CHALMERS

Voice-operated Home Automation

Affordable System using Open-source Toolkits

Bachelor of Science Thesis in Computer Science

MARIKA HANSSON
EMIL JOHANSSON
JACOB LUNDBERG
MARKUS OTTERBERG

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2013

Voice-operated Home Automation
Affordable System using Open-source Toolkits

M. HANSSON
E. JOHANSSON
J. LUNDBERG
M. OTTERBERG

Examiner: A. LINDE

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2013

Abstract

This report details the development process for a voice-controlled–home-automation system. Manually controlling various devices found in one's home might be tiresome and for others nigh impossible. This system is able to ease a user's interaction with home equipment by providing a new, easy to use, interface.

The initial phase of the development involved surveying the current field by conducting studies on speech-recognition software and the needs of potential users. These studies provided a foundation for the prototype requirements that acted as the goal for the implementation phases that followed.

The report also presents the finished prototype, which is a system capable of interpreting speech as commands. These commands can be formulated in several different ways to make the interaction more natural. After a command has been interpreted an appropriate action is taken on the correct home equipment. In order to control a number of different equipment the control is achieved through radio-controlled outlets. The use of outlets enables the prototype to control most equipment that is controlled with on and off switches. The prototype can also be used with a graphical user interface for any case when the user has problems using his voice.

Keywords:
Speech recognition, Home automation, Sphinx-4, Voice-controlled home, Raspberry Pi

Sammanfattning

Denna rapport detaljerar utvecklingsprocessen för ett röststyrt hemautomationssystem. Att manuellt styra enheter som finns i sitt hem kan vara tröttsamt och näst intill omöjligt för somliga. Detta system klarar av att underlätta en användares interaktion med apparater i hemmet genom att tillhandahålla ett nytt, lättanvänt, gränssnitt.

Den initiala utvecklingsfasen innebar kartläggning av det nuvarande området genom att utföra studier på mjukvara för taligenkänning och potentiella användares behov. Dessa studier tillhandahöll en grund för de krav på prototypen som utgjorde målen för implementationsfaserna.

Rapporten presenterar även den färdiga prototypen vilken är ett system som kan tolka tal som kommandon. Dessa kommandon kan formuleras på flera olika sätt för att göra interaktionen mer naturlig. Efter ett kommando tolkats utförs en lämplig handling på den korrekta apparaten. För att kunna styra ett antal olika apparater utförs kontrollen genom radiostyrda uttag. Användningen av uttag möjliggör kontroll av de flesta apparater som kontrolleras med på- och avbrytare. Prototypen kan också användas med ett grafiskt användargränssnitt för de fall då användaren har problem med sin röst.

Nyckelord:
Taligenkänning, Hemautomation, Sphinx-4, Röststyrt hem, Raspberry Pi

Contents

Chapter 1

Introduction

Speech recognition is a familiar and accessible concept to most people in western society. All that is needed in order to use speech recognition is a smartphone with some form of speech recognition application. Most phones even come with a speech-recognition application pre-installed, with the iPhone's voice-operated assistant Siri being a notable example. Speech-recognition exists in a multitude of different environments other than phones and might be found in some automated phone queues or ticket-booking services to name a few.

Home automation might not be as well known as speech recognition but it is still an area that offers a lot of products and solutions for people that are interested in a more comfortable or manageable lifestyle. Most home systems come in complete solutions with different kinds of features and possibilities. They can manage everything from controlling a television set and lighting to door locks and other security features.

1.1 Background

With voice-controlled home automation everyone can lead a more comfortable life. Running around the house to turn on all the lamps can be tiresome, it would instead be easier to use your voice. For people with disabilities the benefit might be even greater since they can now do things that they, due to their disability, could not do before. This kind of voice-controlled system could improve their quality of life, while decreasing the need of assistance.

The idea of combining speech recognition with home automation into a voice-controlled home system is not new. This concept has been seen multiple times in movies and other popular culture. A cursory look online reveals a multitude of existing solutions. On the one hand there are the commercial home-automation products using speech recognition with the goal to be user friendly. On the other hand there are the home-built prototypes with the goal to be inexpensive. Both types of voice-controlled home systems are important but there is a lack of combination. This project focuses on finding a solution to combine these two types.

1.2 Purpose

The purpose of the project is to develop a prototype that can be used for controlling home electronics using voice commands. Based on the project background the prototype should aim to fulfil the following properties:

- Affordable: The use of inexpensive hardware and software is prioritised.
- Ease of use: The user should not need deep technological knowledge to use the prototype.
- Functional: It should not be more difficult to control devices using the prototype than with more conventional methods.
- Relevant functionality: The prototype should be able to perform tasks that are of interest to the user.

- Limit energy consumption: The prototype should be able to help the user limit the energy consumption in their home.

1.3 Delimitations

Automatic Speech Recognition is a heavily researched area with a multitude of well-functioning tools and frameworks, some of which are discussed in section 3.2. It would be extremely difficult to develop software that offers the same level of functionality as existing ones. For this reason the project will instead focus on using established software for speech recognition and adapt them to the needs of the prototype.

1.4 Method

The purpose of the project is mostly focused on the user experience. Therefore, a user study was conducted in order to validate the project vision and formulate the prototype requirements have the correct focus. Since the field of speech recognition offers multiple software solutions it was necessary to conduct a study in order to analyse them with respect to the desired functionality of the prototype.

1.4.1 User study

A user study was conducted to gain a deeper understanding of what demands a potential user might have. This study helped ensure that the prototype's features were relevant for those users. The study was conducted in an early stage of the project in order to help ensure the correct focus of the development and which features to prioritise.

In order to obtain a good sample size for the survey it was distributed to targeted online communities that might have a special interest in or need for the product. We thought that these demographics would be more inclined to answer the survey, which would result in higher participation. The survey was distributed on the website Reddit [1], which is an online community and news site where the users supply the content. The content can then be voted on by the users to make the popular posts more visible. The site is divided into smaller communities, known as subreddits, that focus on a particular subject. These subjects can range from something as wide as news or sports to more narrow topics, such as a particular author or the game of chess. The particular subreddits that was targeted for distribution were mostly for different kinds of disabilities and home automation. The survey was also posted to a larger, general-purpose subreddit in order to attract a wider audience.

1.4.2 Study of speech-recognition software

Since speech recognition is a broad field, with a multitude of frameworks and toolkits, an evaluation of these software solutions was made. The purpose of the evaluation was to analyse which software would be best suited for the prototype. Aspects taken into consideration were: programming language of the software, availability of models and the licensing conditions of the software. An additional aspect that was evaluated, though not necessarily considered a deciding factor, was the availability of support from the community.

Chapter 2

Technical background

Section 2.1 covers a brief history of the speech-recognition technology and how speech recognition works. The technology of home automation will be introduced in section 2.2.

2.1 Introduction to speech recognition

Speech recognition, often referred to as Automatic Speech Recognition (ASR), is a technology in which human speech is processed into text. Most people today probably associate speech recognition with computers. However, the first attempts to process speech signals were in fact done mechanically [2].

2.1.1 History of speech recognition

The first trial of speech recognition was performed at Bell laboratories in 1952 and used spoken digits from a single speaker to isolate the digits in the recording [3]. At this stage the recognition was done by measuring frequencies during vowel regions of the digits. The idea of a phoneme recogniser was implemented at University College in England in 1959 by defining phoneme sequences, which increased the accuracy of words consisting of multiple phonemes. Specialised hardware was built for speech recognition since computers still did not have enough computational power in the early 1960s [3].

In the 1970s computers started to have high enough performance to handle speech recognition and multiple solutions using dynamic programming were invented. In this era, key concepts as *template-based isolated word recognition*, using static patterns in sentences, and *continuous speech recognition*, using dynamic tracking of phonemes, surfaced [3]. The 1970s was also the period when speaker-independent speech recognition started to be researched at the AT&T Bell Labs [3]. This technique is a large part of speech recognition as it is known today.

An important and frequently used approach to speech recognition is to use hidden markov models (HMM), which was first introduced in the 1980s [3]. The use of HMMs have been successful due to the simple and effective way they model the speech waveform into time-varying vectors for processing [4]. Another important approach is to use N-gram models, which try to predict the Nth word using the previous N-1 words.

In the 1990s and 2000s the research focused more on improving the algorithms with respect to error minimisation. Another focus was the concept of model training in order to improve accuracy [3].

2.1.2 How speech recognition works

The process of speech recognition can be generalised to four stages, as seen in figure 1. Since there are multiple ways of recognising speech this generalisation does not fully apply to all implementations but represents an overview.

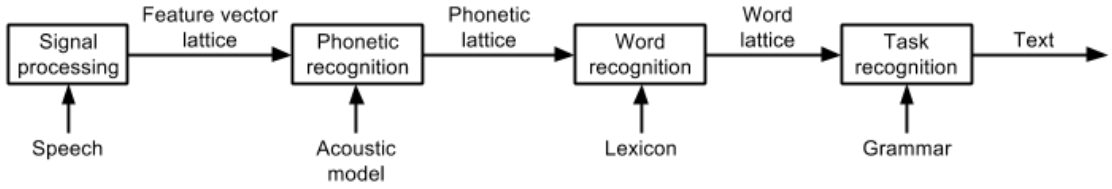


Figure 1: The general process of speech recognition. The speech input is translated into a series of feature vectors. The feature vectors are then transformed to the corresponding phonetic representation. The phonetic representation is translated into words and checked against a set of rules in the form of a grammar. The output of these steps is the digitised representation of the speech input in form of the corresponding text.

The first step is to process the speech input into feature vectors. Feature vectors are observed parts of speech broken down into smaller blocks and serve to keep relevant acoustic information from the original sound [5]. All sounds can be represented with a sound wave that has specific pattern. These patterns are the acoustic information that the feature vectors represent.

The next step is to interpret the feature vectors in combination with the acoustic model. The acoustic model contains acoustic properties of each distinct sound and the phonetic representation of that sound [6]. In this step the acoustic model is used to map the lattice of feature vectors to the corresponding phonetic representation.

After the phonetic lattice has been generated the task is to find the words that corresponds to the phonetic combinations. This task is performed using a lexicon, also known as a dictionary [5]. An example lexicon is shown in listing 1. The output from this task is a lattice with the corresponding words found.

Listing 1: Example lexicon with phonetic representation of a set of words. Words which has multiple ways of pronunciation becomes numbered.

```
ACTIVATE      AE K T AH V EY T
BATHROOM      B AE TH R UW M
BEDROOM       B EH D R UW M
CURRENT       K ER AH N T
CURRENT(2)    K ER N T
CURRENT(3)    K AA R AH N T
```

The word lattice is then processed using a grammar in order to find the final text representation. A grammar is a rule-based language model which consists of a collection of rules that state valid word sequences in the language. In speech recognition the grammar is used to validate that the word sequences in the word lattice are correct. It evaluates the word lattice and outputs the sequence that is most likely to be associated with the speech input [5].

Instead of a grammar a statistical language model can be used. A statistical language model has probabilities associated with each word sequence as opposed to the grammar that uses rules. The probabilities state the likelihood of a certain word succeeding the previously observed word sequence.

Since speech recognition is a model of human hearing, the process can also be described as the steps the human brain uses to understand speech. The waveform reaches the ear sending a signal to the brain of incoming sound which corresponds to the signal processing. As the brain becomes

aware of the sound it starts to map it to phonetics. This mapping is modelled as the phonetic recognition. Using the phonetics the sound was mapped to the brain tries to understand which words the phonetics correspond to. This step is modelled as the word recognition. The last step is to give the words context, which is called the recognition step. The output from the brain is the interpretation of the sentence that was spoken, similarly to the digital output from the recogniser.

2.2 Introduction to home automation

The technique of making one's home more automated is called home automation. This automation is done by connecting multiple functions, for example heating, air conditioning, locks etc, to a centralised computer. These functions could then be accessed by the user from a control panel for manual control or they could be controlled automatically by the system. In some systems it is also possible to connect through the internet, to enable access from any device that have internet access.

One of the first automated home systems was built by an engineer named Jim Sutherland, in 1966 [7]. The system was called ECHO-IV and was constructed with parts that Jim Sutherland brought home from his work. The home system was able to store recipes, turn appliances on and off and control home temperature, among other functions.

Since then the development of cheaper and smaller electronic equipment have made the process of making one's home more automated an easier task. Today there exists a multitude of options for home automation. They can be divided into two main categories; either the homeowners can construct the home-automation system themselves, or they can purchase a complete system from a dealer, for example the control4 system [8]. As this project aims to set up home automation with stand-alone products only techniques that a homeowner can use themselves is presented here. The book Smart Home Automation with Linux [9] contains more thorough information about techniques for building a home-automation system.

