Bachelor Thesis

A Voice User Interface for Human-Robot Interaction on a Service Robot

Mihael Simonič

Supervisors:  Prof. Dr. rer. nat. Andreas Zell
              Wilhelm Schickard Institute for Informatics

              Adrian Zwiener
              Wilhelm Schickard Institute for Informatics

Started:     April 1, 2015

Finished:    August 28, 2015
Declaration of Originality

I hereby confirm that I wrote the present work independently and that I did not use any sources or aids other than those quoted and that if any passages have been copied or in any other way used, all references have been acknowledged and fully cited.

Tübingen, August 28, 2015

Mihael Simonič
Abstract. Human-robot interaction is an important area of robotics. Traditional human-machine and human-computer interfaces do not suffice for all use cases of mobile robots, and in particular humanoid robots. Multimodal interfaces address these needs by involving further human senses. The present work presents an implementation of such system for a humanoid indoor service robot MetraLabs Scitos G5 incorporating a natural language based interface. Using technologies like speech recognition and speech synthesis, the robot is able to accept simple voice commands and respond to them. In order to demonstrate flexibility of the solution, two additional input methods have been developed: desktop GUI for entering natural language commands and a voice-enabled Android remote controller.

Acknowledgements

This thesis is result of my bachelor studies of cognitive sciences at the Faculty of Science at the University of Tübingen, financially supported by The Slovene Human Resources and Scholarship Fund through the “Štipenija za študij slovenskih državljanov v tujini” grant 11010-556/2012.


Hvala družini, da mi je venomr stala ob strani, me motivirala in skozi zanimive dialoge odpirala pot inspiraciji za nove ideje.
# Contents

1. **Introduction** ........................................... 1

2. **Theoretical background** ................................ 2
   2.1. Human-Robot Interaction ............................. 2
       2.1.1. Relation between humans and robots .......... 2
       2.1.2. Existing human-robot interfaces ............ 3
       2.1.3. Multimodal user interfaces and voice control 3
   2.2. Human speech ....................................... 5
       2.2.1. What is speech? .................................. 5
       2.2.2. Modelling speech ................................. 6
   2.3. Speech recognition .................................... 7
       2.3.1. Brief history of speech recognition .......... 7
       2.3.2. Types of speech recognition systems ........ 8
       2.3.3. Functional principle of speech recognizers ... 9
       2.3.4. Accuracy and efficiency measures ............ 10
   2.4. Speech synthesis ..................................... 10
       2.4.1. Formant synthesis ................................ 11
       2.4.2. Overview of open-source TTS systems ....... 12

3. **Platform** ............................................... 13
   3.1. The Scitos G5 Robot .................................. 13
       3.1.1. Odometry ........................................ 15
       3.1.2. Laser scanner measurements .................. 15
   3.2. Robot Operating System .............................. 16
   3.3. Navigation stack ..................................... 17
   3.4. CMUSphinx ........................................... 17
       3.4.1. Architecture of Sphinx-4 ....................... 18

4. **Development of the voice Human-Robot Interface** ..... 22
   4.1. Grammars ............................................. 22
       4.1.1. JSpeech Grammar Format ........................ 23
       4.1.2. Using grammars to constraint speech recognition 24
       4.1.3. Using grammars to simplify syntactic analysis 24
       4.1.4. Detecting Numbers ................................ 25
   4.2. Map of known rooms .................................. 26
   4.3. Integration with ROS ................................ 28
       4.3.1. Voice Input ...................................... 28
       4.3.2. Voice Interpreter ................................ 29
       4.3.3. Voice Output ..................................... 29
   4.4. Selected functions .................................. 30
       4.4.1. Command: *Drive forward/backward* .......... 30
1. Introduction

“The robot was told to pull back the control bar firmly. Firmly. The word was repeated, strengthened, emphasized. So the robot did what it was told. It pulled it back firmly. There was only one trouble. He was easily ten times stronger than the ordinary human being for whom the control bar was designed.”

“Are you implying –” “I’m saying the bar bent. It bent back just enough to misplace the trigger [...] This isn’t the failure of just one robot [...] It’s symbolic of the failure of the robot idea.”

“Come on, [...] the robot was equipped with adequate understanding as well as with brute force. Had the men who gave it its orders used quantitative terms rather than the foolish adverb ‘firmly,’ this would not have happened. Had they said, ‘apply a pull of fifty-five pounds,’ all would have been well.”

Richard (1954) by Isaac Asimov

The excerpt from the collection of short stories, The Complete Robot, by one of the most influential science fiction authors Isaac Asimov narrates about a robot which was supposed to pilot a prototype spaceship with a hyperdrive, but had failed to take off. It turned out that the robot obeyed its orders too literally [Asi83].

Science fiction writers took communication between humans and robots using spoken natural language for granted long before first humanoid robots appeared and have foreseen many key questions of human-robot interaction. Failure of producing expected behaviour despite perfect syntactic understanding of the language shown in the short story touches one among most interesting aspects of language itself, namely the importance of contextualization. It shows why implementation of voice based human-robot interface is far from being trivial. Advanced technologies such as speech recognition, natural language processing and speech synthesis, as well as some approaches from artificial intelligence need to be utilized.

The goal of this bachelor thesis was to create a voice human-robot interface for an indoor service robot MetraLabs Scitors G5 using various open source frameworks. The robot should be able to react to simple commands and interact with persons using voice.

This thesis paper work is sectioned as follows: In the theoretical part, the need for advanced human-robot interfaces is discussed in the broader historical and humanistic context, followed by a presentation of crucial theoretical background on speech technologies needed for developing voice interfaces. In the Chapter 3, the used hardware and key software dependencies are presented. Chapters 4 and 5 are about the voice human-robot interface for the Scitos developed as part of this thesis. They describe its usage, development, the overall approach as well as the various original concepts. Discussion of the results and an outlook for possible improvements is given in Chapter 6. Finally, the work is summarized in Chapter 7. In addition to that, list of abbreviations and glossary are found in the appendix.
2. Theoretical background

This section consists of two parts. The first part presents the lively research in the field of human-robot interaction and elaborates on its importance. The second part provides reader with an introduction to fundamental concepts behind speech technologies, which are the key element of voice user interfaces, one of the possible ways to realize human-robot interaction.

2.1. Human-Robot Interaction

The field of Human Robot Interface (HRI) is an important area of research in robotics. The basic incentive is to create and investigate interfaces that enable natural and effective modes of interaction with robotic technologies. HRI is interdisciplinary study, bringing together methodologies and techniques from robotics, artificial intelligence, human-computer and human-machine interaction, cognitive sciences, and other fields.

The general interest of HRI research are methods for perceiving humans and their intents, or creating cognitive models for more situation appropriate behaviour of robots. Application-oriented HRI research puts focus on exploring motion planning, for instance in conjunction with both verbal and non-verbal communication signals such as bodily gestures humans extensively use.

Though not limited to that, human-robot interaction has been a topic of many speculations ever since first ideas to create human-like machines came to life. The next section provides a brief overview of ideas contributing to the development of the field.

2.1.1. Relation between humans and robots

First documented ambitions to mimic human behaviour with machines date to as early as 10th century BCE, when a Chinese ‘artificer’ named Yan Shi allegedly presented a marvellous automation, which could sing and was able to move in a man-like manner, to the then emperor of the Chinese empire [Nee91]. Throughout the history, out of various motivations, a great many of other people continued to deal with this question – whether or not it is possible to recreate a human being.

From its very beginnings, robotics seems to follow this inheritance. The term robot was coined by the Čapek brothers and is derived from a Czech word “robota” which literally means “serf labor”. It was originally used to describe artificially created servants who can be visually mistaken for humans, but was later generalised to describe reprogrammable machines which do some work. Industrial robots became widely used in manufacturing, assembly, packing, transport. However, these robots usually do not even distantly look like a human being.

First fully autonomous robots appeared in the second half of the 20th century and only recently rapid technological progress made it realistically to expect that it might be possible to create robots with an external structure built to adequately resemble that of the human body. This can be actually already partially achieved, as shown by the famous example of a Japanese researcher, who created a humanoid which is a faithful clone of his looks [Gui10].
However, it seems unjustified to reduce what it is “to be like human” to visual appearance, moving or other directly observable behaviour. One of the main interests of cognitive sciences is the way humans learn from sensory information and consequent acquisition of perceptual skills. This knowledge is used to develop computational models of human behaviour. The main tackle for robotics seems to be, that humanoids should not only be able to move, but to react to external stimuli appropriately, both in terms of starting the correct movement sequence and emotionally.

In order to perform their tasks completely autonomously in not pre-programmed situations, a way should be found, how human like AI could be embodied to the robot’s controlling circuit. Projects like OpenWorm have proven that it is possible to control moving of a simple Lego robot by copying structure of C. elegans’ neural system to a simple artificial neural network\textsuperscript{1}. In fact, fairly similar intelligent systems already show very promising results in much more demanding areas such as object recognition, whereas they already outperform human when it comes to data-supported decision making, finding optimal paths etc.

