	◯	◯	◯
Mir war sofort klar, wie das System funktioniert.	◯	◯	◯	◯	◯
Der Umgang mit dem System erschien mir vertraut.	◯	◯	◯	◯	◯
Bei der Benutzung des Systems sind keine Probleme aufgetreten.	◯	◯	◯	◯	◯
Die Systembenutzung war unkompliziert.	◯	◯	◯	◯	◯
Es gelang mir, meine Ziele so zu erreichen, wie ich es mir vorgestellt habe.	◯	◯	◯	◯	◯
Es fiel mir von Anfang an leicht, das System zu benutzen.	◯	◯	◯	◯	◯
Mir war immer klar, was ich tun musste, um das System zu benutzen	◯	◯	◯	◯	◯
Die Benutzung des Systems verlief reibungslos.	◯	◯	◯	◯	◯
Ich musste mich kaum auf die Benutzung des Systems konzentrieren.	◯	◯	◯	◯	◯
Das System hat mich dabei unterstützt, meine Ziele vollständig zu erreichen.	◯	◯	◯	◯	◯
Die Benutzung des Systems war mir auf Anhieb klar.	◯	◯	◯	◯	◯
Ich tat immer automatisch das Richtige, um mein Ziel zu erreichen.	◯	◯	◯	◯	◯

Demographische Daten

3. **Alter** *

4. **Geschlecht** *

Markieren Sie nur ein Oval.

- ◯ männlich
- ◯ weiblich

5. **Beruf / Studiengang** *

6. **Was ist Ihr höchster Bildungsabschluss?** *

Markieren Sie nur ein Oval.

- ◯ ohne Abschluss
- ◯ Hauptschulabschluss
- ◯ Mittlere Reife
- ◯ Abitur
- ◯ Hochschulabschluss

7. **Tragen Sie eine Brille?** *

Markieren Sie nur ein Oval.

- ◯ Ja
- ◯ Nein

8. **Tragen Sie einen Bart?** *

Markieren Sie nur ein Oval.

- ◯ Ja
- ◯ Nein

9. **Wie oft kochen oder backen Sie zu Hause selbst?** *

Markieren Sie nur ein Oval.

- ◯ Seltener als 1-2 mal pro Monat
- ◯ 1-2 mal pro Monat
- ◯ 1-2 mal pro Woche
- ◯ 3-4 mal pro Woche
- ◯ Täglich oder fast täglich

10. **Würden Sie das Produkt privat zu Hause verwenden wollen?** *

Markieren Sie nur ein Oval.

- ◯ Ja
- ◯ Nein

11. **Gründe dafür**

12. **Gründe dagegen**

Anhang B Ergebnisse der Evaluation

SmartKitchen

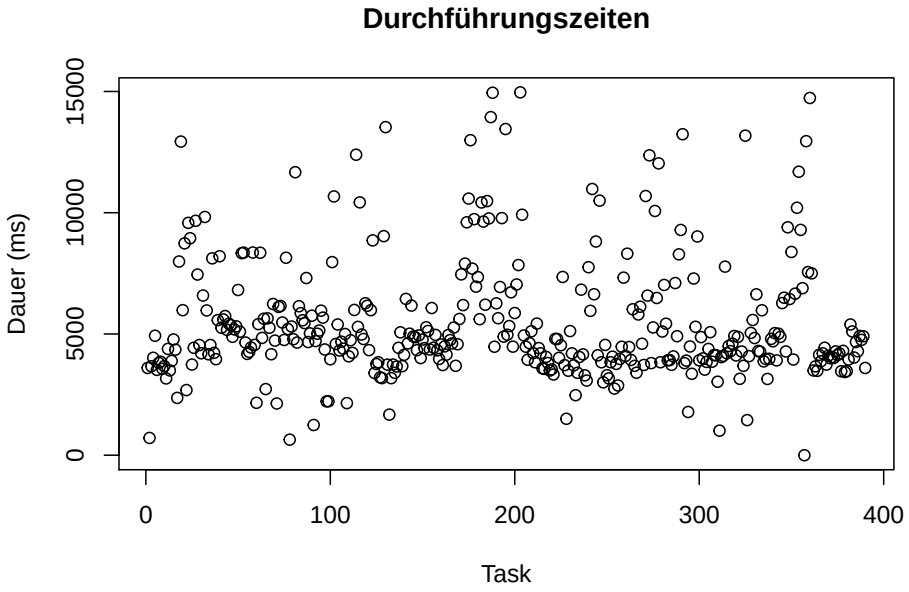
Der Code in diesem Notebook dient als Analyse zur Evaluation des SmartKitchen Experimentes.