One of the more easy and effective techniques for controlling devices is to use WiFi-connected outlets. These will connect to the homeowners regular router, which means that little installation is needed in order to use the outlets as opposed to other techniques. The outlets will be accessible from any device with an internet connection. The downside with these outlets is that they are fairly expensive relative to other alternatives.

Another technique is to use outlets controlled by Radio Frequency (RF). These outlets require about the same installation procedure as WiFi-controlled outlets, the difference is that to be able to control the outlets there has to be some kind of RF transmitter present. In a larger house, there might be necessary to use multiple transmitters due to range restrictions of the transmitter.

Chapter 3

Establishing requirements

An important part of developing a prototype is establishing requirements needed to achieve the desired functionality. The outcomes of the studies together with the initial project vision and purpose were analysed in order to form the prototype requirements, which are presented in section 3.3.

3.1 Outcome of user study

The project vision was quite clear from the start but in order to ensure that the vision was shared with potential users a user study was conducted using Google Drive Forms. Some important findings from the study are discussed below. The study is presented in its entirety, together with the results, in appendix A.

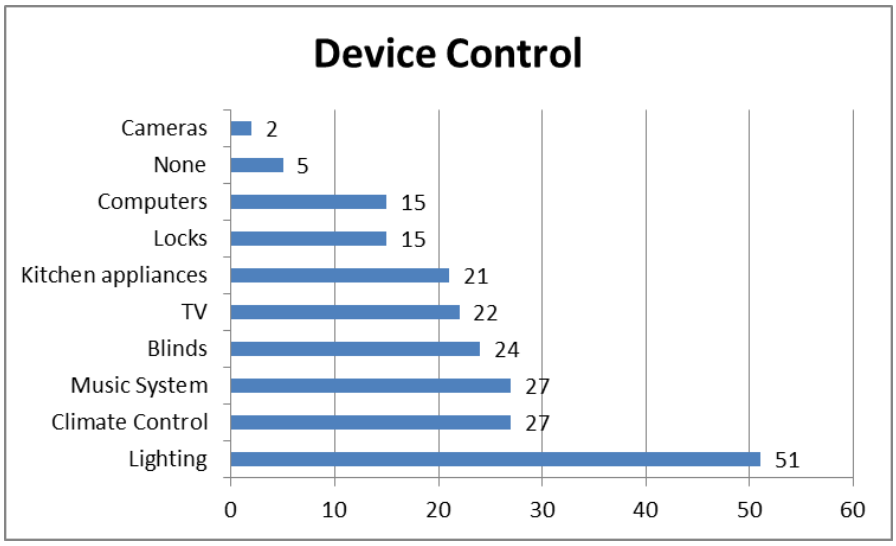


Figure 2: The answers to the question "If you could control equipment in your household solely by using your voice, which would you be most interested in controlling?"

Potential users participating in the user study gave a wide variety of answers as to what equipment they wanted to be able to control with their voice, shown in figure 2. Some of the users wanted to control most of their equipment, and some wanted to control a minimum amount. The most common answers was to be able to control lighting, music systems, climate control and blinds.

Figure 3 shows that other important functionalities were to be able to schedule events and reduce the energy consumption in the home. The answers to this question was overall rather diverse, which shows that the users have a more personal view of what functionality is important.

When it came to the question of what would be the most important functionality in the system the answers converged a lot more as can be seen in figure 4. It was evident that the respondents

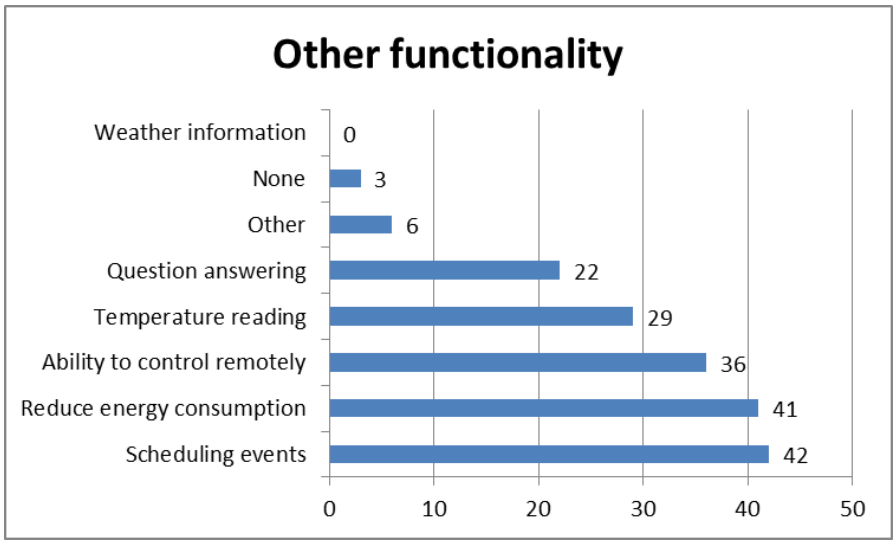


Figure 3: The answers to the question "Which other functions would you want to have in a smart home system?"

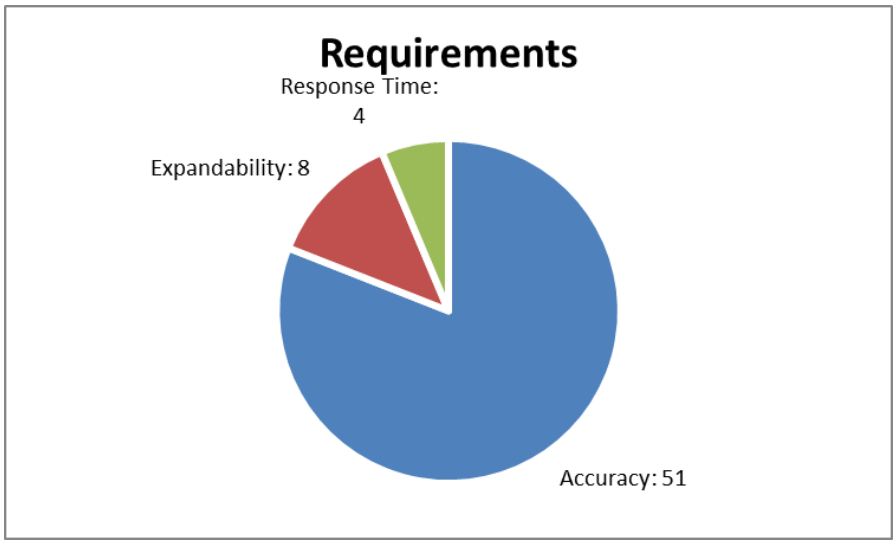


Figure 4: The answers to the question "What would be the most important aspect for you regarding functionality?"

strongly agreed that accuracy of the speech recognition was most important. The system needs to be able to understand users with a high accuracy to reduce the risk of having to repeat a request.

The users had a rather shared view on what a reasonable price for a speech-controlled home system would be. This reasonable price, according to most respondents, was approximately 200 USD. One price example of what an existing system can cost is around 600 USD for each voice unit [10], this price is excluding the needed home-automation system. As shown in figure 5 the respondents answered that most are not willing to pay that much.

The overall outcome of the user study did not present results that diverged noticeably from the initial vision of the project. Therefore, the initial goals were kept intact.

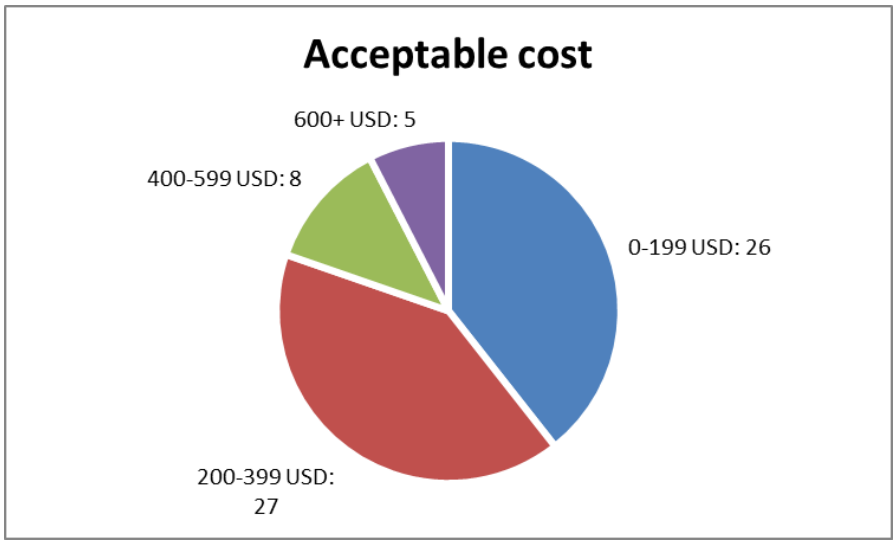


Figure 5: The answers to the question "How much would you be willing to spend on a voice controlled smart home system?"

3.2 Outcome of study on speech-recognition software

The study was conducted on the leading open-source software for speech recognition: HTK speech-recognition toolkit (HTK) [11], CMU Sphinx [12], Julius [13] and Kaldi [14]. It was discovered that all software had little active development on their code base. This discovery made the existence of an active community an important factor since it shows that the software is still relevant. In order to motivate the choice of software all aspects presented in subsection 1.4.2 will be discussed separately and then combined for the final decision.

3.2.1 Programming language

The studied speech-recognition software is implemented in three different programming languages. HTK and Julius are implemented in C, Kaldi in C++, while Sphinx supports both C and Java with two different frameworks, PocketSphinx and Sphinx-4. Since all members in the project group were already well versed in Java, Sphinx-4 was the preferred software in this aspect.

3.2.2 Availability of models

When performing speech recognition some models are needed in order to interpret the speech input. The system needs an acoustic model and either a statistical language model or a grammar. In order to create acoustic models with high quality a large amount of speech data is needed [15]. Since not enough resources for creating new models were available during the project the existence of working models was very important.

Most of the speech-recognition software aimed towards research focuses on enabling the use of multiple types of models rather than providing existing ones. HTK is mainly focused on research of hidden markov models in speech recognition [11] and Kaldi is focused on research of acoustic models [16]. Since most users create their own models during their research there is limited need to provide any.

Sphinx on the other hand provides multiple highly-trained models. Guides for further adaptation and training of these models are also available. Julius has satisfying models, though the acoustic models are not in English which made the usefulness of this software rather limited for this project.

There exist some models that can be used with most speech-recognition software but since Sphinx has software-specific models it was preferred over the others.

3.2.3 Final outcome

After analysing all aspects, we decided to use Sphinx-4. Below follows a short introduction to how the framework is implemented.

Sphinx-4 consists of three main modules: the front end, the decoder and the knowledge base, as seen in figure 6. Each of these parts are built with modularity in mind, so that the modules can be replaced depending on the needs of the developer.

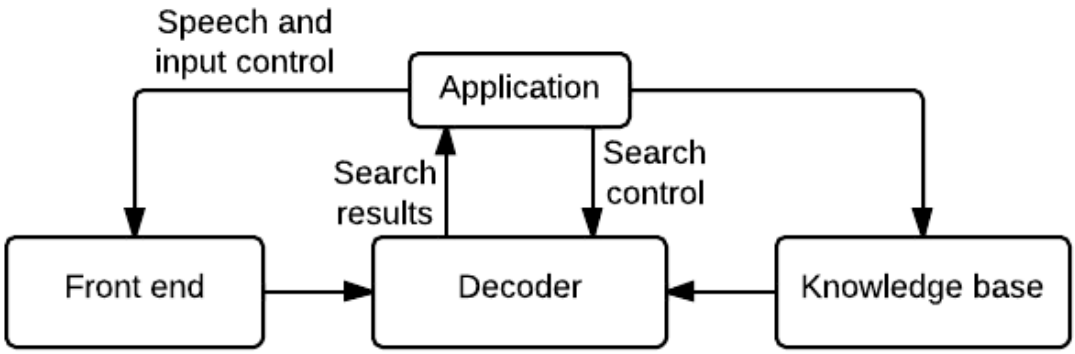


Figure 6: An overview of the Sphinx-4 module structure. The frontend is responsible for computing feature vectors based on the speech input. The knowledge base holds the acoustic model, language model and dictionary to be used by the decoder. The decoder constructs search graphs and computes probabilities for each state based on the models, as well as performs the search.

The front end computes feature vectors using speech input. Each step in the process of computing feature vectors are built as independent and replaceable blocks. This block design makes it easy to modify the way that feature vectors are computed [17].

The knowledge base consist of the models needed to perform speech recognition, such as acoustic models, language models and dictionaries [17].

The decoder is responsible for the actual recognition. In order to perform recognition the resources in the knowledge base are used to construct a search graph. These resources are also used to compute probabilities for each state in order to be able to perform a search. The last task of the decoder is to perform the actual search. Based on the search graph and the probabilities associated with each state the search is performed [17]. When an end state is reached the decoder terminates and returns a result.