This addresses only a tiny fragment of the ambitious goals set for humanoid robots, which are supposed to replace human labour in disaster sites, on space missions or even in households. Especially in the latter case, still much has to be done, in areas such as language – from speech synthesis to the interpretative skills or unaided sentence generation.

### 2.1.2. Existing human-robot interfaces

Early human-robot interfaces were similar to standard human-machine interfaces. In fact, for decades robots were utilized mainly in industrial environments or remote locations, where innovative interfaces are either pointless or not applicable at all. However the robots evolved and are able of much more complicated behaviours. Traditional human-machine interfaces were not designed autonomous robots but stationary machines using them has some major drawbacks.

In many cases, the robot movements are controlled using a wired or wireless joystick. This allows great precision, but might become tedious after a while. Especially when having to navigate to distant points, one should consider using a goal planner, which can determine the best path to target and drive towards it autonomously.

To achieve this, using computers to control robots is to be considered. Humans already interact with computers in many ways, however the prevailing method is using Graphical User Interface (GUI), which enable input based on either pointing devices (such as mouse or touch screen) or keyboards.

Despite other limitations, this requires users to be familiar with desktop paradigms (using mouse still present problems for majority of elderly population) and their constant proximity to input-output devices. This not only presents difficulties for disabled users (for example in the case of paralysis of the hand) but fails entirely for independently moving objects such as autonomous robots.

### 2.1.3. Multimodal user interfaces and voice control

The need for developing human-robot interaction was discussed by science-fiction authors for a long time now, but the discipline started emerging when first humanoid robots appeared and approaches from traditional human-machine and human-computer interaction were proven to be insufficient or obsolete. Key approach is to mimic the way humans interact which each other - which is using different communication methods -a concept known as multimodality.

\textsuperscript{1}See project page at http://www.connectomeengine.com
Multimodal user interfaces try to address issues of traditional interfaces to enable a more hassle-free and natural communication involving all human senses. As the name suggests, this is achieved by combining multiple input and output modalities [SS05].

Specifically, multi-modal systems can offer an additional voice user interface to the existing methods, allowing user to interact through speech and to receive feedback from the system by means of speech synthesis. For user tasks, this is believed to be the most convenient and quickest way of communication with computer, while still allowing usage of traditional command line interfaces for task like system administration or providing input of sensitive personal data using more discrete input methods than dictating.

When designing a new voice interface, it is important to guide users through the system, for example by presenting the commands by the system itself at start or giving regular feedback that the command is being processed or was not successfully executed. It is also important to provide an ample amount of silence before the providing feedback, which allows users to finish their command without being interrupted by the system [Cha’06]. Theoretical background needed for creating voice interfaces is presented in the following sections.
2.2. Human speech

2.2.1. What is speech?

Speech is vocalized form of human communication. It is based upon the syntactic combination of words drawn from large vocabularies specific for the language spoken. Each of these words is created out of a limited set of basic speech units called phonemes.

A phoneme is the shortest segment of speech that, if changed makes a difference in the meaning of a word. For instance, the word *bit* contains phonemes /b/, /i/, and /t/. We know that they are phonemes because we can change the meaning by changing each phoneme individually. Thus, *bit* becomes *pit* if /b/ is changed to /p/, or *bat* if /i/ is changed to /a/, and *bit* changes to *bid* if /t/ is changed to /d/ [Gol08].

Phonemes are not the physical segments themselves, but merely cognitive abstractions of them. Phones refer to the instances of phonemes in the actual utterances - i.e. the physical segments. A common example to illustrate this difference are the words “madder” and “matter”, which are composed of distinct phonemes, but are pronounced almost identically in various English dialects, which means that their phones are very close in the acoustic domain or even same.

![Human anatomy connected with speech production](image)

(a) Pulmonic airstream mechanism: When diaphragm relaxes, the lung volume is reduced, causing an increase in pressure and hence a flow of air out of the lungs.

(b) Shape of vocal tract modifies the phonation: it can be altered by moving the articulators, which include tongue, lips, teeth, jaw, and palate.

Figure 2.1.: Human anatomy connected with speech production [Gol08]

Phones and thus speech are usually produced with pulmonary pressure provided by the lungs (Figure 2.1(a)). This pushes airstream past the vocal cords, which creates phonation, that is then modified by the vocal tract (Figure 2.1(b)) into different vowels and consonants.

Vowels are produced by vibration of the vocal cords controlled mainly by changing the position of lips. This change in shape changes the resonant frequency and produces peaks of pressure. The frequencies at which these peaks occur are called formants and can be treated as a characteristic of distinct vowels [Gol08].

Consonants are produced by a constriction of the vocal tract. For instance, producing the phoneme /d/ requires one to place their tongue against the alveolar ridge (see Figure 2.1(b))
and then release a slight rush of air. Similarly, producing /f/ requires one to place your bottom lip against the alveolar ridge and then pushing the air through it.

These movements of the tongue, lips, and other articulators create patterns in the acoustic signal that can be described using a sound spectrogram. The sound spectrogram indicates the pattern of frequencies and intensities over time that make up the acoustic signal. Frequency is indicated on the vertical axis and time on the horizontal axis; intensity is indicated by darkness, with darker areas indicating greater intensity [Gol08].

![Figure 2.2: Spectrogram for sentence “I owe you a yo-yo”](image)

Although, humans perceive speech easily under most conditions, underlying processes are complex. The spectrogram shown in Figure 2.2 illustrates just one of the challenges of speech recognition often referred to as the segmentation problem. It states that neither borders between phones nor words are easily drawn, because of the continuous nature of the speech. Next sections cover the basics about computer speech recognition systems.

### 2.2.2. Modelling speech

In most speech recognition systems, speech is understood as a continuous audio stream with dynamically changing states. In this sequence of states, two classes of sound units can be identified - dynamically changing and stable states - which both bear lesser or greater resemblance to phones.

To understand that, it is crucial to know that the acoustic properties of a waveform corresponding to a phone can vary greatly depending on its context. The so called co-articulation makes both phones sound differently as they would if spoken separately. Therefore, a phoneme can be understood as consisting of three subphonetic units: the first dynamically changing part of the phone depends on its preceding phone, the middle part is stable, and the last part changes dynamically depending on the subsequent phone.

Given that transitions between two consecutive phones are often more informative than the stable regions, phones are usually considered in context. Such phones in context are called triphones. Therefore, the waveform of the triphone /ə/ in /fən/ (fun) will be a bit different from the waveform of a different triphone /ə/ in /pən/ (pun).

For computational purposes, it is helpful to make detectors for parts of triphones instead of triphones as a whole. So for example, the stable part of a triphone /ə/ can be shared across other triphones to /ə/, reducing the amount of needed memory greatly.

Phones build subword units like syllables, which are for example important for determining missing phones from the context in languages with frequent elisions like German. Subwords
form words, which are interesting from computational point of view, mostly because they restrict combinations of phones significantly. Theoretically, English with its 44 distinct phonemes could contain $\sum_{i=1}^{n} 44^i$ words with length up to $n$ characters. Not all combinations are possible and an average native speaker of English does not use more than 20,000 words, which reduces the computing power for recognition greatly.

2.3. Speech recognition

Speech recognition (SR) has been defined as the ability to convert acoustic signal corresponding to spoken words into text. It is also known as automatic speech recognition (ASR), or just speech to text (STT). Although often misused, the term voice recognition refers to identifying the speaker, rather than what they are saying.

Speech recognition systems are already widely used in automatic call centres, but can also be applied for tasks like automatic aligning of subtitles to videos, or giving voice commands to computer.

2.3.1. Brief history of speech recognition

From the technology perspective, speech recognition has a long history with several waves of major innovations. Already in 1952 Bell Labs researchers built a system for digit recognition. Their system worked by locating formants distinguishing spoken word from other possibilities. The 1950s era technology was limited to vocabularies of around ten words and required the users to make a pause after each word [JR04]. Unfortunately, funding dried up for several years when the then research director of Bell Labs wrote an open letter comparing speech recognition to “extracting gold from the sea” [Pie69].

Nevertheless, some universities continued to research on the topic and in the late 1960s, first continuous speech recognition systems appeared. The pioneering work of Indian-born professor Reddy at Stanford University (later Carnegie Mellon University) was based on dynamic tracking of phonemes [BSH08, p. 525].

Around the same time, another key contribution to speech recognition was made by Velichko and Zagoruyko from the Soviet Laboratory of Pattern Recognition with the invention of dynamic time warping algorithm [ÂÇ69] which enabled operating on bigger vocabulary and provided foundation for speaker independent systems.