Datensatz laden

```
library(readr)
tasks.all <- read_delim("tasks.csv",";", escape_double = FALSE, trim_ws = TRUE)
tasks <- tasks.all[!apply(tasks.all, 1, function(x){any(is.na(x))}),]#remove NAs
questions <- read_delim("questionaire.csv",";", escape_double = FALSE, trim_ws = TRUE)
procontra <- read_delim("pro_contra.csv",";", escape_double = FALSE, trim_ws = TRUE)
```

Plot der gesamten Durchführungszeiten

```
plot(tasks$DURATION, main="Durchführungszeiten", xlab="Task", ylab="Dauer (ms)")
```



Auf Signifikanz zwischen den Räumen prüfen

```
t.test(tasks$DURATION ~tasks$ROOM, var.equal=FALSE, paired=FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  tasks$DURATION by tasks$ROOM
## t = 0.030566, df = 387.77, p-value = 0.9756
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
```

1

```
##  -487.2227  502.6109
## sample estimates:
## mean in group 1 mean in group 2
##        5433.073        5425.379
```

Da der T-Test einen p-Wert sehr nahe an 1 zeigt, kann die Annahme hier verworfen werden, dass es statistische Unterschiede durch die einzelnen Räume gibt. Daher werden alle Datensätze nunmehr gemeinsam betrachtet.

Durchschnittliche Bearbeitungszeit der Tasks

```
mean(tasks$DURATION)
```

```
## [1] 5429.167
```

Median der Bearbeitungszeit der Tasks

```
median(tasks$DURATION)
```

```
## [1] 4702.5
```

Standardabweichung der Bearbeitungszeit der Tasks

```
sd(tasks$DURATION)
```

```
## [1] 2482.354
```

Auf Signifikanz zwischen Geschlechtern prüfen

```
t.test(tasks$DURATION ~tasks$SEX, var.equal=FALSE, paired=FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  tasks$DURATION by tasks$SEX
## t = 0.28689, df = 338.95, p-value = 0.7744
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -433.1859  581.1236
## sample estimates:
## mean in group FEMALE   mean in group MALE
##             5470.324             5396.355
```

Auf Signifikanz zwischen (nicht-)technischen Berufen prüfen

```
t.test(tasks$DURATION ~tasks$TECH_PROFESSION, var.equal=FALSE, paired=FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  tasks$DURATION by tasks$TECH_PROFESSION
## t = -1.4029, df = 385.84, p-value = 0.1614
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -846.4336  141.5013
```

```
## sample estimates:
## mean in group FALSE  mean in group TRUE
##            5254.741            5607.207
```

Auf Signifikanz zwischen Bartträgern prüfen

```
t.test(tasks$DURATION ~tasks$BEARD, var.equal=FALSE, paired=FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  tasks$DURATION by tasks$BEARD
## t = 0.062454, df = 193.43, p-value = 0.9503
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -503.7880  536.7368
## sample estimates:
## mean in group FALSE  mean in group TRUE
##            5433.180            5416.705
```

Auf Signifikanz zwischen Brillenträgern prüfen

```
t.test(tasks$DURATION ~tasks$GLASSES, var.equal=FALSE, paired=FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  tasks$DURATION by tasks$GLASSES
## t = -2.582, df = 280.56, p-value = 0.01033
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1211.1033  -163.2849
## sample estimates:
## mean in group FALSE  mean in group TRUE
##            5156.051            5843.245
```

Wir haben ein statistisch signifikantes Ergebnis zwischen Personen mit und ohne Brille.