3.3 Prototype requirements

Based on the outcomes of the studies the requirements for the prototype were formulated, which define how the purpose should be achieved. The requirements are of varying importance in consideration to the functionality of the prototype. The list of requirements in decreasing order of importance are:

- Understand a small set of defined commands.
- Always ready for new commands.
- Be able to control a device.
- Motion sensing.
- Sleep-mode functionality.
- Status indication, e.g. with a LED-light.
- The prototype should give a response after a command.

Chapter 4

Prototype development

This chapter details the development of the prototype starting with a description in section 4.1 of the agile process used and the software structure of the prototype. After that an overview of the used software tools follows in section 4.2. The chapter ends with section 4.3, which contains a description of the hardware components of the prototype.

4.1 Development process

An agile process, with multiple phases, was used during the development of the prototype. During each phase a number of functions was implemented and tested. The structure of the functions in each phase were based on the order of the requirements presented in section 3.3. The requirements were divided into three phases, where phase 1 consists of requirements that had to be reached in order for the project to be considered successful. The subsequent phases dealt mostly with improving performance and usability.

- Phase 1
  - Understand a small set of defined commands.
  - Always ready for new commands.
  - Be able to control a device.
- Phase 2
  - Improve accuracy.
  - Add a motion sensor.
  - Add sleep-mode functionality.
- Phase 3
  - Status indication, e.g. with a LED-light.
  - The prototype should give a response after a command.

During the development the main focus was on modularity in the source code. The goal of the project was to develop a system with modules that should be easy to exchange. To this end, the source code was divided into four modules; speech recognition, core, client application and hardware controller.

4.2 Programming language and version control

Java uses a virtual machine for executing its programs making them platform independent, which enables a large part of the prototype to be developed for any platform. Another reason for using Java is that the project members are well versed in this particular programming language. Furthermore Sphinx-4, the framework chosen for this project, is implemented in Java.

A negative aspect of using Java is the rather large amount of overhead it introduces which might negatively impact the performance. By using a lower-level language, such as C, some overhead could be removed. This issue was not deemed as critical, instead the focus was on developing solid code that maximise the potential of the language rather than minimising overhead. After considering both the positive and negative aspects we decided that Java would be suitable for this project.

When it came to the programming language used to write the grammar we chose Java Speech Grammar Format (JSGF). The main reason for this choice was that the syntax of JSGF is similar to that of Java and therefore easy for the group members to adapt to.

To enable our group to work on the project independently a good version-control system was needed. The choice for the project was to use GIT [18].

4.3 Hardware setup

Due to the nature of the project a rather large amount of different types of hardware was needed. The hardware ranges from the computing platform used to run the software to the microphone used to record a user's speech. The following subsections aims to provide an overview of the hardware used throughout the project as well as provide arguments for why any specific hardware was used. In order to fulfil the affordability property price was an important factor when considering any hardware. The overall cost of the hardware setup was in the range of 170-190 USD depending on the place of purchase.

4.3.1 Hardware platform

The device used as a platform for the prototype had to be quite cheap, while still providing the necessary computing power in order to enforce the affordability property. Furthermore, the device should not be a disturbance to the user. An ordinary desktop computer provides the necessary capabilities but could be hard to find a place for. This criterion extends further than just physical space, a user should not constantly be reminded by the presence of the device, e.g. by it creating a lot of noise.

These factors led to us choosing a small single-board computer as they are inexpensive while still providing adequate performance. Their compact form factor also makes them unobtrusive to the user. There exists several products to choose from but the final decision was to use a Raspberry Pi, specifications presented in appendix B, for primarily two reasons. The Raspberry Pi has good capabilities for attaching a number of varying sensors and transmitters, which is a desirable property for controlling home equipment. The other reason was the active online community that exists for the Raspberry Pi which we identified as a large potential source for help during development.

The prototype should be fast and perform well which makes an operating system (OS) that is lightweight and heavily configurable optimal. Based on this criterion the ARM version of Arch Linux [19] made the best fit. On the other hand the prototype also needs to be easy to set up and use. This characteristic is something that Arch is not known for and therefore another OS might be more suitable. In this regard we found Raspbian [20], which is a Debian port optimised for the Raspberry Pi, to be the preferred OS.

Ultimately, we decided to leave this decision to the user by not developing for any one particular OS. On the prototype Arch was primarily used in order to have more control over the environment, but the software was also tested periodically on a Raspbian installation. This way a user that is experienced in UNIX environments could use Arch to optimise performance. Someone with less experience could use the more user friendly Raspbian, or another OS of their choice

4.3.2 Peripheral hardware units

Due to the time frame of the project we decided that the prototype should focus on one type of device to control. The type of device chosen was remote-controlled outlets as they are easy to set up and can be used to control a large variety of different home equipment. More specifically the prototype uses a set of three self-learning–radio-controlled outlets, see specification presented in appendix B. We chose these particular outlets based on several advantages. The most important of these advantages was that the protocol used by the remote for sending messages to the outlet is available online [21]. The fact that it is radio controlled means that it can be placed more freely than if it were controlled by IR signals. Another viable option would be to use WiFi outlets but since they are more expensive than radio-controlled outlets they did not fit the projects purpose.

In order to control these outlets a radio transmitter needs to be connected to the Raspberry Pi. A transmitter module was found that correspond well with the overall requirements of the project. It is small, has low energy consumption and operates at the correct frequency (433.92 MHz). Full specifications of the transmitter can be found in appendix B. A short piece of connecting wire was used as an antenna in order to amplify the signal.

A very essential device needed for this project was a microphone in order to capture the speech of the user. The choice of microphone can have large effects on the overall performance since poor sound quality might result in low recognition accuracy. Ideally for the prototype one would like a microphone that can capture speech from anywhere in a room, thus removing the need for the user to be mindful of the positioning of the microphone. These microphones are however quite expensive and since this project aims to create an affordable product we decided to instead use a somewhat lower-end microphone. The microphone used was a Logitech Rock Band USB Microphone, as it had sufficient quality.

The prototype should be able reduce its energy consumption by interrupting the speech-recognition process while no user is around to give commands. To achieve this interruption a motion sensor was used to detect motion in the vicinity of the prototype. This ability was used to decide whether or not there is a user present to use the prototype. The motion sensor we selected, specifications presented in appendix B, was chosen for being relatively low priced, highly compatible with the Raspberry Pi and unobtrusive due to its small size.

Chapter 5

Result: implementation and testing

The results produced during the project can be categorised into two parts: the implementation and the accuracy results. The implementation of the prototype will be explained with an overview in section 5.1. The internal modules of the prototype will be discussed in more detail in sections 5.2 through 5.6. The results from the accuracy tests of the prototype will be covered in section 5.7.

5.1 Implementation of the prototype

In order to have modularity in the source code it was divided into four modules, highlighted with frames in figure 7. Each module was, to the highest possible degree, implemented as a black box. This structure enables developers to exchange modules without the need to make large changes in the overall system. The four modules are: speech recognition, core, client application and the hardware controller. A complete diagram of the functions in each module, as well as the flow of the system, is shown in figure 7.

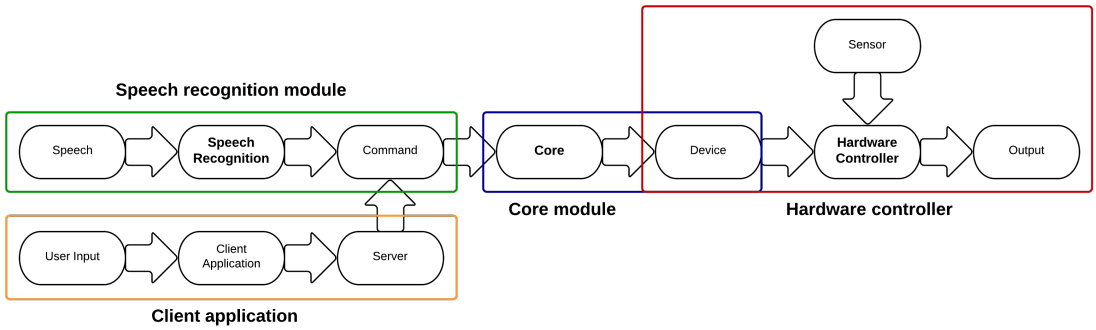


Figure 7: The prototype's functionality and overall flow, divided by frames into the four modules.

5.2 Implementation of the speech-recognition module

As shown in figure 8 the speech-recognition module receives speech as input and returns an abstract representation in the form of a Command object. A description of the Command object can be found in section 5.4.

In order to generate a command, the speech needs to be recognised. This process uses the concepts and models shown in the middle element in figure 8. How to handle invalid speech input, known as out-of-grammar rejection, will be covered in subsection 5.2.1, together with accuracy and response time of the module. Acoustic models, dictionaries and grammars will be discussed in subsection 5.2.2. The process of generating suitable output based on the recognised result will be covered in subsection 5.2.3.

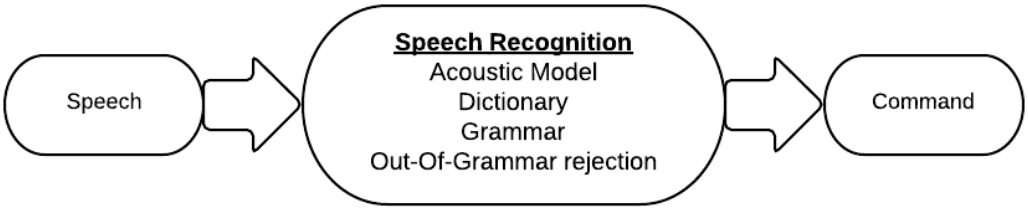


Figure 8: An overview of the speech-recognition module. The input is received as speech and the output is returned as a Command object. The main concepts and problems covered in the speech-recognition module is listed in the middle element.

5.2.1 Requirements for the recognition module

One important requirement was that the prototype should be able to interpret speech. A first step for satisfying this requirement was to decide whether to use a grammar model or a statistical language model. A grammar model offers the ability to create rules for specific sentences, which accommodate the form of input used in a home-automation system.

A well functioning speech-recognition implementation needs to be able to recognise speech input with a satisfying accuracy. One task with the implementation of the speech-recognition module was to improve the accuracy throughout the development process. This improvement was performed using adaptation of the acoustic model which is discussed in further detail in subsection 5.2.2.

Response time was also an important aspect. In the speech-recognition module response time refers to the time elapsed between receiving speech input and generating a Command object. For the case when the speech input was not detected by the microphone the prototype was unable know whether a command was spoken or not. A problem in this case was that the program was unable to find a result during the search, which resulted in the decoder not terminating since no end state in the search graph was reached. The solution was to assume that if a command is spoken, speech input will be detected within a certain time limit. If speech input was not detected a timeout would be issued. The time limit ensures that the maximum response time of recognition is the value of the time limit.

One problem that was found towards the end of the development process was that out-of-grammar input needed to be rejected. The accuracy with valid input was satisfying, discussed in subsection 5.7.1, but one problem was that Sphinx-4 gave recognition result for all input and not just valid input. The reason for this behaviour was that the decoder always returned a sentence even if the result did not correspond to the speech input. To solve this problem an out-of-grammar result was returned when no path in the search graph had a probability above a certain threshold.

5.2.2 Development of models

A decision that had to be made was whether to use a rule-based language model, also know as a grammar, or a statistical language model in order to describe the sentences the prototype should be able to understand. Grammars are usually created manually and used when interpreting commands or smaller sets of speech. A tool is used to generate a statistical language model based on written sentences that the system should be able to understand. Statistical language models can handle a greater amount of variance in the word order of a sentence.

We decided to write a grammar, using Java Speech Grammar Format (JSGF), specialised for the prototype's needs. This decision led to an increased control over which speech inputs were considered valid. Another compelling advantage was that the grammar could include tags. The tags enabled the use of scripts that allowed for manipulation of a Command object when certain words were spoken. This solution removed the need to parse the piece of text received as output from Sphinx-4 in order to construct commands that could be passed on to the core module.

Another decision was to adapt the standard US English model included with the Sphinx-4 software rather than creating a new acoustic model. Creating a new acoustic model requires training of said model. In order to train an acoustic model for Sphinx-4 you need one to five hours of recorded speech data as well as about one month for adjusting parameters in order to create an optimised model [15]. However, adapting the original model only requires about five minutes of recorded speech in order to increase the accuracy of the model [22].

The adapted acoustic model improved the accuracy when a Swedish user pronounces the English words that make up the commands. Another reason for the adaptation was to enhance the accuracy for the different combinations of words specified in the grammar. This enhancement was achieved by recording sentences and writing down the transcript of what was said in each recording. These recordings were then incorporated into the standard acoustic model, with the use of software included in the Sphinx-4 framework, using the guide on the CMU Sphinx website [23].