Rapidly increasing compatibilities of computers, opened doors for statistical approaches, which allowed researchers to combine different sources of knowledge, such as acoustics, language, and syntax, in a unified probabilistic model. Most notable contributions were made at IBM by the group led by Czech-born researcher Jelinek [IBM11], who regarded speech recognition more as an information theory problem and put less emphasis on emulating the way the human brain processes in favour of using statistical modelling techniques like Hidden Markov Models, which are still extensively used today along with newer approaches like n-grams.

Most recently, the field has benefited from advances in deep learning and availability of huge amounts of data in clouds. The advances are evidenced not only by the surge of academic papers published in the field, but by the world-wide industry adoption of a variety of deep learning methods in designing and deploying speech recognition systems. These industry players include (in alphabetical order) Apple, Baidu (China), Google, IBM, Microsoft, and Yandex (Russia) among others.
2. Theoretical background

2.3.2. Types of speech recognition systems

There are three ways of categorizing speech recognition systems; either in terms of supported vocabulary size, number of speakers, or speech style [Kač95].

Vocabulary size

Early speech recognition systems could recognize only limited number of words, due to low processing capacities. Today, most of the speech recognition systems are able to work with big vocabularies (around 100,000 words), but for some applications smaller vocabularies are more appropriate. Vocabulary size is usually correlated to the error rates, so in systems where accuracy is crucial and only a limited set of words is needed, such as direct voice input for operation of some aircraft, limited vocabularies are used.

Speaker dependence

If an application is speaker-dependent, the system will work only for one person and needs to be trained by the user before it can be used. The system adapts parameters to the user so even speakers with strong accents can use it. This type tends to perform better in terms of recognition rates, is easier to develop, but lacks flexibility. Speaker independent systems use a default language model, which causes lower recognition rates, but is usually used for telephony applications, which need to work for arbitrary speaker.

Speech style

In terms of speech style, we can define three main groups of speech recognition systems:

- systems for recognizing individual (isolated) words,
- systems for recognizing connected words, and
- systems for recognizing continuous speech.

When identifying isolated words, system requires a pause between each spoken word. The beginning and end of each word must be precisely marked, by making a pause between each word. This is the simplest form of recognition, because the system does not have to deal with the segmentation problem (described in the Section 2.2).

Recognizing words from signal consisting of more connected words is very similar to the identification of isolated words, but allows words to “run-together” with a minimal pause between them.

When recognizing words from continuous speech, words are linked together and often influenced by their preceding and succeeding word (most notable in French where some phonemes might or may not be spoken depending of the last sound in the word preceding it) and thus hinder identification. Such systems perform well when a person speaks slowly and clearly, and when there is no background noise. Systems that use this method of speech recognition are complex to develop because they can not presume any restrictions on what identifies word boundaries. Such system must also be able to adapt to the speaker, his articulation manner and speed of speaking.
2.3. Speech recognition

2.3.3. Functional principle of speech recognizers

Speech recognition consists of several steps [BSH08]:

1. capturing audio signal and preprocessing,
2. parametrisation and extraction of distinctive features,
3. decoding feature vectors, and
4. generating output.

Capturing audio signal and preprocessing

Recorded audio signal from microphone does not only contain speech data, but also noise from the environment. Noise removal can be done either at hardware level or using software algorithms. Furthermore, speech signal needs to be re-sampled to match frequencies of the reference data against which speech units are compared. Usually 16 kHz are used.

The signal is split into utterances - separate chunks of audio between pauses - with each of them being processed separately. One should note, that utterances do not necessary match with sentences, which are semantic concepts.

Parametrisation and extraction of distinctive features

Audio signal contains many pieces of information, most of which are redundant. First step is to do Fourier transformation of the waveform to get discrete data rather than continuous waveforms. Depending on a language only a fraction of parameters provide data that distinguish sound units from one another (for example tone is not important in most European languages, but is crucial for some of the Asian languages). Parameters providing enough variability to distinguish sound units are called distinctive features. They are extracted in each time frame and form a feature vector, that represents the speech unit.

Decoding feature vectors

Decoding feature vectors can be understood as matching features to language units using an appropriate model.

Usually, three models are used in speech recognition to do the matching:

- an acoustic model, which contains acoustic properties for each (tri)phone,
- a phonetic dictionary, which contains a mapping from words to phones containing possible different pronunciations, and
- a language model, which restricts word search by defining, which word could follow previously recognized words.

These mappings are language- and in some extreme cases even speaker-specific. They are generated by comparing reference texts with spoken versions of them using machine learning algorithms\textsuperscript{2}. The actual matching is specific from implementation to implementation. Description of the system used for this work is provided in Section 3.4.1.

\textsuperscript{2}This goes beyond scope of this work. Details are described in [HH92]
2.3.4. Accuracy and efficiency measures

Speech recognition systems are usually assessed in terms of accuracy and speed.

Accuracy is usually estimated by the word error rate. Given a reference text (original) of length $N$ words and its transcription (recognized text), it is calculated as follows:

$$WER = \frac{I + D + S}{N}$$

where $I$ stands for the number of words which were inserted to the recognized text in comparison to original, $D$ for deleted words and $S$ wrongly recognized (or substituted) words.

Other measurements of accuracy include the simple word error rate (which tells how often a single word was recognized correctly in different contexts or isolated) and command success rate.

Speed of automatic speech recognition systems are given by the real time factor. If the audio file contains 2 hours of audio (at the normal speed), and the decoding takes 6 hours, the speed is counted as $3RT$.

2.4. Speech synthesis

Speech synthesis is the artificial production of speech. In many ways, the manner of operation can be thought as a reverse process to speech recognition.

A Text-to-speech (TTS) system converts normal text consisting of words into speech, whereas older systems could only render phonetic transcriptions into speech. TTS is composed of frontend and a backend as outlined in Figure 2.3.

![Figure 2.3.: Schematic overview of text-to-speech system. Based on [BSH08, p. 414]](image)

The frontend pre-processes text containing symbols like numbers and abbreviations into the fully spelled out words and divides text into utterances based on punctuation. This process is often called text normalization. Phonetic transcriptions are assigned to each word. The process of assigning phonetic transcriptions to words is called grapheme to phoneme conversion. More advanced systems try to compute prosodic information (pitch contour, phoneme durations). Together, that makes up the symbolic linguistic representation. The backend then does the actual synthesis by converting the symbolic linguistic representation into sound.

Text to speech systems can be roughly divided in two groups, depending on which technology is used for the actual synthesis. These are systems using (a) concatenative synthesis, or (b) formant synthesis.

The first group functions as the name tells by concatenating pieces of recorded human speech segments that are stored in a database. Generally, concatenative synthesis produces
more natural-sounding synthesized speech, though there sometimes audible glitches at the parts where the output was stitched together from two or more recordings.

The other group is more interesting from the algorithmic point of view, because it does not use human speech samples but tries to create them.

### 2.4.1. Formant synthesis

The synthesized speech output is created by simulating waveform transformations as the would occur in the human vocal tract.

The basis for this is generating formants (see Section 2.2) using additive synthesis - technique that creates timbre by adding sine waves together.

Parameters such as fundamental frequency, voicing, and noise levels are taken into consideration. This enables direct synthesis of vowels. Consonants (and possible other sounds that are articulated with partial closure of the vocal tract) are done by deforming the waveform. By variating waveforms over time a waveform of artificial speech is created and played using a speaker. Operation of the specific transformations goes beyond the scope of this work; a schematic overview is given in Figure 2.4.

![Figure 2.4.: Schematic overview of a formant synthesizer [BSH08, p. 418]](image)

Systems based on formant synthesis usually generate clean speech without audible glitches, yet robotic-sounding. However, virtually every possible voice in different prosodies can be created by taking further parameters into consideration, a more natural sounding speech can be obtained.³

Rather than concatenative systems, formant synthesizers are usually small in size because they do need big databases of speech samples. They are commonly used in embedded systems, where memory available is limited.

³Recently deep learning approaches have been utilized to learn patterns on data containing different human speech samples, see for example [Zen13].
2. **Theoretical background**

2.4.2. **Overview of open-source TTS systems**

**Festival**

Festival is a TTS synthesis system and research platform developed at The Centre for Speech Technology Research at University of Edinburgh.\(^4\)

It can be used with several different voices, which provide models to convert typed text into audible speech. These include the standard Festvox diphone voices (included by default), the MBROLA voices from TCTS Lab of the Faculté Polytechnique de Mons,\(^5\) and the Arctic voices from the Language Technologies Institute at CMU.\(^6\)

It uses concatenative synthesis and provides Scheme-like interface, to write custom rules for text preprocessing and manipulation of prosodic information.

**eSpeak**

eSpeak\(^7\) uses the formant synthesis method, which allows support for many languages. It is possible to change pitch, colour (gender), and speed of the voices. It can be used as a tool for generating phonetic transcriptions to be used with other speech synthesis or recognition engines.