Differenz zwischen Brillenträgern

```
glasses <- split(tasks, tasks$GLASSES)
mean(glasses$`TRUE`$DURATION) - mean(glasses$`FALSE`$DURATION)
```

```
## [1] 687.1941
```

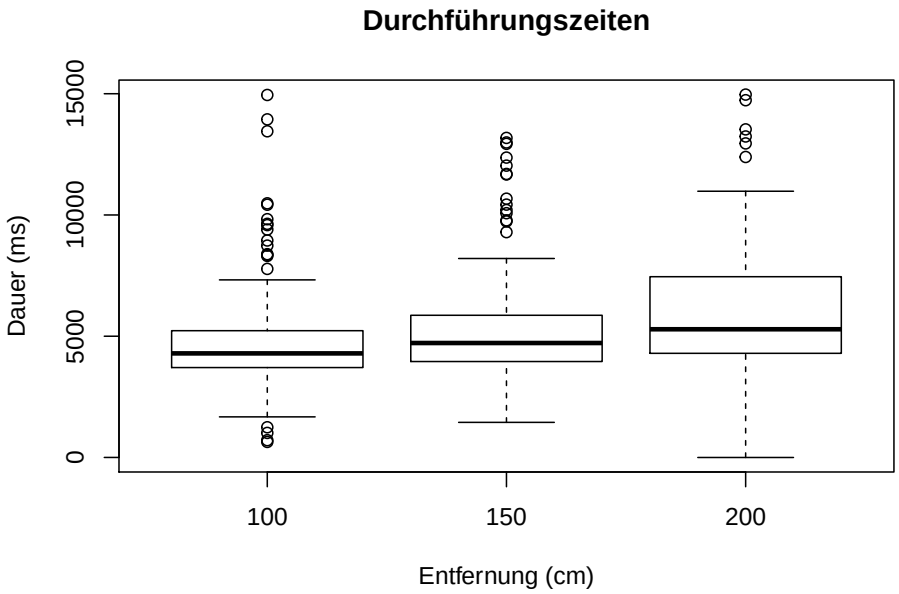
Durchschntl. Durchführungszeiten nach Entfernung gruppieren

```
setNames(aggregate(tasks$DURATION, by=list(tasks$DISTANCE), FUN=mean, na.rm=TRUE),
         c("Entferneung (cm)", "Dauer (ms)"))
```

```
##   Entferneung (cm) Dauer (ms)
## 1              100   4879.779
## 2              150   5395.870
## 3              200   6112.888
```

Box Plot der Dauern gruppiert nach Distanzen

```
boxplot(tasks$DURATION ~ tasks$DISTANCE, main="Durchführungszeiten",xlab="Entfernung (cm)",
        ylab="Dauer (ms)")
```



Wir haben hier eine leichte Tendenz zur Annahme, dass je weiter weg man von der Kamera steht, desto schlechter wird die Head Pose Estimation

ANOVA der Dauern gruppiert nach Entfernungen (100,150,200)

```
distance = factor(tasks$DISTANCE)
distances.aov = aov(tasks$DURATION ~ distance)
summary(distances.aov)
```

```
##              Df    Sum Sq  Mean Sq F value   Pr(>F)
## distance      2 9.543e+07 47714229   8.023 0.000386 ***
## Residuals   387 2.302e+09  5947343
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Tukey HSD - Honest Significant Difference

```
TukeyHSD(distances.aov)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = tasks$DURATION ~ distance)
##
## $distance
##              diff         lwr      upr     p adj
```

```
## 150-100  516.0902 -177.186429 1209.367 0.1875920
## 200-100 1233.1085  507.934781 1958.282 0.0002230
## 200-150  717.0184   -5.732411 1439.769 0.0523807
```

Wir haben einen hoch signifikantes Ergebnis zwischen der Entfernung 100 zu 200 cm. Für die Entfernung 150 zu 200 cm ergibt sich ein Wert von 0.052 der gerade noch nicht signifikant ist. Man kann sagen, dass ab einer Entfernung von 150 cm von der Kamera die Head Pose Estimation immer schlechter wird, je weiter man sich entfernt.

Berechne Anzahl der Variablen der Konfusionsmatrix

```
types <- split(tasks.all$TYPE, tasks.all$TYPE)
tp <- length(types$TP)
fp <- length(types$FP)
tn <- length(types$TN)
fn <- length(types$FN)
cat(" TP:",tp,"\n", "FP:",fp,"\n","TN:",tn,"\n","FN:",fn,"\n")
```