The last important model, the dictionary, was generated using the Sphinx knowledge-base tool [24]. A text document containing all valid sentences was used and the tool generated a corresponding dictionary specialised for the prototype.

5.2.3 Parsing of recognition results in the grammar

The prototype consists of two general concepts, speech recognition and home automation, and in order to connect these two the speech-recognition module had to produce some sort of relevant output. Two techniques to produce relevant output arose during the development, both executed the parsing of the recognition result in the grammar.

Initially, the output consisted of the recognised sentence in the grammar without any modification. Since speech contains filler words that are not of interest for our prototype, crude filtering of the output from the grammar was needed. Tags were used to remove the need for filtering the output [25]. The tags returned words in the recognised sentence that was of interest and discarded the rest, as an example "please turn on the lamp" was parsed to "on lamp". As seen in listing 2 the tags return simple strings and no modification of the word order is done. A tag is defined using curly brackets, e.g. "{tag}". When tag parsing is performed, the content inside the curly brackets is processed.

Listing 2: Snippet from the grammar file with string tags for output parsing. The tags in this snippet are "{enable}" and "{disable}"

```
<onSwitch> = ([Turn] On | Enable | Activate) {enable};
<offSwitch> = ([Turn] Off | Disable | Deactivate) {disable};
```

To generate a consistent word order, which makes it easier to interpret the output, scripts were added to the tags. With the use of a command object the scripts in the tags could add the correct value to the parameters of the object and thereby remove the word-order constraint. This implementation of scripted tags are shown in listing 3. For a complete implementation of a grammar using scripted tags see appendix C.

Listing 3: Snippet from the grammar file with scripted tags for output parsing. Parameter appObj refers to the command object.

```
<onSwitch> = ([Turn] On | Enable | Activate) {appObj.addAction("enable");};
<offSwitch> = ([Turn] Off | Disable | Deactivate) {appObj.addAction("disable");};
```

5.3 Implementation of the client-application module

To expand on the usability of the prototype other means of controlling it was desired. Therefore a client application was implemented which enabled control with a graphical interface from another computer, over a network. This implementation was made due to two main reasons. The first was that if a user is unable to use his voice, perhaps due to a cold, the prototype should still be usable. The second reason was to give a user the functionality to control his home from anywhere.

A diagram of the client application's functionality is shown in figure 9. The server runs on the Raspberry Pi. The graphical user interface (GUI) and the client application runs on any other device that the user wants to use to control equipment.

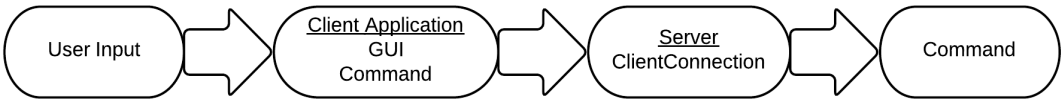


Figure 9: User input is generated through a GUI and stored in a Command object. This object is then transferred through a network and the specified action is performed.

5.3.1 Graphical user interface

The initial view shown when starting the Graphical User Interface (GUI) is a login screen, figure 10. The user is able to input which IP address to connect to, and on which port. After a connection has been established the GUI changes to the device-controller view, shown in figure 11, where controllable devices are displayed. The possible actions are displayed as buttons on a specific device. When a user presses one of these buttons, the name of the device is stored in the Command object as well as the chosen action. The Command object, described in section 5.4, is then passed on to the client.

The connection with the server is established through a TCP-connection. When the connection is made the GUI changes to a device-controller view, where the user has the ability to control connected devices. A user input through the GUI triggers the transfer of a client side Command object to the server with the appropriate parameters stored.

5.3.2 Server

The server runs on the Raspberry Pi and is continuously ready for new connections. There exists functionality to receive multiple connections at the same time so that multiple users are able to control devices over different connections.

After a connection request is made, the server will create a new thread that runs the actual connection. This thread will listen for information from the client application and execute the appropriate action depending on what information it receives.

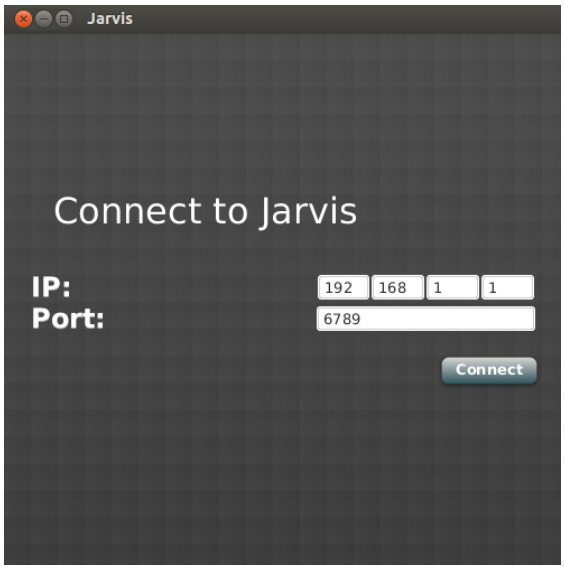


Figure 10: Login screen for the client application. The client connects to the IP address of the server application running on the prototype.

Figure 11: Controller view of the client application. A list of available devices are shown. The user is able to control outlets by pressing a button, marked with enable or disable.

When the server receives a client Command object, it will have to instantiate a new Command object that has access to the Core class, see section 5.5. The reason for this behaviour is that the client application is not run on the prototype. The server will copy variables from the client-side Command object to the newly created server-side Command object, and finally the new object will send its variables to the core module.

5.4 Implementation of the Command class

The Command class is the bridge between the speech recognition and core modules. A Command object has string variables for device, action, position and parameter. The variable called device is used for recognising which device is supposed to be controlled. The action variable is mapped

to the desired method that should be executed on the device. To be able to control devices that are of the same type, e.g. lamps, that are positioned in different places the position variable can be used. The last variable, parameter, is a variable that could be passed on as an input parameter for the desired method, e.g. if the user wants to raise the volume on the stereo with a specific value, that value would be stored in the variable parameter and passed on to the desired method.

The results from the speech-recognition module is temporarily stored in a Command object. When a command is recognised, a call to the method generateCommand is issued and the Command object will pass on its variables to the Core class. This relationship is visualised in figure 12.

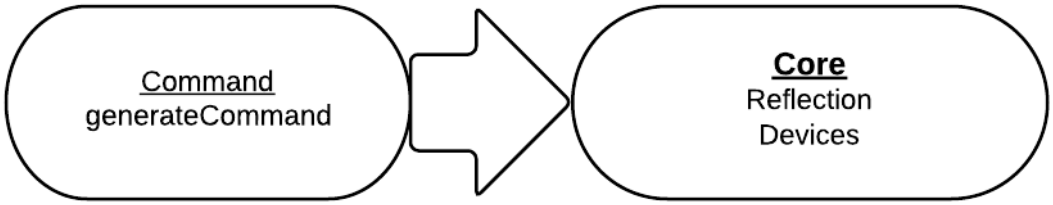


Figure 12: When a call to the method generateCommand on an Command object is made, the object will pass on its variables as parameters to the method controlDevice in the Core class.

5.5 Implementation of the core module

The Core class acts as the connection between speech recognition and the hardware controller. It was implemented using the singleton pattern so that only one instance of the core module communicates with the hardware controller. The core module has the responsibility of instantiating device objects, described in 5.6.2, that can be controlled. These devices are stored in a hashmap for easy retrieval.

In order to facilitate a large amount of different devices and actions a proper way to interpret strings was needed. A common implementation in for instance text-based–role-playing games is to use case-switches with enumeration as seen in listing 4. Case-switches work well if there is a small sample of commands that needs to be interpreted, e.g. north, south, west, east. It would, however not be feasible for a greater set of available commands, where a need for a dynamic way of executing the appropriate action based on a given string is evident. The solution to this problem was resolved in the prototype by using reflection.

Listing 4: Example of using case-switch implementation with enumeration.

```
public enum Direction {NORTH, SOUTH, WEST, EAST};
Direction dir = Direction();

switch(dir) {
    case NORTH:
        System.out.println("Moving " + Direction.NORTH);
        break;
    case SOUTH:
        System.out.println("Moving " + Direction.SOUTH);
        break;
    case WEST:
        System.out.println("Moving " + Direction.WEST);
        break;
    case EAST:
```

```
        System.out.println("Moving " + Direction.EAST);
        break;
    default:
        System.out.println("Bad input");
        break;
}
```

Reflection enables a program to invoke methods on a class during runtime without the need to specify which method to invoke at compile time [26]. Instead it compares a given string to all available methods on that specific class and retrieves the desired method. The retrieved method can later be invoked by the program to execute the desired action.

The Core class extracts the name of the device from the Command object and retrieves the device object from a hashmap. After this step, reflection is used to map the string variable called method to available methods on the specific device. The last step is to invoke the desired method on said device. The actual implementation can be viewed in listing 5.

Listing 5: Use of reflection in the Core class

```
AbstractDevice dev = devices.get(device); // obtain a device from the hashmap
Method method = dev.getClass().getDeclaredMethod(action, new Class<?>[0]);
method.invoke(dev);
```

5.6 Implementation of the hardware-controller module

The hardware controller is the part that bridges the prototype's software with the various devices connected to the Raspberry Pi. It is in this module that the action corresponding to a certain spoken command actually takes place.

Figure 13 illustrates how the various components of the hardware controller interact with each other. The core module creates the virtual devices, which refers to the software abstraction of the controllable devices. These virtual devices contain the logic for controlling the physical device they represent using the Controller class. A motion sensor can be connected to the device in order to provide an additional source of input.

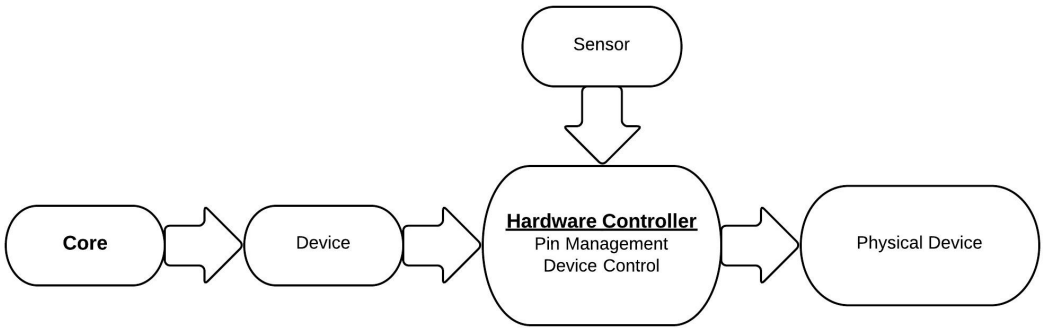


Figure 13: The Core class initiates all devices which in turn control the physical devices through the Controller class. A sensor can provide additional input to the hardware controller.

5.6.1 The Controller class

The hardware controller is the only module that was developed for a specific platform and will therefore not be deployable on any other. It is however possible to extract the hardware controller to use with other solutions, should one be so inclined.

The Controller class is considered to be the owner of all hardware. For this reason any effort to control the hardware must be done through this class. This constraint might lead to Controller objects being held in several different places, therefore the class was designed with the singleton pattern to achieve consistency.

In order to be able to control home equipment several different devices must be connected to the platform running the software. With the Raspberry Pi this connection is achieved by utilising the General-purpose input/output (GPIO) pins situated on the board. These pins need to be configured at runtime to achieve the desired functionality. The Controller class contains methods for assigning pins and setting them up as needed. There are two ways a pin can be assigned, one can either try to obtain a certain pin or just ask for the first available one.

5.6.2 Virtual representation of controllable devices

Devices that can be controlled by the prototype are represented as various classes in the software. These classes correspond to different types of devices, for instance, media devices or remote controlled outlets. Each class is a subclass of the abstract class AbstractDevice that contains empty methods for all possible interactions. Each type of device class then overrides the appropriate methods and implements the needed functionality.

5.6.3 I/O devices

One device capable of providing input to the prototype is the motion sensor. The Controller class assigns a pin as input for the sensor to be connected to and then sets up an anonymous inner class as a listener to the assigned pin. This inner class is triggered when the input pin it listens to experiences a change in logic level. If the state goes from low to high it means that the sensor detected motion and the current system time is stored as the time for the last movement.

The Controller class contains a method called isAsleep. This method determines if enough time has passed since the last movement and a device should be turned off or the prototype should be put in sleep mode. The decision of when and if to use this functionality lies outside the hardware-controller module.

In order to control the radio transmitter, used to control the radio outlets, a pin is set up as output and assigned to the transmitter by the Controller class. The Controller class contains a method called radio that controls the output of the pin in order to construct the message to be sent.