\(^4\)http://www.cstr.ed.ac.uk/projects/festival/
\(^5\)http://tcts.fpms.ac.be/synthesis/mbrola.html
\(^6\)http://www.lti.cs.cmu.edu/content/cmu-arctic-databases-speech-synthesis
\(^7\)http://espeak.sourceforge.net/
3. Platform

This chapter presents the hardware and software components used for the purposes of this thesis.

3.1. The Scitos G5 Robot

The MetraLabs Scitos G5 mobile robot\(^1\) was used as the development platform. Geometrical model is shown in Figure 3.1.

The robot is equipped with a differential drive with two high torque gear-motors that enable of translation up to 1.4 m/s and rotation up to 200 °/s. It is powered by 24V lithium batteries providing up to 12 hours autonomy.

![Figure 3.1.: URDF model of Scitos generated in RViz](image)

\(^1\)Full specifications are available on the official website: [http://metralabs.com/index.php?option=com_content&view=article&id=70&Itemid=64](http://metralabs.com/index.php?option=com_content&view=article&id=70&Itemid=64)
3. Platform

and closing of the eyelids (independently for each eye). In the upper part of the head, there is a
circular array of LEDs, which can be configured to show effects like blinking.

Figure 3.2.: Scitos G5 with its Human Machine Interface and RoboHead in the institute com-
puter museum

Rather than the microphones integrated in the Onboard-PC, Logitech C170 webcam, was
used. The microphone integrated in the camera is optimized for hardware noise cancellation
and extraction of human speech, leading to better speech recognition result.
3.1. The Scitos G5 Robot

3.1.1. Odometry

For many task, it is crucial that the robot knows his own position, for instance when it should move towards a certain point or map its own environment. Due to its simplicity, usually an approach called odometry is used for this task. Data collected from motion sensors is used to estimate change in pose - that means both position and orientation - for each small time interval. The accumulated data is then used to compute the offset from a known starting position [SK08, chapter 20].

There are several options to determine translation and rotation of the robot relative to its pose in last time interval. The simplest one is to use data sent to the motors in conjunction with a mathematical model of the robot, which tells how a specific command changes the pose. However, due to external factors such as friction, the actual pose reached usually differs from the expected one by a small error factor. Another common source of errors are discharging batteries, which results in small oscillations in power, so the expected velocity is not achieved.

Better results can be approached by using sensor data in conjunction to mathematical model. Inside of the wheel rims usually a strip with alternating white and black bars and an infra-red brightness sensor. When the wheels move, the sensor counts these bars. Using this data, it is possible to calculate both achieved speed and distance travelled. Similarly, data about rotation can be obtained.

Nevertheless, even using sensor data is prone to systematic errors such as measurement uncertainties. The longer the robot drives, the greater is the total error because of the accumulation of these errors, making this approach quite inaccurate and as such inappropriate to use as the sole source of positioning data. Instead, data fusion from different sources including laser scans (for indoor applications) or GPS (outdoor) is recommended. For our purposes, odometry was accurate enough.

3.1.2. Laser scanner measurements

Laser sensors are often used in mobile robots, as they enable distance measurements and thus detection of obstacles in the vicinity of the robot. They function by emitting an infrared laser beam.

Depending on the design of the sensor, the distance to the obstacle can be calculated by measuring the time needed for the reflected signal to come back or phase shift of the returning signal.

To determine position of the obstacle, the laser beam is steered into different directions by laterally moving the mirror in the interior of the sensor. Each measurement point is then determined in polar coordinates \((\varphi, r)\). The angle \(\varphi\) indicates direction of the beam (\(\varphi = 0\) usually means straight forward), the value \(r\) is the distance to the first obstacle in the scanned direction.

Typical applications for laser scanners are obstacle detection, mapping and localization. Operation of laser scanners is more thoroughly described in [SK08, chapter 22].

The Scitos used in this thesis, is equipped with a SICK S300 laser scanner\(^2\) with 270° scanning angle and 3 m scanning range.

\(^2\)Full specifications are available from the official website: http://www.sick.com/group/EN/home/products/product_portfolio/optoelectronic_protective_devices/Pages/safetylaserscanners_S300.aspx
3. Platform

3.2. Robot Operating System

The software basis for the project is the Robot Operating System (ROS) version Indigo. Although the name may suggest otherwise, ROS is not a real operating system, but a modular open source robotics framework providing versatile hardware abstraction to ease the access to sensors and actuators, message-passing system to enable the communication between different processes or running code across multiple devices.

The message-passing system is based on a publish-subscribe design pattern, where publishers send messages to subscribers over a server which allows them to obtain published data of interest. This concept enables a much more flexible network topology, because the sender does not need to be preprogrammed to send data to each specific subscriber directly. Additionally, there may be multiple concurrent publishers and subscribers, and a single publisher may publish to multiple subscribers or even subscribe itself to one of other publishers [Sch96].

![Diagram of ROS network with two publishers and subscribers communicating over /topic and /topic2](image)

Figure 3.3.: A simple ROS network with two publishers and subscribers communicating over /topic and /topic1

In ROS jargon the entities in such a network are called nodes. They are processes that perform computation. A robot control system usually consists of many nodes. For example, one node controls a laser scanner, the other one the wheel motors, another one performs navigation, and so on. Nodes communicate with each other by passing messages. A message is simply a data structure comprising arbitrarily information types (for instance coordinates of the goal in the map). The messages published to the so-called topics. The topic is usually named after the type of content of the message (for instance /navigation/goal). A node that is interested in a certain kind of data will subscribe to the appropriate topic. Nodes are usually not aware of each other’s existence. In order to find each other or exchange messages, they connect to the master node called roscore, which provides name registration, and lookup to the rest of the network, and parameter server for storing shared data. An example network is shown in Figure 3.3.

Despite the flexibility of the publish-subscribe model, such communication paradigm is not appropriate for request-reply interactions. In ROS they are done via services, which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service and a client uses the service by sending the request message and awaiting the reply [Ope].

ROS is distributed and comes with various packages, that provide common functionality expected from a robotics framework. For instance, when doing tasks with a robot, it is crucial for the robot to be aware of its own position as well as of the position of other objects in the

---

Available from the project website: [www.ros.org](http://www.ros.org)
world relation to the robot or vice versa. \( \text{tf} \) is a package that lets the user keep track of multiple coordinate frames over time. \( \text{tf} \) maintains the relationship between coordinate frames and lets the user transform vectors, quaternions, etc between any two coordinate frames at any desired point in time [Foo13].

The ROS Application Programming Interface (API) is implemented primarily for C++ and Python, but community provides semi-official implementations for various programming languages, Java, and LISP among others. C++ and Java implementations are used in this thesis.\(^4\)

### 3.3. Navigation stack

For navigation and localisation the already developed collection of packages (i.e. “navigation stack”) from The Chair of Cognitive Systems was used [MZ04].

It provides a ready to use node, which only needs one input - the target pose in world coordinates - and then performs the necessary calculations and controls the motors, so that the robot moves to the specified destination point. Besides the position, the desired speed and orientation of the robot at the target point can be set.

In order to achieve the goal with the correct orientation, the program performs following steps: First, an attempt is made to take the direct line that passes through the goal and the target point is. If there are no obstacles between the two points, the robot moves on straight to the target point and rotates to the wanted orientation. This behaviour has some drawbacks, because the robot always tries to travel the straight line first, often leading to weird behaviour, especially when there is a wall between the robot and goal. Instead of driving to left or right to circumvent the obstacle, the robot will first rotate in the direction of the wall, to see if he can drive directly to the goal and then again to left because it obviously can not drive through it. A partial work-around is to use maps of the environment, providing basic information of stationary obstacles.

An integrated controller attempts to keep the speed of driving as constant as possible. To avoid collisions, a simple collision avoidance is already integrated, which the stops the robot immediately, if the laser scanner detects a nearby obstacle.

In conjunction with the AMCL package, the navigation stack provides methods to query robot’s current position in statical map relative to a pre-set origin.

### 3.4. CMUSphinx

CMUSphinx is common name of a group of open source speech recognition systems.\(^5\) These include series of speech recognizers, an acoustic model trainer, along with various resources such as software for language model compilation and a public-domain pronunciation dictionary.

There is an existing ROS package, providing a simple wrapper around the simplest speech recognizer from the CMUSphinx family, pocketsphinx, written in C and aimed at embedded systems.

However, it lacks some key features needed for development of effective Voice User Interface (VUI), such as incomplete support for grammars, which are extensively used in this work.

\(^4\)More detailed information about implementation and usage of ROS as well as download links are available from the website of the project \(\text{www.ros.org}\). Rosjava is hosted by Github at \(\text{github.com/rosjava}\).

\(^5\)The project is hosted by SourceForge and is available for download at \(\text{cmusphinx.sourceforge.net}\).
For the purpose of this thesis, a Java based Sphinx-4 was used, developed as cooperation of Sun Microsystems, Mitsubishi Electric Research Laboratories, Hewlett Packard, and Carnegie Mellon University, which utilises HMM-approach to speech recognition.