```
##  TP: 381
##  FP: 12
##  TN: 4
##  FN: 53
```

Berechne Recall

```
tp / (tp + fn)
```

```
## [1] 0.8778802
```

Berechne Accuracy

```
(tp + tn) / (tp + tn + fp + fn)
```

```
## [1] 0.8555556
```

Berechne Precision

```
tp / (tp + fp)
```

```
## [1] 0.9694656
```

Berechne False Alarm Rate

```
fp / (fp + tn)
```

```
## [1] 0.75
```

Gruppiere Fragebogen nach Hitrate

```
goodHitrate <- questions[questions$HIT_RATE >=100, ]
badHitrate <- questions[questions$HIT_RATE <100, ]
cat("Anzahl Probanden - gut:",nrow(goodHitrate),"; ","schlecht:",nrow(badHitrate))
```

```
## Anzahl Probanden - gut: 9 ;  schlecht: 21
```

Berechne QUESI Scores für gute Hitrate

```
answersW_good <- c(goodHitrate$Q1,goodHitrate$Q6,goodHitrate$Q11)
meanW_good <- round(mean(answersW_good),2)
answersG_good <- c(goodHitrate$Q2,goodHitrate$Q7,goodHitrate$Q12);
meanG_good <- round(mean(answersG_good),2)
answersL_good <- c(goodHitrate$Q3,goodHitrate$Q8,goodHitrate$Q13)
meanL_good <- round(mean(answersL_good),2)
answersF_good <- c(goodHitrate$Q4,goodHitrate$Q9,goodHitrate$Q14)
meanF_good <- round(mean(answersF_good),2)
answersE_good <- c(goodHitrate$Q5,goodHitrate$Q10)
meanE_good <- round(mean(answersE_good),2)
sumAnswers_good <- c(meanW_good,meanG_good,meanL_good,meanF_good,meanE_good)
meanTotal_good = round(mean(sumAnswers_good),2)
cat(" W:",meanW_good,"\n", "G:",meanG_good,"\n","L:",meanL_good,"\n","F:",meanF_good,"\n","E:",meanE_go
```

```
##  W: 4.48
##  G: 4.44
##  L: 4.44
##  F: 4.48
##  E: 4.44
##  Total: 4.46
```

Berechne QUESI Scores für schlechte Hitrate

```
answersW_bad <- c(badHitrate$Q1,badHitrate$Q6,badHitrate$Q11)
meanW_bad <- round(mean(answersW_bad),2)
answersG_bad <- c(badHitrate$Q2,badHitrate$Q7,badHitrate$Q12);
meanG_bad <- round(mean(answersG_bad),2)
answersL_bad <- c(badHitrate$Q3,badHitrate$Q8,badHitrate$Q13)
meanL_bad <- round(mean(answersL_bad),2)
answersF_bad <- c(badHitrate$Q4,badHitrate$Q9,badHitrate$Q14)
meanF_bad <- round(mean(answersF_bad),2)
answersE_bad <- c(badHitrate$Q5,badHitrate$Q10)
meanE_bad <- round(mean(answersE_bad),2)
sumAnswers_bad <- c(meanW_bad,meanG_bad,meanL_bad,meanF_bad,meanE_bad)
meanTotal_bad = round(mean(sumAnswers_bad),2)
cat(" W:",meanW_bad,"\n", "G:",meanG_bad,"\n","L:",meanL_bad,"\n","F:",meanF_bad,"\n","E:",meanE_bad,"\
```

```
##  W: 4.3
##  G: 4.06
##  L: 4.43
##  F: 4.24
##  E: 3.5
##  Total: 4.11
```

Berechne QUESI Scores gesamt

```
answersW_all <- c(questions$Q1,questions$Q6,questions$Q11)
meanW_all <- round(mean(answersW_all),2)
answersG_all <- c(questions$Q2,questions$Q7,questions$Q12);
meanG_all <- round(mean(answersG_all),2)
answersL_all <- c(questions$Q3,questions$Q8,questions$Q13)
meanL_all <- round(mean(answersL_all),2)
answersF_all <- c(questions$Q4,questions$Q9,questions$Q14)
```