An effort was made to develop our own software for controlling the radio outlets but this proved to be very difficult. The difficulties were likely caused by the attempt to implement the control in Java. Due to the Java virtual machine it is hard to achieve the timing that is needed to transmit the messages successfully. Instead an open-source project called PiHAT [27] was found that provided the desired functionality for the type of outlets used for the project. The program was installed on the Raspberry Pi and then run from inside the Java code.

5.7 Results of accuracy tests

Tests were performed in order to map the difference in effectiveness of the prototype during the development. Recognition accuracy was recorded for different models and implementations.

The accuracy tracker already built into Sphinx-4 was used to gather the exact percentage regarding accuracy of valid commands accepted by the prototype's grammar. By following the guide on the CMU Sphinx homepage [28] the accuracy testing could be executed during the development process and thereby verify whether the accuracy improved or degraded with a new feature.

5.7.1 Verbose grammar

The test was conducted by recording ten sample sentences each, that the prototype can understand, and then running the test program with the same configuration file and dictionary but with two different acoustic models. As seen in table 1 the accuracy was higher with the adapted acoustic model, described in subsection 5.2.2, than with the standard acoustic model for all but one participant. The reason for the decrease in accuracy for that one participant on the adapted model could be because of the pronunciation in the test data. However, it is hard to draw a good conclusion since the test is rather limited. When calculating the accuracy with the different acoustic models for all the sampled audio recordings at once the data shows an increase in accuracy by twelve percentage points. This increase indicates that the adapted model does in fact improve the recognition results for the participants.

Table 1: Word accuracy data observed during two tests, one on the adapted acoustic model and one on the standard acoustic model. The word accuracy states how many words in a sentence the recogniser interprets correctly.

	Word Accuracy	
Participant	Adapted Acoustic Model	Standard Acoustic Model
Emil	73.68%	50.00%
Jacob	70.37%	79.63%
Marika	98.49%	92.42%
Markus	93.94%	65.15%
Total	86.16%	74.11%

Table 2: Sentence accuracy data observed during two tests, one on the adapted acoustic model and one on the standard acoustic model. The sentence accuracy states how many complete sentences the recogniser interprets correctly.

	Sentence Accuracy	
Participant	Adapted Acoustic Model	Standard Acoustic Model
Emil	30.00%	20.00%
Jacob	30.00%	40.00%
Marika	60.00%	60.00%
Markus	50.00%	10.00%
Total	42.50%	32.50%

Table 2 presents the amount of sentences that were correctly interpreted and where no extra words were incorrectly added. The received results indicate an increase in sentence accuracy while using the adapted model. Although the numbers themselves might seem low, they are not necessarily an accurate image of the prototype's ability to interpret commands. The grammar does allow the instance of multiple polite words at the beginning and end of a sentence, e.g. "please" and "could you". If the model inserts or skips one of those words the accuracy tracker will say that the sentence is wrong, but the program itself would still have been able to understand

the underlying command and execute the corresponding action. Thus, the number of correctly performed actions is likely to be higher for both the adapted acoustic model as well as for the standard acoustic model.

5.7.2 Specialised grammar

A larger test with nearly 30 minutes of recorded speech from six different users, who each read 100 sentences, was also conducted. The test was performed on a smaller, more specialised, grammar. This grammar is the one used with the finished prototype. The results as a whole can be seen in appendix D, where values from both the adapted acoustic model as well as the standard acoustic model are presented.

In table 3 the comparison between the adapted acoustic model and the standard acoustic model in regard to word accuracy shows that even though the acoustic model is only adapted with a small amount of recorded speech data the accuracy is greatly improved. The accuracy is also improved for users, outsiders, whose voices were not recorded for the adaptation of the model. This result indicates that the goal to improve the model for Swedish users most likely was achieved. The data also shows that the users, developers, whose voices were part of the adaptation session received a greater increase in accuracy in comparison to the outsiders. This improved result was to be expected since the model was adapted using their voices.

Table 3: Word accuracy data observed during two tests, one on the adapted acoustic model and one on the standard acoustic model. The developers refer to the users involved in the adaptation process and the outsiders refer to those who were not.

	Word Accuracy	
Participant	Adapted Acoustic Model	Standard Acoustic Model
Developers	91.50%	77.10%
Outsiders	97.50%	90.25%
Total	93.21%	80.86%

Chapter 6

Discussion

This chapter is divided into three sections in order to discuss the results of the project. Section 6.1 covers whether the purpose of the project is satisfied or not. Section 6.2 discusses the impacts the project might have on society. Section 6.3 covers challenges discovered during the development of the prototype.

6.1 Evaluation of prototype based on purpose

The following subsections will thoroughly evaluate the progress made in satisfying the purpose, which is presented in section 1.2.

6.1.1 Affordable

As mentioned in section 4.3 the overall cost of the prototype was approximately 180 USD. This cost includes the Raspberry Pi, motion sensor, radio transmitter and three radio-controlled outlets as well as a microphone of sufficient quality. It should be noted that this cost only accounts for the purchase of the hardware and not the cost of the development. If the prototype were a commercial product the price would be increased if development costs were included.

The affordability property is difficult to objectively evaluate as it differs from person to person. The results from the user study does however provide some basis. The final cost of the prototype seems to be within an acceptable range for most users, as presented in section 3.1.

Even though the prototype cost appears to be within acceptable limits it might seem like a high price to pay for the functionality received. However, the largest part of the cost is the Raspberry Pi which accounts for approximately half the cost. Further additions would be relatively cheap, for example three additional controllable outlets would only incur a cost of approximately 20 USD.

6.1.2 Ease of use

The ease-of-use property can be divided into two parts, the first one being the installation of the prototype and the other being the actual usage. It can be quite difficult to assess whether or not the prototype satisfies this property since the project members possess the level of technical knowledge that a user should not need. Despite this difficulty it is quite clear that one of these criteria is satisfied while the other is not.

Installation of the prototype is where it might fall short of the intended level of ease. Apart from the physical connection of devices to the Raspberry Pi, which is hard to remove, the process of setting up the specific devices for a particular home requires changes to the code. This modification is something that the average user is unlikely to be able to do. Ideally the

modification would be handled by a graphical, or a speech, interface.

We deemed that the set up could be done for the user, either by the provider of the system or by anyone else with the required knowledge. Therefore the ease of setting up the prototype was given a lower priority than making the prototype easy to use while up and running. When the prototype is up and running it is very easy to use, the user only needs to speak a command and the appropriate action will be taken, assuming that the command is correctly recognised. The prototype is capable of suspending itself when it is not being used and automatically resume when motion is detected without any user interaction. The effect of this automation is that after the initial installation no further configuration is needed from the user.

6.1.3 Functional

According to the property of functionality it should not be more difficult to control devices using the prototype than with more conventional methods. As the prototype sometimes fails to interpret a given command, forcing the user to repeat it, this property might not seem fully satisfied. However, since the prototype enables users to control devices that might otherwise be difficult to control, the need to sometimes repeat a command is somewhat mitigated

Another aspect that improves the functionality is the ability to control multiple devices from the prototype. For instance a light switch usually controls only one device. The ability to control devices over a network further solidifies the functionality property as the user is able to control his home equipment from anywhere in the world. This enables the user to shut down devices that he might have forgotten to turn off.

6.1.4 Relevant functionality

The results of the conducted user study, presented in section 3.1, show that potential users are most interested in controlling lights in their home. Therefore this feature was the focus during development. By using wall sockets that can be turned on and off by the use of radio signals controlling lights became simple. Any other device that uses a wall socket as power source and only needs to be turned on to perform its main task, can easily be controlled the same way, e.g. electric kettles.

With this functionality in mind the grammar used by the prototype was constructed for supporting control of lights with several different command formulations. By following this pattern and implementing the functionality that the respondents to the user study found most important, the prototype's functionality was kept relevant throughout the project.

6.1.5 Limit energy consumption

With the useage of motion sensors the prototype can perceive if a room is empty. This feature allows the prototype to switch off lights in an empty room in order to save energy. The prototype uses a Raspberry Pi as its base due to its low energy consumption, which was important since it is supposed to be powered on at all times.

As can be seen in the results of the user study, as presented in section 3.1, reducing the energy consumption was one of the most important functionalities for the respondents. This fact motivated the work on reducing energy consumption by implementing sleep mode to the prototype, which it can resume from through the use of the motion sensor. In sleep mode the prototype

does not try to interpret sound as commands, which saves computational power and thereby energy.

6.2 Societal impacts of the prototype

The day-to-day life for people with disabilities can be improved using the prototype since it can enable them to do things they otherwise would not be able to do. We hope that this improvement may decrease the gap between people with disabilities and other social groups making people with disabilities feel more as a part of society.

When it comes to environmental benefits the prototype can assist with some optimisation in one's home. For example the energy consumption in a user's home can be limited by using the sleep mode implemented in the prototype. This limiting of energy consumption is possible since the prototype can disable devices when the house, or room, is empty. Lastly, some societal-economical benefits can also be accomplished using the prototype since it enables disabled people to live a more independent life. The use of the prototype can decrease the need of assistance for users with disabilities, which in turn would lower the expenditures associated with this assistance.

6.3 Challenges

The most complex part of the system is the speech-recognition module. It was also this module that offered the most challenges and some of these will be discussed in subsection 6.3.1 and 6.3.2. Another big challenge was found in obtaining sufficient performance of the speech recognition on the Raspberry Pi which is discussed in subsection 6.3.3

6.3.1 Speech-recognition module

During the development of the prototype some larger problems were found. When it came to speech recognition most of these problems concerned accuracy and more specifically incorrect recognition. Due to the fact that computers are not very good at distinguishing speech it is very hard to receive a correct recognition of words that have similar sound. Words such as "activate" and "deactivate" were often confused by the recogniser. The speech input of the words needs to be pronounced very clearly in order for the recogniser to produce the expected result. One solution to this problem is to define a grammar without words that are often confused though this solution enforces unwanted constraints. The models can of course be trained or adapted to possibly produce improved recognition but this method did not provide the desired results for the project. A good solution to this problem was not found but one working option is to simply define valid sentences that all have some distinct feature that sets them apart. For example the command for activating a lamp could be "activate lamp" and the command for deactivating a lamp could be "disable lamp".

Another challenge was the rejection of out-of-grammar sentences. Sphinx-4 returns the sentence that is most likely to correspond to the speech input. With a prototype that is supposed to only interpret specific commands, rejection of invalid input is very important. By the end of the development it was found that Sphinx-4 has a setting to help with out-of-grammar rejection. This setting adds a path to the search graph with a static probability, which is not affected by the speech input and corresponds to an invalid command. With this modified search graph the recogniser returns "unknown" as a result if the probability of the sentences during search is below the probability of the invalid path. However, one challenge with this setting is that you have to be able to specify a suitable static probability, which is not trivial. There is no value that works perfectly on every setup, instead it has to be adjusted to the system it is run on.

The last distinct challenge discovered during the development was that the prototype sometimes

became unresponsive when no speech input was detected. We modified the Sphinx-4 decoder with the ability to specify how long the decoder may try to process the input and interrupt the process if it exceeds this set timeout. This feature is a very good tool in order to prevent the program from becoming unresponsive. The downside with this approach is that the program might be interrupted in the middle of a valid recognition attempt. The effect of setting the timeout value too low may cause the system to interrupt longer, more demanding, valid attempts of recognition. Conversely, a high value may cause the system to block new commands for long periods as it is occupied with processing previous input.

6.3.2 Parsing the recognised sentence

The digital representation of the speech input returned by the recogniser is the complete sentence spoken. No modification of the sentence is made by the recogniser. Since a sentence usually contains filler words that serve no other purpose than to bind the sentence together, some parsing is required in order to extract the relevant information. The ideal situation is that the output only contains words that are relevant in order to decide which action should be taken.

To parse the output of the recognised sentence, two options were discussed: returning a string for every word and let the core module parse them or generating a command object with a variable for each type of word, e.g. a device or an action. Performing the parsing in the core module would have increased the dependency between the modules, this fact made it an undesired option. Instead scripted tags were used. The scripts enabled the values of the Command object to be entered automatically depending on the recognised sentence. If scripted tags would not have been used the prototype would probably have suffered from degraded performance due to added complexity of the parsing.

6.3.3 Java on the hardware platform

The choice to use Java was based primarily on the project group's familiarity with the language and the portability it offers. It was deemed that the overhead introduced by Java would not be a major hindrance and would be offset by the previously mentioned benefits. This belief was further enforced after testing Sphinx-4 with a very basic grammar.

However, after implementing more of the prototype's features a considerable decrease in performance was noted on the Raspberry Pi (RPi). As soon as commands were spoken the program would hit the computational cap of the RPi which noticeably slowed down the response time and overall flow. Considering this limitation it would have could have produced better performance to implement the program in something other than Java, such as C or C++, and use speech-recognition software implemented in the same language.