It provides much more flexible framework for research in speech recognition in comparison to its lightweight counterpart pocketsphinx.

3.4.1. Architecture of Sphinx-4

The Sphinx-4 architecture is modular and consists of three main blocks controllable from an external application.

Any module in the system can be smoothly exchanged for another without having to modify other modules. In this thesis, for instance, the language model was changed from a statistical N-gram language model to a context free grammar.

The main blocks are Frontend, Knowledge Base, and Decoder. The Frontend module takes in speech and extracts characteristic features. The Knowledge Base provides the information for decoder to select only relevant features from the ones extracted in frontend. The decoder is the main block and performs the actual speech recognition using these features.

The communication between these modules, as well interface for external applications, are depicted in Figure 3.4.

![Sphinx-4 system architecture](figure3_4.png)

Figure 3.4.: Sphinx-4 system architecture [Lam+03a]
Frontend

Frontend is responsible for processing input data. Audio signal from microphone is usually sampled at 16 kHz and parametrized into a sequence of output features.

The frontend module consists of one or more parallel data processing chains as shown in Figure 3.5.

The chain consists of several communicating blocks, each with an output and an input linked to the output of its predecessor. When a block is ready for more data, it reads data from the predecessor, and interprets it to find out if the incoming information is speech data or a control signal, such as silence which marks the end of the utterance in most cases. If the incoming data is speech, it is processed and the output is buffered until request from the successor block comes.

Such design, also known as pull pattern, permits starting the chain at any of the intermediate blocks. This enables us to run the system using not only speech signals, but also features computed using independent information sources, such as contextual data in parallel to the features from the speech signal. So it would be for example possible to foster recognition of the speech utterances naming items using features obtained by object recognition software. The last link in the chain should encapsulate features in a format compatible with the decoder module.

Frontend can be run in four different modes of operation with respect to sectioning input stream [Lam+03a]:

- **continuous**, where data is constantly obtained from a stream of input speech (useful for semi-automatic generation transcriptions of large spoken texts),
- **push to talk**, where the user indicates both the beginning and the end of a speech segment,
- **click to talk**, where the user indicates the beginning of a speech segment but the system determines when the speech ends automatically, and
- **fully endpointed mode**, where the system performs explicit endpointing, determining both beginning and ending endpoints of a speech segment automatically.

Despite reduced accuracy, the last mode of operation is used for the purpose of this thesis, since robot should be operated also when its primary HMI (touch screen) is not in user’s direct proximity.

Endpoint detection is implemented using a simple algorithm that compares the “energy level”, that is simply speaking what humans perceive as the loudness, to predefined threshold levels. If a certain value is exceeded, then the segment is marked to determine the start of speech, and similarly when silence is detected, the incoming audio segments are discarded as non-speech segments. Processing time is thus not wasted for further analysis of irrelevant segments.
3. Platform

Knowledge Base

Knowledge Base compromises itself three modules: Acoustic Model, Dictionary and Language Model, which provide data to the Linguist module from the Decoder.

Acoustic Model contains Fourier transformed representations specific for each of the phones. These reference representations are compared against features extracted from the actual input in the decoder. Recognizers for distinct phones make use of Hidden Markov Model (HMM).6

Dictionary is used to map words into sequences of acoustic model elements. Usually, dictionary has the Arpabet pronunciation of all supported words (currently more than 134,000 words are included and can theoretically be recognized). Arpabet is a phonetic transcription code, which contains only a subset of IPA-recognized 39 phonemes, that are found in standard English language. Every phoneme is represented by one or two capital letters. Additionally, digits are used as stress indicators and are placed at the end of the stressed syllabic vowel. So for example phonetics (IPA: /f@'nEtIks/) can be represented as F1 AH N EH T IH K S in Arphabet.

Language Model contains a representation of the probability of occurrence of words. These can be determined using either statistical approaches like n-grams or graph-based approaches such as using context free grammars. n-grams determine probability of one word following another given the presence of n − 1 preceding words.7 In this work, the second approach is used, where each word is represented as a node in a graph and arcs represent the overall probability of the two words appearing next to each other. This is realized using JSpeech Grammar Format (JSGF), which is presented in detail in Section 4.1.1.

Language model as well as dictionary for the corpora of words needed for a designated speech recognition system can be compiled online using CMU’s lmtool8 or using offline language modelling tools like SRILM9

Decoder

The primary role of the Sphinx-4 Decoder block is to use features parsed from input audio in the frontend in conjunction with data from knowledge base to generate hypotheses about what the spoken input sentence might have been. The Decoder block itself comprises three modules: Linguist, Search Manager, and Acoustic Scorer.

![Figure 3.6.: Example search graph for a language model with vocabulary comprising only two words: one (ARPA: /W-AX-N/) and two (ARPA: /T-OO/). [Wal*04]](image-url)

The Linguist generates a search graph, tree of possibilities for the best hypothesis, that is used in the Search Manager to find the best one.

---

6The way how Hidden Markov Models operate goes beyond the scope of this work. For description of the model see for example [RN95, p. 578].
7See [MS99, chap. 6] for further details on usage of n-grams in natural language processing.
8Available at: http://www.speech.cs.cmu.edu/tools/lmtool.html
9Available for research purposes at: http://www.speech.sri.com/projects/srilm/
A typical Linguist implementation constructs the search graph using the language structure defined in knowledge base.

The search graph is a directed graph composed of search states (vertices) and transition probabilities (arcs). Each state in the graph represents either a component from the language model (words in rectangles), dictionary (ARPA phonemes in dark circles) or acoustic model of the phone (using HMM). An example is shown in Figure 3.6.

Search graph states can be either emitted or not. A state starts emitting, if search manager triggers a matching feature in the set of features coming from the Frontend.

At each step of the process, the Search Manager creates list of paths containing only emitting states. Based on the vocabulary size and chosen language model, either entire or merely a small portion of the potential search space is kept in memory, when generating the search graph. For instance, a simple application for recognizing only digits uses a search graph similar to the one shown in example above (Figure 3.6) and can check input for identity with each digit separately every time a new signal comes. On the other hand, a text transcription application cannot process the entire dataset at once must prune away some parts of the search graph.

Pruning is done using Acoustic Scorer, which gives score to each of the states adjacent to the already emitted ones, based on the probability of their emission. In the subsequent steps only the paths starting from the highest scoring states, will be checked (others are pruned and will not be considered any more). Paths are explored using depth-first search (to get list of all already emitting states) and breadth-first search (to compare scores of adjacent states). For a more detailed description of calculating the score and for an example how two competing words are compared see [Lam03a] and [Lam03b]. For description of breadth-first and depth-first search as well as pruning in general refer to [RN95].
4. Development of the voice Human-Robot Interface

Goal of this thesis was to develop a Voice User Interface (VUI) as a mean of Human Robot Interface (HRI) using freely available software or components developed at the institute and integrated with the ROS.

The robot should be able to recognize, interpret and execute simple commands using open source speech recogniser and interact with users using speech synthesis software.

Firstly, overall approach to the problem is presented. A special type of grammars is used, which not only constrains the search space for speech recognizer by defining the syntax, but also contains hints which simplifies parsing the underlying semantics greatly.

Description of developed ROS packages as well presentation of the basic functionality are given further below.

4.1. Grammars

Human communication is only possible if there are symbols, each with certain meaning that all those involved agree upon.

However, mutual agreement on meaning of symbols quickly becomes insufficient, even to describe a simple relation between two objects, let alone expressing more abstract concepts. This gap can be solved by defining conventions how groups of symbols can be used in conjunction with each other.

When it comes to language based communication, such conventions are outlined with using grammars. They usually describe the whole system and structure of a language, consisting of syntax, morphology and sometimes also phonology and semantics.

Simply said, grammar names the types of words, their respective function, and word groups that make up sentences.

Similarly, if we want to build an application for not only recognizing what has been said but also to get what kind of response behaviour user wanted to achieve by saying it, we need to define some kind of mapping from patterns that may be spoken to their meaning. Yet, despite numerous efforts of artificial intelligence researchers, computers are unable to learn grammars on their own without any previous knowledge like the children do in the process called language acquisition.

In most applications that deal with natural language user input, “grammars” are only implicitly present by defining a set of predefined commands to which computer can respond. Alternatively, they can be omitted in lieu of probability based models.

We took approach that makes a direct use of grammars to parse sentences in order to tell, if they are in language and more specifically, if our robot will “understand” them. This allows much greater flexibility compared to solutions using a hard-coded set of sentences and relatively big variability of valid input sentences (though still smaller than that achieved in systems using probabilistic methods). For example, multiple sentences can trigger the same action. Instead of
having to say “go in the laboratory” with exactly these words in exactly same word order each
time we want a robot to go to the laboratory, we can tell it to “drive”, “head towards”, “move
itself” to the “robot laboratory” or simply to the “lab”.