```
meanF_all <- round(mean(answersF_all),2)
answersE_all <- c(questions$Q5,questions$Q10)
meanE_all <- round(mean(answersE_all),2)
sumAnswers_all <- c(meanW_bad,meanG_bad,meanL_bad,meanF_bad,meanE_bad)
meanTotal_all = round(mean(sumAnswers_all),2)
cat(" W:",meanW_all,"\n", "G:",meanG_all,"\n","L:",meanL_all,"\n","F:",meanF_all,"\n","E:",meanE_all,"\
```

```
##  W: 4.36
##  G: 4.18
##  L: 4.43
##  F: 4.31
##  E: 3.78
##  Total: 4.11
```

Wilcoxon Test auf beide Hitrates

```
wilcox.test(answersW_good, answersW_bad)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  answersW_good and answersW_bad
## W = 891.5, p-value = 0.692
## alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(answersG_good, answersG_bad)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  answersG_good and answersG_bad
## W = 1060.5, p-value = 0.04649
## alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(answersL_good, answersL_bad)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  answersL_good and answersL_bad
## W = 814, p-value = 0.7195
## alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(answersF_good, answersF_bad)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  answersF_good and answersF_bad
## W = 1015.5, p-value = 0.1124
## alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(answersE_good, answersE_bad)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
```

```
## data:  answersE_good and answersE_bad
## W = 570, p-value = 0.001219
## alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(sumAnswers_good, sumAnswers_bad)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  sumAnswers_good and sumAnswers_bad
## W = 25, p-value = 0.01091
## alternative hypothesis: true location shift is not equal to 0
```

Struktur der Stichprobe

```
cat("Teilnehmer:",nrow(questions),"[ Ø-Alter:",round(mean(questions$AGE),2),", Min:",
    min(questions$AGE),", Max:",max(questions$AGE),", SD:",round(sd(questions$AGE),2),"]\n")
```

```
## Teilnehmer: 30 [ Ø-Alter: 34.23 , Min: 12 , Max: 61 , SD: 14.18 ]
```

```
tech_profs <- split(tasks.all$TECH_PROFESSION, tasks.all$TECH_PROFESSION)
cat("Technischer Beruf: ",length(tech_profs$`TRUE`)/15,
    "; Nicht-Technischer Beruf: ",length(tech_profs$`FALSE`)/15)
```

```
## Technischer Beruf:  15 ; Nicht-Technischer Beruf:  15
```

```
sapply(split(questions$GRADUATION,questions$GRADUATION),length)
```

```
##              Abitur Hauptschulabschluss  Hochschulabschluss
##                   4                   3                  11
##      Mittlere Reife      ohne Abschluss
##                   9                   3
```

```
sapply(split(questions$PRIVATE_COOKING,questions$PRIVATE_COOKING),length)
```

```
##              1-2 mal pro Monat              1-2 mal pro Woche
##                              6                              6
##              3-4 mal pro Woche Seltener als 1-2 mal pro Monat
##                              4                              7
##      Täglich oder fast täglich
##                              7
```

```
glasses <- split(tasks.all$GLASSES, tasks.all$GLASSES)
beards <- split(tasks.all$BEARD, tasks.all$BEARD)
cat("Brillenträger: ",length(glasses$`TRUE`)/15,"; Bartträger: ",length(beards$`TRUE`)/15,
    "; Beides: ",nrow(tasks.all[tasks.all$GLASSES == TRUE & tasks.all$BEARD == TRUE, ])/15)
```

```
## Brillenträger:  12 ; Bartträger:  8 ; Beides:  3
```

Prozentsatz private Nutzung des Systems

```
cat(round(nrow(split(questions, questions$PERSONAL_USE)$`TRUE`)/nrow(questions)*100,1),"%")
```

```
## 76.7 %
```

Pro

```
table(procontra$PRO)
```

```
##
##             Andere Geräte Steuern            Erleichterte Bedienung
##                                 3                                 7
##                       Frei im Raum                        Hände Frei
##                                 2                                10
##                          Hilfreich Ideal für Leute mit Behinderungen
##                                 2                                 1
##                           Praktisch                           Schnell
##                                 4                                 2
##                               Spaß                 Technikaffinität
##                                 1                                 1
##                     Unkompliziert
##                                 4
```

Contra

```
table(procontra$CONTRA)
```

```
##
##                   Datenschutz        Gefühl der Überwachung
##                             4                             2
##        keine Dialekterkennung                       Kosten
##                             1                             1
##           Messungenauigkeiten                    Spielerei
##                             6                             1
## Unrelevant für Privatgebrauch                   Zu Langsam
##                             1                             3
```

Anhang C Installationsanleitungen