This problem did however highlight how advantageous the portability of Java and the modular structure of the prototype can be. Due to the performance problems a workaround was tested. By using the implemented server functionality the prototype was modified to not run the speech-recognition logic on the RPi. Instead this was handled on another, more capable, computer that interprets the speech and sends a command object to the RPi. This change took less than an hour to implement and resulted in the system performing within acceptable parameters.

Chapter 7

Conclusions and future directions

This chapter starts with a brief summary of the most important results and challenges of the project with conclusions on how well the prototype is deemed to operate considering the most important requirements.

With a project of this magnitude the prototype could always be improved. The development process of the prototype resulted in discovering new features that would have been desirable to implement. Some of these features and thoughts are discussed in section 7.2.

7.1 Conclusions

The project resulted in a prototype that is capable of controlling remote-controlled outlets and through them many different kinds of home equipment. The user can perform this control either by using voice commands or a graphical interface. The prototype is always ready to receive commands from the user.

The prototype developed during the project performed with satisfying results. The accuracy reached with users that has distinct pronunciation resulted in a limited need to repeat commands. The adaptation of the acoustic model was an important factor to the high accuracy since it was adapted to the Swedish accent. To use the adapted model with a different accent would most likely reduce the accuracy.

The chosen hardware and framework did not have maximum compatibility which meant that an optimal result could not be reached. To increase performance the prototype would need more computational power, which would incur a higher cost. This increased cost would counter the affordability property defined in the project purpose. It might have been better to use a speech-recognition framework based on a lower-level language, for example C, and a more light-weight framework with less demanding search algorithms. These changes would probably optimise the execution of the software on the Raspberry Pi, which could lead to a more satisfying result.

The properties defined by the project purpose were fulfilled. The prototype has a relatively low cost and is easy to use. The implemented functionality corresponds well to the answers from the user study making the prototype relevant. By being able to automatically turn off equipment the prototype can limit the energy consumption of the user's home.

7.2 Future directions

In a project of this type it is common to have many more ideas than what is possible to implement and this project is not an exception. The next, most obvious, step would be to add more devices to control and new methods for controlling them beyond radio. By adding an IR-transmitter the prototype would be able to emulate remote controls for operating TV-sets, sound systems etc.

Instantiation of devices are currently hardcoded, though a future implementation could support dynamic construction of devices from a settings file. As reflection supports runtime execution of

constructors, an implementation supporting this solution would make it possible to instantiate devices that are desired at the moment. This implementation would make the prototype more flexible and extendable.

Another positive aspect from the dynamic-instantiation solution would be the possibility to use the same implementation between different setups and in different homes. The only thing that would need to change is the actual settings file, which would contain the desired devices.

Expanding the prototype with security sensitive functionality such as locking and unlocking a door, would introduce the need for some type of user authentication. Only a specific user should be able to issue sensitive commands. One way to implement this security would be to use speaker identification which is the process of determining the identity of a speaker. This process would give the system the capability of blocking access to sensitive commands from unauthorised users.

Speech recognition today is very sensitive to background noise or unclear speech input. This sensitivity means that a speech-recognition application performs worse in noisy environments and that recognition with multiple sound sources active at the same time is rather difficult. Future goals for a home-automation–speech-recognition system could be to understand commands or input even in noisy environments, such as understanding commands when there are multiple people talking in the background. This task however would most likely demand a lot of research and may also need completely new implementations of speech recognition.

Bibliography

[1] *reddit: the front page of the internet*, [online]. Available: `http://www.reddit.com/` [Accessed: 2014-06-05]

[2] J. Benesty, M. M. Sondhi, Y. Huang, *"Introduction to Speech Processing" in Springer Handbook of Speech Processing*. Würzburg: Springer Science+Business Media, 2008. [E-book] Available: Springer Link.

[3] S. Furui, *"History and Development of Speech Recognition", in Speech Technology: Theory and Applications*. New York: Springer Science+Business Media, 2010. [E-book] Available: Springer Link.

[4] M. Gales, S. Young. (2008, Feb). "The Application of Hidden Markov Models in Speech Recognition", *Foundations and Trends in Signal Processing*. [online]. vol. 1, nr. 3, pp. 195–304. Available: doi:10.1561/2000000004 [Accessed: 2014-03-14].

[5] A. Buchsbaum, R. Giancarlo. (1997, Jan) "Algorithmic aspects in speech recognition: An introduction", *Journal of Experimental Algorithmics*. [Online]. vol. 2, article nr. 1. Available: doi:10.1145/264216.264219 [Accessed: 2014-03-22].

[6] *What is an acoustic model?*, [online]. Available: `http://www.voxforge.org/home/docs/faq/faq/what-is-an-acoustic-model`

[7] D Spicer. (2000, August 12). *If You Can't Stand the Coding, Stay Out of the Kitchen: Three Chapters in the History of Home Automation*. [Online]. Available: `http://www.drdobbs.com/architecture-and-design/if-you-cant-stand-the-coding-stay-out-of/184404040?pgno=1`. [Accessed: 2014-05-19].

[8] *Home Automation and Smart Home Control*, [online]. Available: `http://www.control4.com/` [Accessed: 2014-05-19].

[9] S. Goodwin, *Smart Home Automation with Linux*. New York, USA: Apress, 2010.

[10] *VoicePod - Frequently Asked Questions*, [online]. Available: `http://www.voicepod.com/faq/` [Accessed: 2014-05-19].

[11] *HTK speech recognition toolkit*, [online]. Available: `http://htk.eng.cam.ac.uk/` [Accessed: 2014-02-14].

[12] *Sphinx-4 Application Programmer's Guide - CMUSphinx Wiki*, [online]. Available: `http://cmusphinx.sourceforge.net/sphinx4/`. [Accessed: 2014-02-12].

[13] *Open-Source Large Vocabulary CSR Engine Julius*, [online]. Available: `http://julius.sourceforge.jp/en_index.php` [Accessed: 2014-02-14].

[14] *Kaldi*, [online]. Available: `http://kaldi.sourceforge.net/` [Accessed: 2014-02-14].

[15] *Training Acoustic Model For CMUSphinx*, [online]. Available: `http://cmusphinx.sourceforge.net/wiki/tutorialam`. [Accessed: 2014-05-16].

[16] D. Povey et al., "The Kaldi Speech Recognition Toolkit", in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, Hilton Waikoloa Village, HaWaii, 2011.

[17] P. Lamere et al., "Design of the CMU Sphinx-4 decoder", in *Proceedings of the 8th European Conference on Speech Communication and Technology (Eurospeech 2003)*, Geneve, Switzerland, 2003, pp. 1181–1184.

[18] *GIT*, [online]. Available: http://git-scm.com/. [Accessed: 2014-04-29].

[19] *Arch Linux ARM*, [online]. Available: http://archlinuxarm.org/. [Accessed: 2014-05-19].

[20] *Raspbian*, [online]. Available: http://www.raspbian.org/. [Accessed: 2014-05-19].

[21] *Home Automation – RF Protocols*, [online]. Available: http://tech.jolowe.se/home-automation-rf-protocols/. [Accessed: 2014-05-19].

[22] *Adapting the default acoustic model*, [online]. Available: http://cmusphinx.sourceforge.net/wiki/tutorialadapt [Accessed: 2014-05-19]

[23] *CMU Sphinx Wiki*, [online]. Available: http://cmusphinx.sourceforge.net/. [Accessed: 2014-05-16].

[24] *Sphinx Knowledge Base Tool – VERSION 3*, [online]. Available: http://www.speech.cs.cmu.edu/tools/lmtool-new.html. [Accessed: 2014-05-19].

[25] *JSpeech Grammar Format*, [online]. Available: http://www.w3.org/TR/2000/NOTE-jsgf-20000605/ [Accessed: 2014-05-09].

[26] *Trail: The Reflection API*, [online]. Available: http://docs.oracle.com/javase/tutorial/reflect/. [Accessed: 2014-05-08].

[27] *433.92MHz Nexa/Status remote control wall sockets using raspberry pi's GPIO pin 4*, [online]. Available: https://github.com/s7mx1/pihat. [Accessed: 2014-05-19].

[28] *Instrumentation for Sphinx-4*, [online]. Available: http://cmusphinx.sourceforge.net/doc/sphinx4/edu/cmu/sphinx/instrumentation/doc-files/Instrumentation.html [Accessed: 2014-05-19]

Appendix A

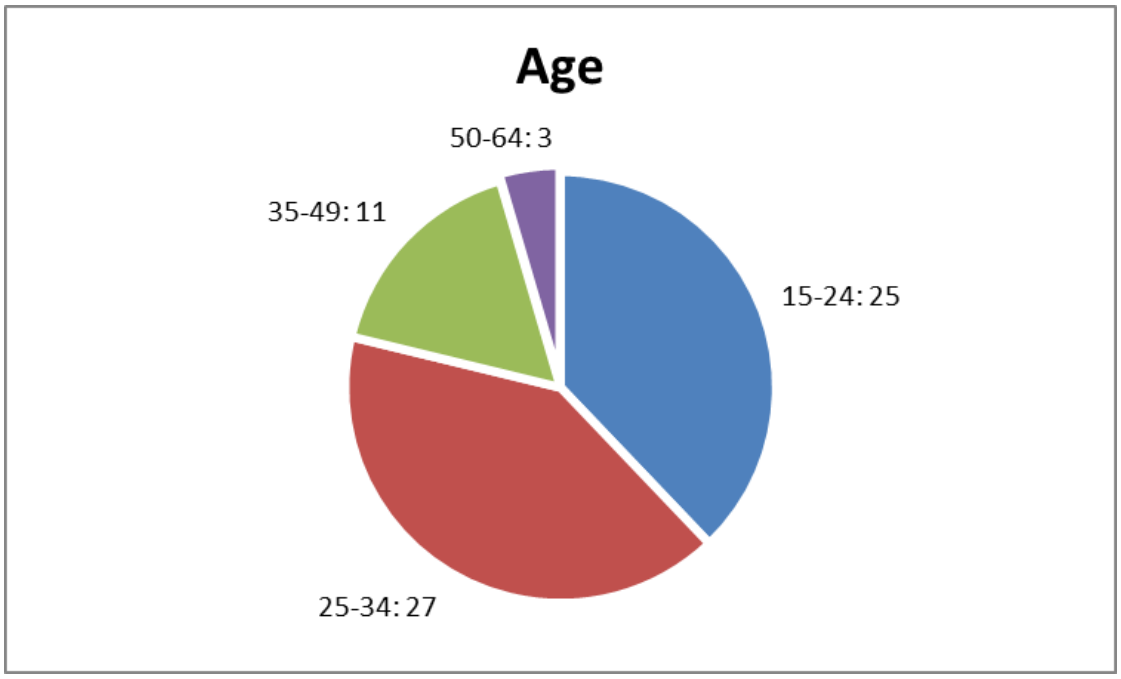
Results from User Study

This appendix details the performed user study. Each section contains a question as it appeared to the respondents together with a chart showing the gathered results.

A.1 Question 1

How old are you?