We use *Java Speech API (JSAPI)* to build grammars that indicates what words a user is
expected to say and in what patterns those words may occur.

### 4.1.1. JSpeech Grammar Format

The *JSpeech Grammar Format (JSGF)* is a platform-independent textual representation
of grammars for use in speech recognition. JSGF adopts the style and conventions of the Java
Programming Language in addition to use of traditional grammar notations (like the Kleene
star *, alternation | and the plus operator +) from the Backus–Naur Form.¹

**Code snippet 1** A simple grammar specifying commands for controlling movement of a robot

```java
#JSGF V1.0;

grammar movement;

<name> = scitos | robot;

<direction> = left {DIR(LEFT)} | right {DIR(RIGHT)} |
               (forward | ahead) {DIR(FORWARD)};

<transl> = (go | move | drive) {ACTION(TRANSLATION)}
           [to] [the] <direction>;

<rot> = (turn | rotate) { ACTION(ROTATION) } [for]
       (ninety degrees {ROT(90)} | around { ROT(180)});

<wait> = wait { ACTION(STOP) };

public <command> = [<name>] (<transl> | <rot> | <wait>);
```

The format was chosen for its well-thought structure, which will be illustrated on the example
grammar defining movement commands (Code snippet 1).

Every grammar is composed of a set of rules that define possible utterances. Rules itself
are combinations of speakable text and references to other rules. Units of speakable text (i.e.
everything that is not a reference to rule) are called tokens.

Each rule has a unique rulename. A reference to a rule is represented by the rule’s name in
surrounding angle brackets < >.

There are two types of rules, local and global ones. Local rules define parts of utterances and
have to be included in global ones (they are declared using modifier word public). Recognition
of utterances can start only in global rules. Parts of the utterances defined by local rules can
only be recognized as part of utterances and not separately. So for example simply saying “left”
would not result into any match but saying “go to the left” would.

¹Refer to [Sch01] for definition of BNF as well as of grammars in computer science context. See [Hun00] for
full specification of JSGF.
4. Development of the voice Human-Robot Interface

The only global rule in the given example is `<command>` consisting of references to other rules. If we take a look at the Line 18 where that rule is defined, one will notice new types of brackets. Square brackets `[]` define parts of utterance that is optional (so in our example the command may or may not start with a name). Normal brackets define groups and are usually used in conjunction with the vertical bar symbol `|`, which is the exclusive or operator. So, a valid command can start with a name followed by either the expansion of translation `<transl>`, rotational `<rot>` or wait `<wait>` command rule. The last, consists of another group of tokens that each describe one of alternatives to tell the robot to start moving into one of possible directions, defined by the `<direction>` rule.

Grammars are used in this thesis out of two main reasons, which are presented in the consecutive subsections.

4.1.2. Using grammars to constraint speech recognition

Grammars can be thought of state automata and thus represented as search graphs. The graph for our example grammar is shown in Figure 4.1.

![Figure 4.1.: The movement grammar represented as a search graph](image)

Utterances are defined as the set of all possible paths between the two silence nodes (dark red rectangles). The word representation of utterances is composed as concatenation of strings in the green rectangles on the path. One can note that some of the rectangles can be omitted by going from one circle directly to another, which corresponds to optional rules defined by the square brackets.

From the search graph it is easy to see that grammars in fact define which words can follow previously recognized words. Using grammars helps to restrict word search space.

This helps to significantly restrict the matching process by stripping out the words that are not predicted in our use cases (it doesn’t make sense to order a robot to do the dishes, if it doesn’t even have the actuators would enable it to move objects).

For the purposes of this thesis, three grammars are used. One similar to the example above for movement commands (including the RoboHead), one for setting navigational goals and querying own position, and finally one for small talk.

4.1.3. Using grammars to simplify syntactic analysis

The most powerful feature of grammars in the JGSF format are the so called tags. They are the blocks within curly brackets `{}`.

Tag attachments do not affect the speech recognition itself, but provide a mechanism to get semantic information from the utterances.
In the example search graph discussed previously (Figure 4.1), the blue blocks highlight the parts of utterances bearing actual information. This corresponds to rule expansions followed by tags in Code snippet 1. Considering lines 13 and 14, for example utterance “scitos turn around” contains tag ACTION(ROTATION) for word turn and ROT(180) for around.

For the purpose of this work, tags have to follow the predicate-value structure (for example: PREDICATE(VALUE)).

The tags are attached to the object encapsulating result utterance and returned for further interpretation (see Section 4.3.2).

### 4.1.4. Detecting Numbers

At the time of writing this thesis, Sphinx-4 did not support detecting numbers out-of-the-box. In order to permit detecting names of numbers (useful for naming rooms in the institute or telling the robot how far away it should move), a grammar specifying names of numbers up to a million was created (see Code snippet 2).

**Code snippet 2** Numbers grammar, which specifies all possible combinations of units so that only valid numbers can be detected.

```java
#JSGF V1.0;
grm numbers;
//units
<digit> = zero | one | two | three | four | five | six |
seven | eight | nine;
//numbers between ten and nineteen have special structure
<teen> = ten | eleven | twelve | thirteen | fourteen |
fifteen | sixteen | seventeen | eighteen | nineteen;
//tens
<ty> = twenty | thirty | forty | fifty | sixty | seventy |
eighty | ninety;
//magnitudes
<big> = hundred | thousand;
//compounds up to hundred
<tens> = ([<ty>] <digit>) | // e.g. twenty one, five
<teen> | // e.g. thirteen
<ty>; // e.g. forty
//more complex compunds, e.g. five hundred thousand,
two thousand five
<compund> = ([<tens>] <big>)* [<tens>];
//sequence of digits, e.g. one-one-two
public <digits> = ( oh | <digit>)*;
//numbers up to million
public <number> = <tens> | <compund>;
```

The grammar does not contain any tags, which would tell about the underlying semantics of the words (i.e. if they are tens or hundreds...). Because most programming languages, among others Java and C++ which are used in this work, can only work with numerical representations, the Sphinx-4 GrammarParser was extended, to check if the sentence contains any number
4. Development of the voice Human-Robot Interface

words. If it does, a simple algorithm outlined below is used to get the numerical representation and append it to other tags parsed from sentence.

Enums with string representation of (1) digits, (2) numbers from eleven to nineteen, (3) tens, and (4) magnitudes are used. Enum is a data type consisting a set of named values called elements, which name enumeration constants - each of them corresponds to a numerical value, in this case to the number the word is naming.

If the detected token containing words of numbers has only one word in it, it is simply checked with which of the enums they form an identity. If the case of word compounds, such as “three hundred twenty one”, objects like triples or quadruplets are used to represents units, tens and magnitudes separately (Code snippet 3).

**Code snippet 3** Structure of a triple object representing number 28. Tripples can be used for numbers up to a thousand.

```
triple ( 0, 2, 8 );
//      |   |   |
//      |   |   |-> * 1
//      |   |----> * 10
//      |-------> * 100
```

A switch statement is used to check position of words in the compounds, because order of the words matters (“hundred one” vs. “one hundred”). Full implementation can be found in the NumberParser class of the Sphinx-4 Wrapper package.

4.2. Map of known rooms

We use the ROS map_server\(^2\) package which provides a node with the same name that offers map data to other nodes as a ROS service.

Map itself is loaded from an image, that describes the occupancy state of each cell of the world by the colour of the corresponding pixel. Lighter pixels are free, darker pixels are occupied, and pixels in between are unknown. The map used for the purpose of this thesis is shown in Figure 4.2.

![Figure 4.2.: Unlabelled map of the Department of Cognitive Systems based on the floor plan. Some of the doors are permanently closed and were manually removed from the map by adding full lines to prevent robot from planning paths through non-existing gateways.](http://wiki.ros.org/map_server)
4.2. Map of known rooms

However, such representation of robot’s environment does not contain any labels; i.e. the rooms are not named, so the only way of telling the robot to go to a certain place is to give him the exact coordinates of that place in the map. One can imagine, that dictating coordinates through voice interface is very inconvenient.

To enable more natural voice interaction, the ROS parameter_server was used. Parameter server is a shared dictionary that can be used by any ROS node to store and retrieve parameters of different types at runtime. One of the important features of the parameter server for the implementation is its capacity to export data in a human-readable data serialization format YAML.

Code snippet 4 YAML showing a dictionary storing coordinates of the students’ computer pool and institute museum

```
rooms:
  museum: [101.75370992339758, 25.827493249080124]
  students: [64.049280012579161, 14.680146656916876]
```

However, this approach has its own limitations, as it only enables one directional lookup. That means, we can only send a request to the server to tell us the coordinates of the computer museum, but not vice versa, i.e. a request using these coordinates wouldn’t return name of the corresponding key. For this reason, parameter server is only used for storing or exporting data for usage in latter sessions. Internally, a hash map structure is used to enable such bidirectional queries.