```
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install oracle-java8-jdk
echo export JAVA_HOME="/usr/lib/jvm/jdk-8-oracle-arm32-vfp-hflt/" >> ~/.profile
echo export PATH=$PATH:$JAVA_HOME/bin >> ~/.profile
source ~/.profile
```

Algorithmus 5: Java Installation

```
sudo apt-get install build-essential cmake pkg-config
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get install libxvidcore-dev libx264-dev libatlas-base-dev gfortran
cd ~ && wget -O opencv.zip https://github.com/opencv/opencv/archive/3.2.0.zip
unzip opencv.zip
cd ~/opencv-3.2.0/ && mkdir build && cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D WITH_TBB=ON -D BUILD_SHARED_LIBS
    =ON ..
make -j4
sudo make install
sudo ldconfig
```

Algorithmus 6: OpenCV Installation

```
cd ~ && wget -O dlib.zip https://github.com/davisking/dlib/archive/master.zip
unzip dlib.zip
cd dlib-master/ && mkdir build && cd build/
cmake .. -DUSE_AVX_INSTRUCTIONS=ON
cmake --build . --config Release
sudo make install
sudo ldconfig
```

Algorithmus 7: Dlib Installation

```
sudo apt-get install autoconf bison swig libasound2-dev python-dev
cd ~ && mkdir sphinx && cd sphinx
wget -O sphinxbase.zip https://github.com/cmusphinx/sphinxbase/archive/master.zip
wget -O pocketsphinx.zip https://github.com/cmusphinx/pocketsphinx/archive/master.zip
unzip sphinxbase.zip && mv sphinxbase-master/ sphinxbase
unzip pocketsphinx.zip && mv pocketsphinx-master/ pocketsphinx
cd sphinxbase/
./autogen.sh && make && sudo make install && sudo ldconfig
cd ../pocketsphinx/
./autogen.sh && make && sudo make install && sudo ldconfig
cd swig/java/ && make
```

Algorithmus 8: PocketSphinx Installation

```
%We assume SmartKitchen Content is copied to ~/
wget -O apache-karaf-4.0.7.zip http://archive.apache.org/dist/karaf/4.0.7/apache-karaf-4.0.7.zip
sudo cp apache-karaf-4.0.7.zip /usr/local/ && cd /usr/local/
sudo unzip apache-karaf-4.0.7.zip
sudo chown pi apache-karaf-4.0.7/
sudo ln -sf apache-karaf-4.0.7/ karaf
cp -R ~/SmartKitchen/Implementation/smartkitchen-config/etc /usr/local/karaf/etc/
%In order to start the Apache Karaf Container manually:
cd /usr/local/karaf/bin/
./start
%In order to start the Karaf SSH Console (user,pw=karaf):
ssh karaf@localhost -p8101
```

Algorithmus 9: Apache Karaf Installation

```
%We assume SmartKitchen Content is copied to ~/
cd ~/SmartKitchen/Implementation/smartkitchen-config/
%Customize Variables $WORKSPACE and $KARAF_HOME
sh scripts/deploy_local.sh
```

Algorithmus 10: SmartKitchen Installation

Erklärung zur Urheberschaft

Ich habe die Arbeit selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und bisher keiner anderen Prüfungsbehörde vorgelegt. Von den zu § 27 Abs. 5 der Prüfungsordnung vorgesehenen Rechtsfolgen habe ich Kenntnis. Die vorgelegten Druckexemplare und die vorgelegte digitale Version sind identisch.

Ort, Datum	Unterschrift

Inhalt des beigefügten Datenträgers

/Appliances	Programmcode für die imitierten Haushaltsgeräte
/Bibliography	Literatur in PDF und BibTex
/Documentation	Anleitungen sowie diese Masterarbeit
/Evaluation	Auswertungen und Rohdaten der Evaluationen
/Implementation	Die eigentliche Implementierung in Maven-Struktur
/Material	Projektabhängige Binärdateien und Applikationen
/Native Wrappers	Java-C-Wrapper Projekte
/POCs	Proof of Concept Projekte
/Presentation	Präsentationen