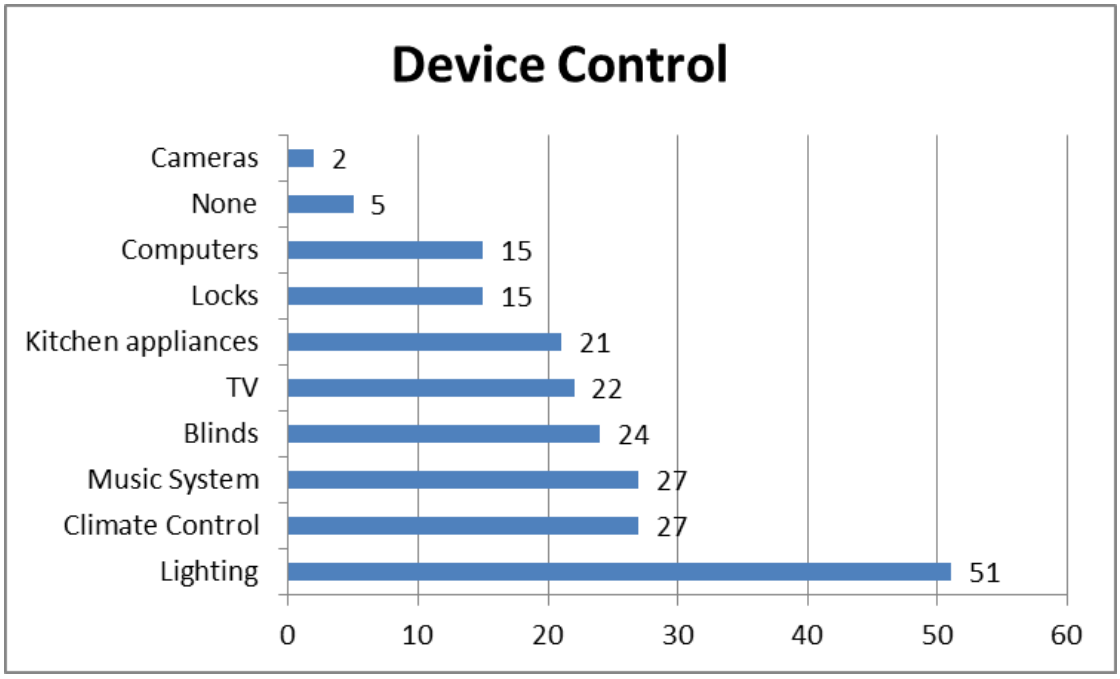
- 0-14
- 15-24
- 25-34
- 5-49
- 50-64
- 65+



A.2 Question 2

If you could control equipment in your household solely by using your voice, which would you be most interested in controlling?
Choose a maximum of four options

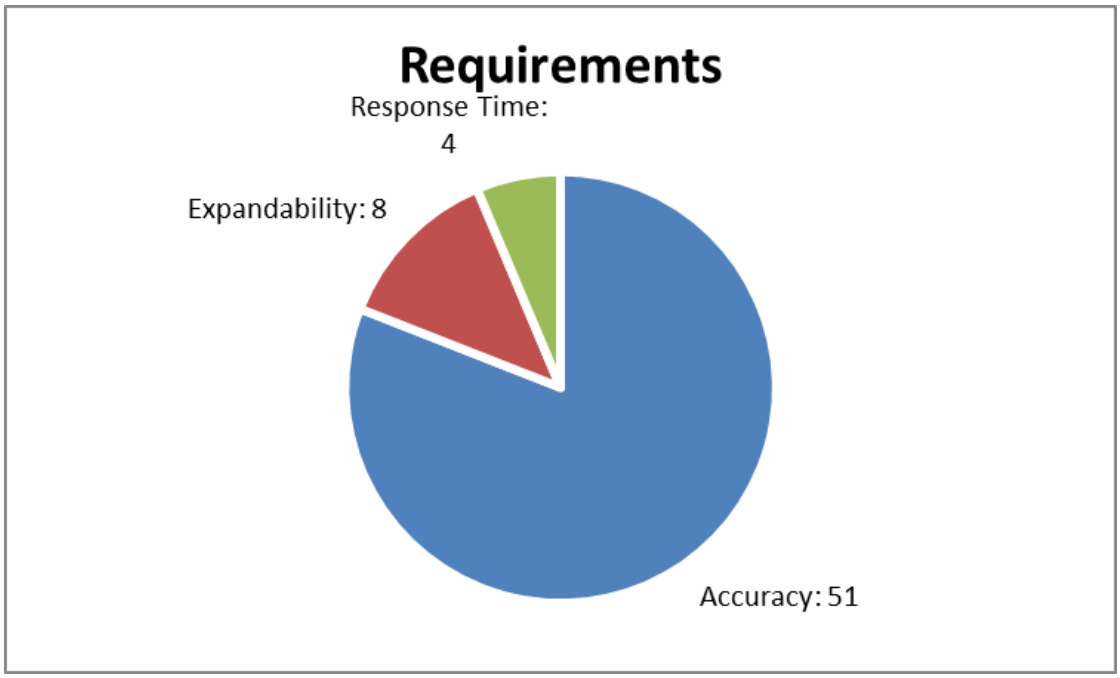
- ☐ Lighting
- ☐ TV
- ☐ Climate Control
- ☐ Blinds
- ☐ Kitchen appliances (e.g. coffee maker, dish washer)
- ☐ Locks
- ☐ Music System
- ☐ Computers (e.g. turn on or off)
- ☐ Cameras
- ☐ None
- ☐ Others



A.3 Question 3

What would be the most important aspect for you regarding functionality?

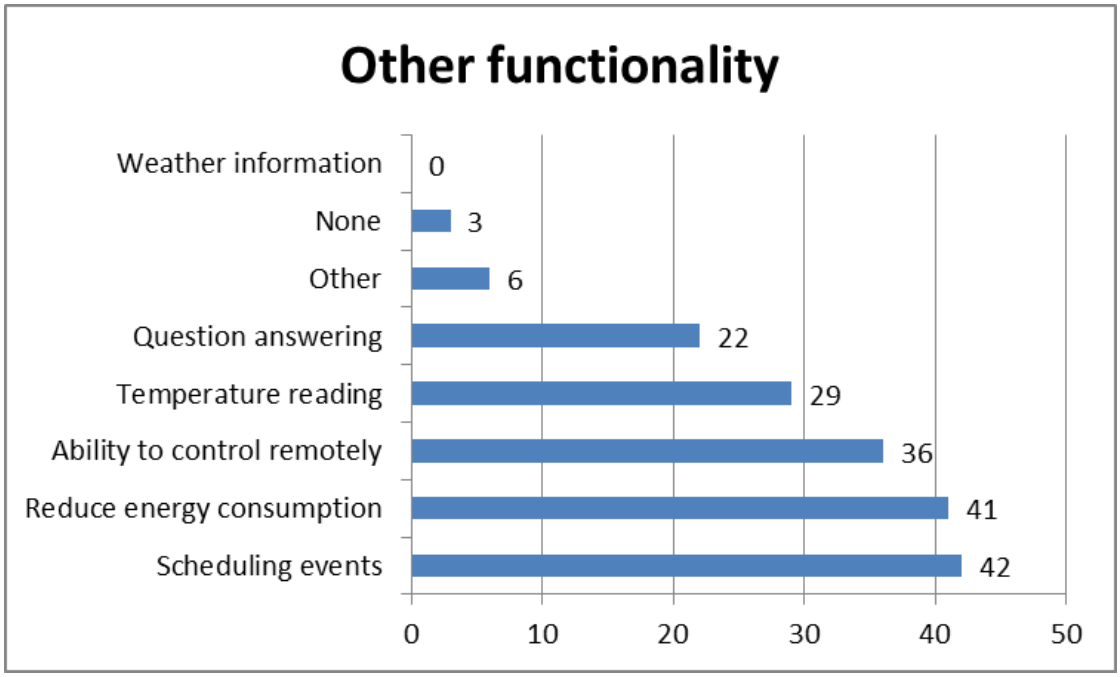
◦ Accuracy (how often the system interprets your commands correctly)
◦ Response Time (measures how long it takes from speaking a command to the corresponding action being performed)
◦ Expandability (a measure of how easy one can add to the system, such as adding more devices to control)
◦ Other



A.4 Question 4

Which other functions would you want to have in a smart home system?

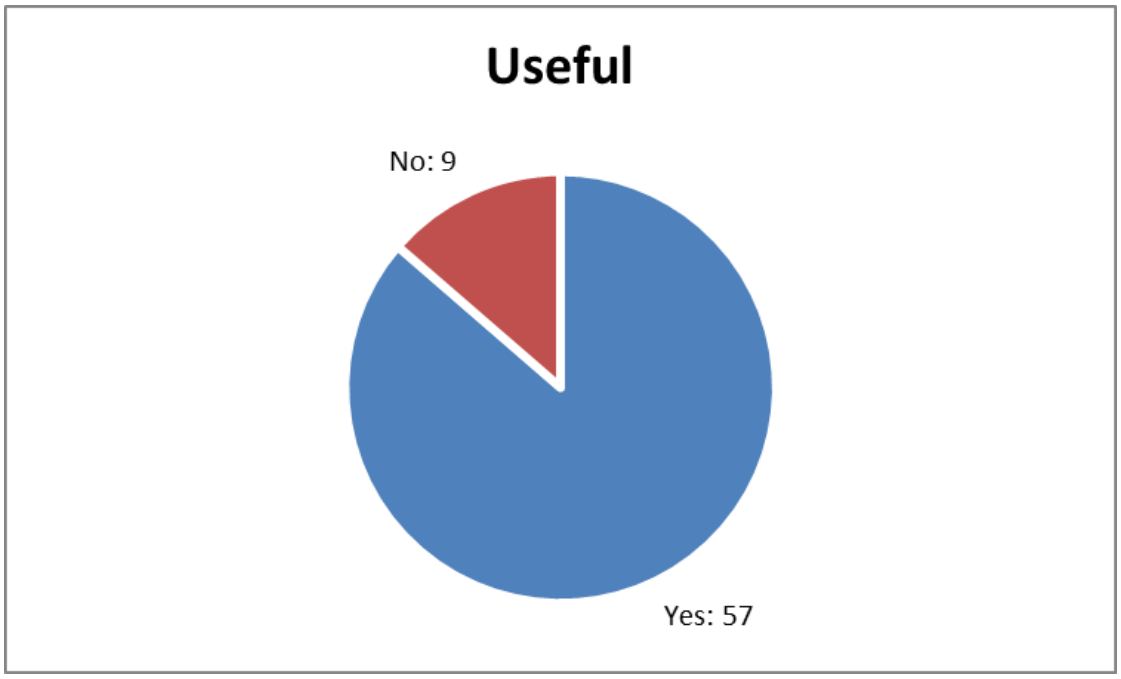
- ☐ Temperature reading
- ☐ Scheduling events (e.g. open the blinds at a certain time)
- ☐ Question answering (able to query Google or other search engine)
- ☐ Ability to control remotely (e.g through phone app)
- ☐ Reduce energy consumption (e.g. by switching off lights when no one is home)
- ☐ Weather information
- ☐ None
- ☐ Other



A.5 Question 5

Do you think that a voice controlled smart home could have a positive impact on your day-to-day life?

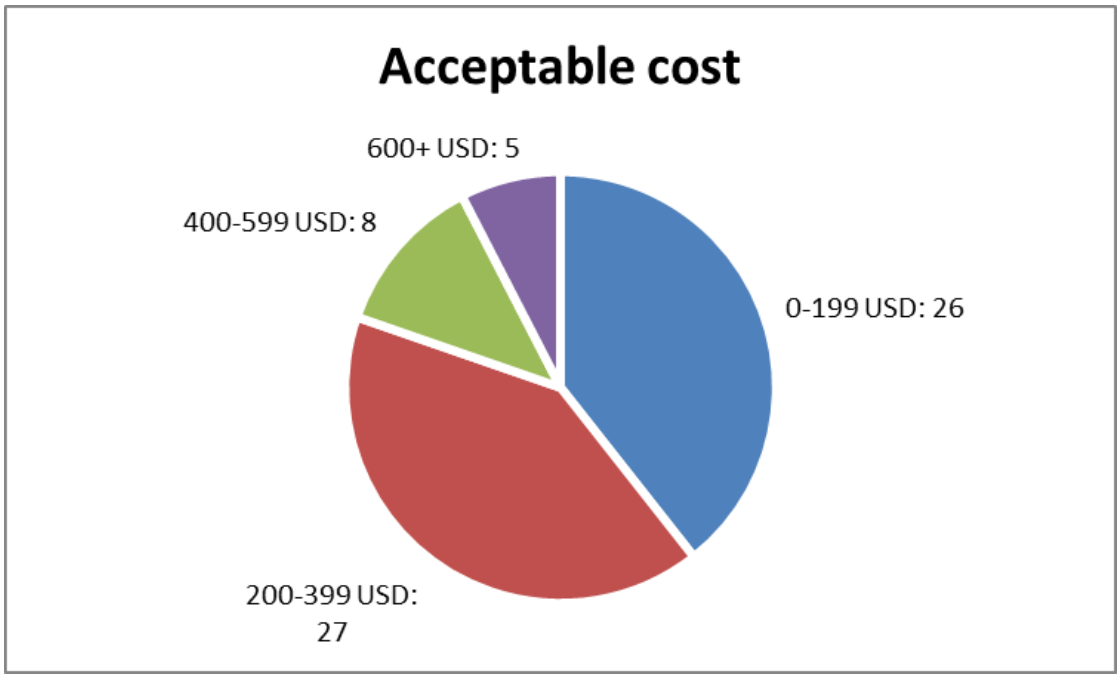
- Yes
- No



A.6 Question 6

How much would you be willing to spend on a voice controlled smart home system?
Excluding the price for the devices that are to be controlled

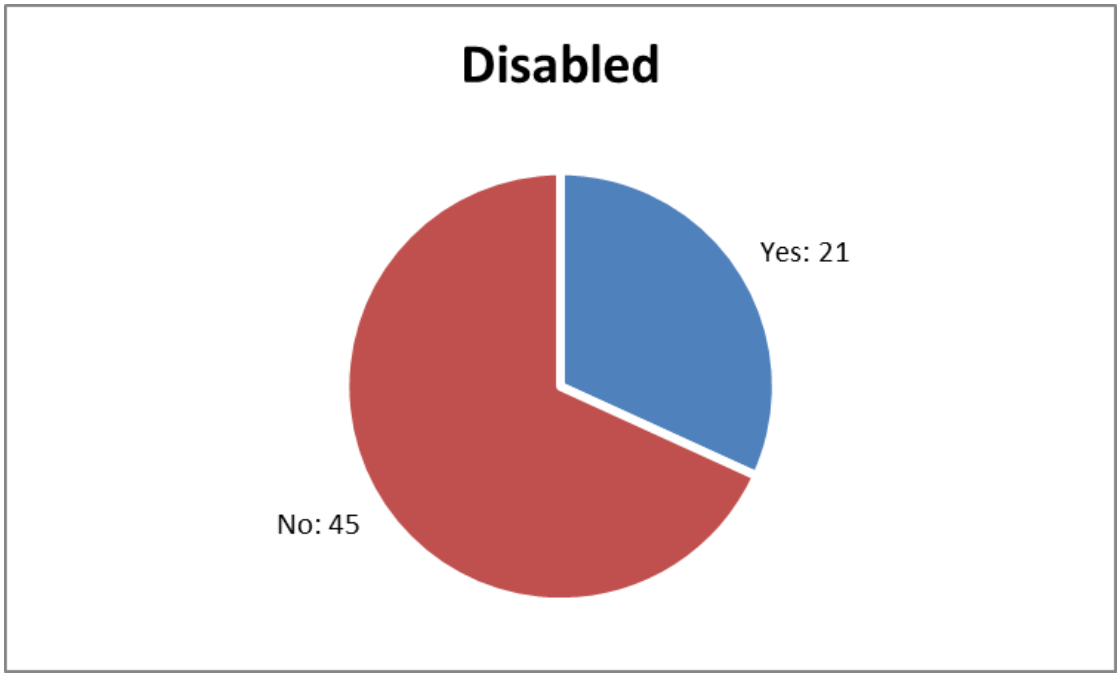
- 0-199 USD
- 200-399 USD
- 400-599 USD
- 600+ USD



A.7 Question 7

Do you have any physical disability?