To demonstrate usage of this map, several voice commands have been implemented, such as “you are in the corridor”, “where are you”, or “go to the kitchen”. They are presented in the Section 4.4.3.

[^3]: http://wiki.ros.org/Parameter_Server
4. Development of the voice Human-Robot Interface

4.3. Integration with ROS

Following the ROS software design patterns (see Section 3.2), the implementation of the human-robot interface is structured modularly and split into packages:

- **Voice Input**, which consists of speech recognizer and does some preprocessing,
- **Voice Interpreter**, which does semantic analysis of the input, generates output, and communicates with other software components such as head controller, and
- **Voice Output**, which consists of a wrapper for speech synthesis software.

Figure 4.3 shows how the voice stack is connected with the rest of the network.

![Diagram of voice stack connections](image)

Figure 4.3.: Packages in the voice stack publish to the `/cmd_vel` and `/scitos_head` topic and are exchanging requests with the navigation stack.

4.3.1. Voice Input

Obtaining voice input is done by the (voice) `input` package. The package is essentially a wrapper for the Sphinx-4 platform, used for speech recognition.

Sphinx-4 provides a Java API, therefore the node was implemented with rosjava. The wrapper consist of following classes:

- `GrammarRuleParser`,
- `NumberParser`,
- `RunNode`,
- `SentenceInputConsole`,
- `SentenceInputPanel`, and
- `TagsPublisher`,

Additionally to that, a Sphinx-4 configuration file, sample grammars, as well as a bash script for extracting and a java applet for visualizing grammars are provided in a separate resources folder.

The most important class is `TagsPublisher`, that implements the node (which we call “sphinx_node”), sets up voice recognition (reading configuration) and publishes tags to `/voice/cmd_server` for further command interpretation, that is described in section 4.3.2.

The configuration file `input.config.xml` parameters of the voice recognition. For example, Sphinx-4 is started in continous recognition mode and the language model are grammars.
GrammarRuleParser is a class which traverses the supplied grammars for possible commands tags in recognized utterances. NumberParser does the same with numbers (see Section 4.1.3 and 4.1.4 for details).

The GUI for ROS Voice Input Console is implemented through the SentenceInput* classes and is useful for debugging purposes.

The node also starts listener for the topic /voice/input/sentence, that can be used in conjunction with the Android app presented in the Section 5.2

### 4.3.2. Voice Interpreter

Syntactic parsing, or command interpretation which is how we call it, is done in nodes within separate interpreter package. The nodes are written using the standard C++ ROS API.

The main parts are:

- head_node,
- interpreter_node,
- navigation_node, and
- stop_node.

They all use abstract VoiceInterpreter class, which defines an interface for easier parsing of the tags and allows greater scalability. As described in Section 4.1.3, the tags should follow the PREDICATE_(VALUE) structure. In this way it is possible to generate a hash map from the sequence of tags describing the underlying semantics of the recognized utterance. The utterance “go to the kitchen” for instance would result in a tag string ACTION(NAVIGATE) ROOM(KITCHEN), which can be stored as two key-value pairs in the map, providing easier access to the relevant data in the further process.

Not all types commands are parsed in the same process. This is important since some of the processes are blocking loops. So for example, when a request is sent to the navigation stack (containing the position of a goal) the process will wait until a response with either a success message or failure is received. However, in many cases one would like to communicate with the robot even when it is following the path to the goal, for example to tell him to stop.

Therefore, each of the four nodes listed above subscribes to the /voice/cmd_server topic and waits until a new message comes, containing the right key that triggers its functionality. In the main, interpreter_node, the behaviour triggered by basic control commands (like changing direction) is outlined as well as some basic small talk commands are defined. As the name suggests, the head_node and navigation_node deal with commands connecting with controlling the RoboHead and navigation stack, respectively. The stop_node is of great importance even though it merely defines one behaviour, namely it mimics the (emergency) stop button found on robot, by resting bumber, preceded by cancelling of all navigational goals.

These “behaviours” include generation of the feedback sentences, which are in this version hard-coded, i.e. by defining output strings in the functions themselves rather than using more advances approaches including usage of grammars. The sentences are published to the /voice/tts/msg topic.

Some of the other interesting functionalities are described in the Section 4.4.

### 4.3.3. Voice Output

The voice output package contains wrappers for both TTS engines described in the Section 2.4.2.
4. Development of the voice Human-Robot Interface

Both festival_node and espeak_node subscribe to the /voice/tts/msg topic, and start the speech synthesis for the received utterance. Language and dialect of the later (and thus the used voice) can be changed by setting /voice/tts/lang_name and /voice/tts/dialect_code parameters.

When the robot is speaking, voice recognition is temporarily deactivating to prevent the robot recognizing text it said as user’s commands.

4.4. Selected functions

This section provides an overview of selected commands together with details about their implementation. A full list, showing great variability of possible sentences using relatively small grammars, is provided as an appendix.

We tried to exclude imprecise commands to avoid situations like the one in the excerpt from the short story in introduction. So rather than saying ”go left” one has to specify estimated distance to the target. This should prevent misunderstandings such as for instance, that by saying go left user might want the robot to go left in the corridor, but the robot could as well just turn left immediately and collide with wall.

4.4.1. Command: Drive forward/backward

The simplest of all are movement commands, which merely specify target linear velocity, which can be in its size either positive (driving forward) or negative (driving backward). It can be used in conjunction which increase/decrease speed commands.

They are published to the /cmd/vel topic, which triggers the differential drive.

4.4.2. Command: Move X meters forward/left/right

This enables setting the position and demonstrates the use of numbers grammar, which was discussed in Section 4.1.4.

When a command with action tag “MOVE”, direction tag specifying driving direction relative to robot’s current pose, and distance tag containing number is received, the robot will calculate the target pose in its own coordinate frame first.

To do so, first the current orientation is obtained (by querying the designated functionality in the navigation stack) as a quaternion. The quaternion for the target orientation is calculated as multiplication of the current orientation quaternion with the quaternion calculated from the wanted yaw (rotation in z-axis). For example going to left corresponds to positive yaw of $\pi/2$. Translation is calculated by setting x and y coordinates relative to the base.

The pose consisting of that translation and rotation needs to be transformed to the map frame, which is done using tf package.

When the goal is set, the robot will respond with a confirmation that he got the command: “Ok, I’ll drive one meter forward”.

4.4.3. Command: Describe your surroundings

This command demonstrates feedback generation abilities of the developed system.

When speech recognizer detects one of the following question describe your surroundings, where are you, what is the nearest room, the command tag ACTION(FEEDBACK)
4.4. Selected functions

CONTENT(NEAREST_ROOM) which tells the command interpreter to call the generateFeedback method with CONTENT(NEAREST_ROOM) as parameter.

With the limited set of data robot has, the robot can describe his surroundings by naming either the room where it currently is or the nearest known room.

In order to do that, the robot first has to query its own position (by querying a designated functionality of the navigation stack) and compare it to the coordinates of known rooms. In big maps such linear search is costly, so the nearest neighbour search was used, which returns the coordinates room with that lies most closely relative to the robot’s position. The algorithm is explained bellow.

**Nearest neighbour search using \(k\)-d tree**

Nearest neighbour search addresses an optimization problem for finding closest points the given query. Closeness is generally expressed in terms of a dissimilarity function: the less similar the objects, the larger the function values.

In our case of a two dimensional vector space representing points on the map, this problem can be formally described as follows: given a set \(R\) of points in a space \(M\) and a query point \(l \in M\), find the point \(r\) in \(R\) so that Euclidean distance between \(l\) and \(r\) will be smaller than for any other point in \(R\). Euclidean distance \(d\) between \(l\) and \(r\) is calculated using:

\[
d(l, r) = \sqrt{(l_x - r_x)^2 + (l_y - r_y)^2}
\] (4.1)

The actual search is performed using \(k\)-dimensional trees [RN95]. The \(k\)-d tree is a binary tree, with nodes representing \(k\)-dimensional vectors. In each node, depending on its depth in the tree, one of the components is selected as origin of the axis against which the values are compared. For example, if the \(x\)-axis is chosen, all vectors with a smaller \(x\) value than that of the node will be placed to the left subtree and all those with a larger \(x\) value to the right subtree. For better understanding algorithm is presented in code snippet 5.

**Code snippet 5 Construction of a \(k\)-d tree**

```java
KDNode kdtree(List<KDNode> points, int depth) {
    // select axis based on depth
    int axis = depth % k;
    // sort point list according to elemets at axis
    sortList(points, axis);
    // choose median as the pivot element
    int median = getMedian(points);
    KDNode node = points.at(median);
    //split list at median construct subtrees
    List<KDNode> pointsBeforeMedian, pointsAfterMedian;
    node.leftChild = kdtree(pointsBeforeMedian, depth+1);
    node.rightChild = kdtree(pointsAfterMedian, depth+1);
    return node;
}
```

\(k\)-d trees allow searches involving a \(k\)-dimensional search key (in our case point from a two dimensional map).