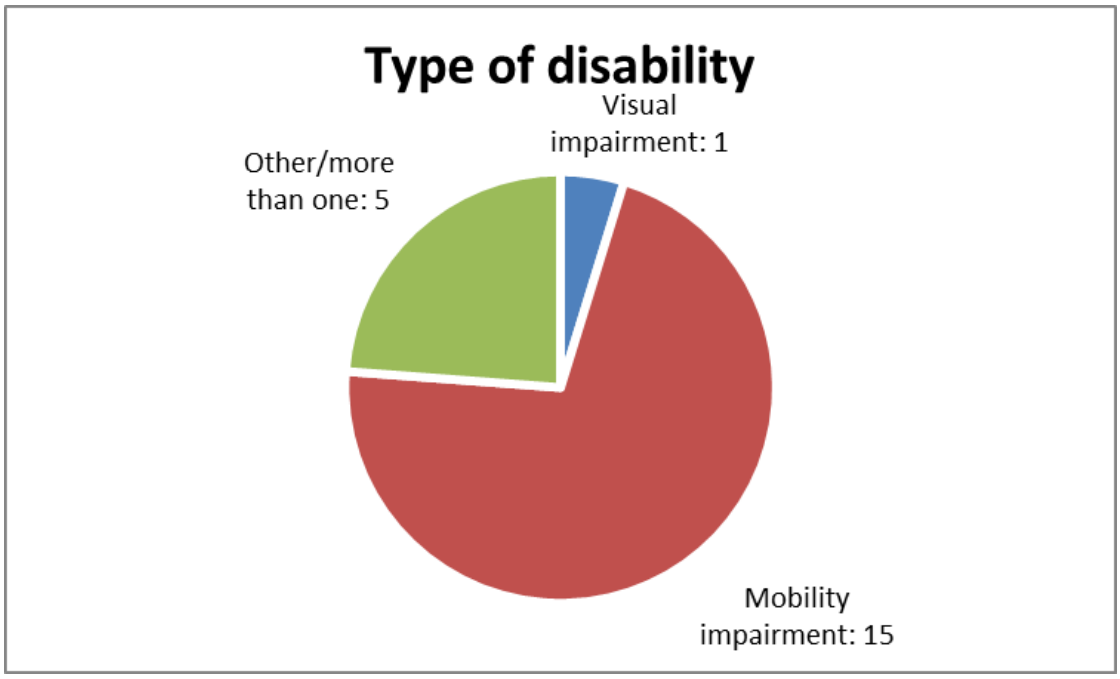
- Yes
- No



A.8 Question 8

Which type of disability do you have?

- Visual Impairment (e.g. Blindness, Cataracts)
- Hearing Impairment (e.g. Hearing Loss, Tinnitus)
- Mobility Impairment (e.g. Arthritis, Limb Loss, Paralysis)
- Other/more than one
- I prefer not to answer



Appendix B

Technical Specifications

This appendix details the technical specifications of several devices used in the project.

B.1 Single-board computer - Raspberry Pi Model B Rev 2

Processor	
Speed (MHz)	700
Cores	Single Core
Type	ARM1176JZF-S
System on Chip	Broadcom BCM2835
Memory	
RAM (MB)	512
Ports	
HDMI	1x 1.3a
Composite	1x RCA
USB	2x USB 2.0
Ethernet (Mbit/s)	1x 10/100
Memory card reader	MMC, SD, SDHC
Dimensions	
Length (mm)	85.6
Width (mm)	53.98

B.2 Remote controlled outlets - Nexa PB-3-VIT

Characteristics	
Rated Voltage (V)	230
Max load (W)	2300
Method of control	
Radio (MHz)	433.92
Range (m)	20-25

B.3 Radio transmitter module - Aurel TX -4MDIL

Characteristics	Minimal	Typical	Maximal
Supply Voltage (Vdc)	3	-	5
Supply Current (mA)	3	-	6
Carrier frequency (MHz)	-	433.92	-
Effective radiated power (dBm)	-1	-	2
Square wave modulation (KHz)	-	-	4
Operating temperature range (°C)	-20	-	+80
Dimensions			
Length (mm)	-	17.78	-
Width (mm)	-	10.16	-

B.4 Motion sensor - Adafruit PIR (motion) sensor

Characteristics	Minimal	Typical	Maximal
Supply Voltage (V)	5	-	16
Output voltage (V)	-	3.3	-
Sensor capabilities			
Range (m)	-	~7	-
Angle (°)	-	120	-
Dimensions			
Length (mm)	-	32.34	-
Width (mm)	-	24.03	-
Height, with lens (mm)	-	24.66	-

Appendix C

Grammar with Scripted Tags

```
#JSGF V1.0;

/**
 * JSGF Grammar for Voice controlled home, Jarvis
 *
 * @author Marika Hansson
 * @author Markus Otterberg
 */

 /*{tag}, [optional]*/

grammar orignJarvis;

/* Pleasantries */
<polite> = Please | Could You | Greetings | <name>;
<greeting> = <NULL> {appObj.addDevice("info");} ([Good] (Morning
    {appObj.addAction("morning");} | Night {appObj.addAction("night");} | Bye
    {appObj.addAction("bye");}) | (Hello | Hi) {appObj.addAction("hello");});
<name> = Computer;

/* Numbers */
<number> = one {appObj.addParam("1");} | two {appObj.addParam("2");} | three
    {appObj.addParam("3");} | four {appObj.addParam("4");} | five
    {appObj.addParam("5");} | six {appObj.addParam("6");} | seven
    {appObj.addParam("7");} | eight {appObj.addParam("8");} | nine
    {appObj.addParam("9");} | ten {appObj.addParam("10");};

/* Predefined codes */
<channelName> = Euro Sport {appObj.addParam("25");} | CNN {appObj.addParam("42");};

/* Devices */
<device> = <simpleDevice> | <electronicDevice>;
<simpleDevice> = (Lamp | Lights | lamps) {appObj.addDevice("lamp");} | Coffee Maker
    {appObj.addDevice("coffeeMaker");}| Fan {appObj.addDevice("fan");};
<electronicDevice> = (TV | Television) {appObj.addDevice("tv");} | (Cd Player | Music
    [Player]) {appObj.addDevice("musicPlayer");};

/* Locations */
<room> = Kitchen {appObj.addPosition("kitchen");} | Bedroom
    {appObj.addPosition("bedroom");} | Living Room {appObj.addPosition("living");} |
    Bathroom {appObj.addPosition("bathroom");} | Hallway
    {appObj.addPosition("hallway");};

/* Actions */
<onSwitch> = ([Turn] On | Enable | Activate) {appObj.addAction("enable");};
<offSwitch> = ([Turn] Off | Disable | Deactivate) {appObj.addAction("disable");};

/* Sound related */
<mute> = Mute {appObj.addAction("mute");};
```

```
<activate> = (Reactivate) {appObj.addAction("unmute");};
<volume> = [Turn] [[The] Volume] (<upVol> | <downVol>) {appObj.addParam("5");
    appObj.addDevice("tv");} [[The] Volume] [By <number>] [On [The] <electronicDevice>];
<upVol> = (Increase | Up) {appObj.addAction("upVolume");};
<downVol> = (Lower | Down) {appObj.addAction("downVolume");};

/* System information related */
<date> = [What Is (The [Current] | Todays)] <dateTime> {appObj.addDevice("info");};
<dateTime> = Date {appObj.addAction("date");} | Time {appObj.addAction("time");} | Week
    {appObj.addAction("week");} | Study Week {appObj.addAction("studyWeek");} |
    Temperature {appObj.addAction("temp");};

/* Rules for commands */
<lights> = <light> | <light2> | <light3>;
<light> = (<onSwitch> | <offSwitch>) [The] [<room>] <device>;
<light2> = [<room>] <device> (<onSwitch> | <offSwitch>);
<light3> = (<onSwitch> | <offSwitch>) [The] <device> In [The] <room>;

<channel> = (<channelTo> | <channelStep>) {appObj.addAction("channel");
    appObj.addDevice("tv");} [On <electronicDevice>];
<channelTo> = [Switch] To (Channel <number> | <channelName>);
<channelStep> = Channel (Up {appObj.addParam("up");} | Down {appObj.addParam("down");});

<simpleCommand> = <NULL> {appObj.addDevice("microphone");} (<mute> [<electronicDevice>]
    | <activate> [<electronicDevice>] | <greeting> | <date>);

/*#########*/

/* Actual spoken commands */
public <spokenCommand> = [<polite>*] (<lights> | <simpleCommand> | <channel> |
    <volume>) [<polite>*] <NULL> {appObj.generateCommand();};

public <filler> = Thank You | Thanks;
```

Appendix D

Accuracy Test Data

Data from the first test were conducted with ten sentences for each user and with only users that were also part of the adaptation of the acoustic model (developers). The results are presented in table D.1 and table D.2.

Table D.1: Word accuracy data observed during two tests, one on the adapted acoustic model and one on the standard acoustic model. The word accuracy states how many words in a sentence the recogniser interprets correctly.

Adapted Acoustic Model		
Participant	Word Accuracy	Sentence Accuracy
Emil	73.68%	30.00%
Jacob	70.37%	30.00%
Marika	98.49%	60.00%
Markus	93.94%	50.00%
Total	86.16%	42.50%

Table D.2: Sentence accuracy data observed during two tests, one on the adapted acoustic model and one on the standard acoustic model. The sentence accuracy states how many complete sentences the recogniser interprets correctly.

Standard Acoustic Model		
Participant	Word Accuracy	Sentence Accuracy
Emil	50.00%	20.00%
Jacob	79.63%	40.00%
Marika	92.42%	60.00%
Markus	65.15%	10.00%
Total	74.11%	32.50%

For the second test a larger amount of data were collected, 100 sentences per user, and a broader set of users were participating. The results are split into categories depending on which acoustic model was used and who participated. The Word Error Rate (usually written as WER) is the percentage of words that were wrong in some way. This parameter measures partly the same as the inverse of the word accuracy but with the added information on words falsely inserted. The results are presented in table D.3 and table D.4.

Table D.3: Accuracy data received from the accuracy tracker during the larger test. Data from the adapted acoustic model is present. The developers refer to the users involved in the adaptation process and the outsiders refer to those who were not.

Adapted Acoustic Model			
Participant	Word Accuracy	Sentence Accuracy	Word Error Rate
Developers	91.50%	86.25%	8.90%
Outsiders	97.50%	94.38%	2.50%
Total	93.21%	88.57%	7.07%

Table D.4: Accuracy data received from the accuracy tracker during the larger test. Data from the standard acoustic model is present. The developers refer to the users involved in the adaptation process and the outsiders refer to those who were not.

Standard Acoustic Model			
Participant	Word Accuracy	Sentence Accuracy	Word Error Rate
Developers	77.10%	71.00%	25.70%
Outsiders	90.25%	84.38%	11.25%
Total	80.86%	74.82%	21.57%