Searching for the nearest neighbour in a \(k\)-d tree is a recursive procedure. Starting with the root node, the algorithm moves down the subtree; it goes left or right depending on whether the
4. Development of the voice Human-Robot Interface

point is lesser than or greater than the current node in the dimension defined by the axis (line 3 in the snippet 5 above).

Once the algorithm reaches a leaf node, it saves that node point as the current best.

The algorithm unwinds the recursion of the tree and does following checks on each node whether it is closer than the current best (resulting in updating current node) or whether there could be any points on the other side of the subtree that are closer to the search point than the current best by comparing other dimensions of current best and the non explored sub-tree (resulting in moving down the subtree recursively until the next leaf node is reached)

The algorithm finishes when getting back to the root node and the recursion is fully unwound.

Generating response

After the position of the nearest room has been found, robot gives feedback in natural language.

Depending on how far away the room is there are three possible ways of generating the response string:

- If the Euclidean distance (see Eq. 4.1) between the current position and the saved position of the nearest room is smaller than 1 meter, it can be assumed that the robot is in the room. A sentence like “I am in the big laboratory” will be generated.
- If the Euclidean distance is greater than 1 meter but smaller than 3 meters, then the robot will say something like “I am near kitchen”.
- If the Euclidean distance is greater than 3 meters, the response will be like “I am near office, which is 20 meters away.” Distances are rounded to the nearest meter.

As the name of the room the key string of the entry in the map of known rooms is used.

4.4.4. Commands for moving the RoboHead

Simple “emotional” models can be built using the RoboHead. The robot might for example, wink if he understood a command or shake its head, when he can’t process the request.

As a side product, a node was created which can move the eyes, eyelids, tilt and pan head, and control LEDs array on the Scitos’ RoboHead. This effects can also be achieved using voice commands.
5. Usage of the voice Human-Robot Interface

This chapter provides a short handbook on how to use the developed voice interface.

5.1. System requirements and configuration

To use the developed voice user interface, ROS “indigo” or newer has to be installed, including following extra packages: tf, amcl, std_msgs, sciros_msgs, and full navigation stack with its dependencies\(^1\). OpenJDK Java 7 or newer and rosjava are required to run the voice input console.

Furthermore, either espeak (including development libraries) or festival has to be installed needed for performing text-to-speech synthesis.

For the correct functioning of the voice stack, at least following two nodes should be started before running packages from the voice stack.

- roslaunch scitos base.launch
- ROBOT_CONTROLLER=scitos_othexp roslaunch navigation / rviz_controlled_full.launch

The nodes from the voice stack should be started using the provided launch files, which set some of the parameters:

- roslaunch tts (espeak|festival).launch
- roslaunch voice sphinx.launch
- roslaunch interpreter scitos.launch

When no other applications are running, the default Scitos On-Board computer is powerful enough to run all three components of the voice user interface along with navigation stack. The most resources demanding components are Sphinx-4 Decoder and path planner from the navigation stack.

Given a reliable wireless connection, it is possible to run ROS nodes across multiple machines to optimize performance.\(^2\)

The software was tested on Ubuntu 14.04 “Trusty Tahr”, but it should work on any *nix system supported by ROS.

---

\(^1\)For this work the improved navigation stack developed internally at the Chair of Cognitive Systems was used ([https://gitlab.cs.uni-tuebingen.de/apps/navigation](https://gitlab.cs.uni-tuebingen.de/apps/navigation)), but it can be easily exchanged for the standard ROS navigation package

\(^2\)The ROS Wiki contains more information on network setup along with some practical examples [http://wiki.ros.org/ROS/NetworkSetup](http://wiki.ros.org/ROS/NetworkSetup)
5. Usage of the voice Human-Robot Interface

5.2. Input methods

5.2.1. Voice input

The primary way of passing information is using the voice user interface, which is quite self explanatory. When the system is up and running, the robot greets the user by saying something like “Hello, what can I do for you ...”. If the user doesn’t know any of the voice commands, the robot can briefly explain what capacity it has.

If that fails, it is probably due to misconfigured sound settings. Using various tools, such as PulseAudio Volume Control, it is possible to check if the sound signal is being recorded and pipelined to the correct sink, namely to Java/Sphinx-4 Audio Stream. Alternatively, removing the local configuration using `rm -Rf $HOME/.config/pulseaudio` might help.

![PulseAudio Volume Control](image1.png)

Figure 5.1.: PulseAudio Volume Control is a tool to control the sound system used in Ubuntu.

The voice input node should ideally run on the robot itself, so that it captures sound input from the robot and not server. However, it is possible to configure the PulseAudio system to be used over network too.3

5.2.2. Touch-screen interface

In order to reduce dependence of potentially error-prone speech recognition system, especially in noisy environments, where more than one person is speaking, user is also able to provide input in natural language using virtual keyboard displayed on the integrated touch display.

![ROS Voice Recognition Console](image2.png)

Figure 5.2.: ROS Voice Recognition Console enables input through Scitos’ human-machine interface using virtual keyboard.

3Detailed instructions are available on StackExchange [http://superuser.com/a/432954/175643](http://superuser.com/a/432954/175643)
5.3. Android application

If the virtual keyboard does not show up automatically, one can activate it by calling onboard command from within Ubuntu’s application’s dashboard.

This applet also has debugging function; parsed tags are shown in the command history frame if the entered sentence was grammatically correct, independently of its source (direct voice input, using the applet, rostopic tool or the Android app).

5.3. Android application

Android phones come with an integrated API allows access to the Google’s superb cloud speech recognizer (marketed as “Google Now” or “Ok, Google”).

Using the Android port of the rosjava project, a simple Android application was developed, that enables communication between ROS nodes running on the phone and ROS master on the robot.

![Android ROS Voice Input](image)

(a) Configuration options

![Default screen](image)

(b) Default screen

![Response screen](image)

(c) Response screen

Figure 5.3.: Simplistic user interface of Andoid ROS Voice Controll app developed as part of this work

In order to use the app, both devices should be in the same network and the ROS master on the robot should be set up for network usage. That essentially means, that some environment variables should be reset:

```bash
1. $ export ROS_IP=http://IP_OF_THE_ROBOT:11311
2. $ export ROS_MASTER_URI=http://IP_OF_THE_ROBOT:11311
```

By typing `hostname - I` into the command line, one can determine the IP address of the robot in the internal network.

---

4 Project page is hosted by the ROS wiki: http://wiki.ros.org/android, useful tutorials are also available from the EPFL Wiki: http://wiki.epfl.ch/roscontrol/androidstudio-and-rosjava
Next step is to copy the package file (*sentence-forwarder.apk*) to the phone and install it by opening the file. Installation from untrusted sources should be enabled to do so (our application is not available over Google Play Store). Alternatively, one might want to use the Android Debug Bridge or Android Studio\(^5\) to compile app from sources before installing with (USB Debugging has to be activated on the phone):

```
adb install sentence-forwarder.apk
```

If the app does not start automatically, it can be found in the app drawer as “Android ROS Voice Input”.

When the application on phone is opened, a simple GUI appears asking user to provide the IP address of the ROS master (Figure 5.3(a)). As soon as the app successfully connects to the ROS master, user can enter commands using integrated speech recognition (depending on the Android version one has to press the microphone button before speaking). When the text was recognized, it can be sent to the grammar parser node on the ROS master, which forwards parsed tags to the interpreter, which will finally send a response to user’s query in a natural language.

One should note, that Google Speech Recognition API does not use grammars, so it can recognize any possible combination of the words, leading to sentences that will not be processed by the interpreter node.

\(^5\)Available from [http://developer.android.com](http://developer.android.com)
6. Final considerations

6.1. Discussion

The speech recognition using Sphinx-4 is not perfect. Following designing guidelines for voice interfaces, to avoid confusion of the human user, the robot says that if command was not recognized, but not why.

Sometimes commands are not recognized. The usual trick to foster recognition results, reducing the vocabulary size, is not applicable to this situation, since bigger vocabulary enables greater preciseness of voice commands, which is crucial for successful communication (see Section 4.4). Better results can be achieved by generating personal acoustic models adapted to the target user (using tools like sphinxtrain) or, as suggested in [Cha+06], the set of commands could be altered that the voice commands different in contour of the spoken text reducing possible substitutions significantly.

The voice output sounds extremely robotic, when using system default configuration for both eSpeak and Festival. Using different voice models in Festival TTS engine can solve this problem.

Sometimes the robot drives zigzag along the paths which should be straight lines. The main reason for that is the buggy middleware for controlling the differential drive of the robot. It has been suggested, that upgrading from the original drivers to newer drivers from the Mira-Project should solve this problem. Unfortunately, the robot could not be upgraded to new software at the time of writing present work.

6.2. Summary

A human-robot interface enabling interaction of user and robot using voice was developed as part of this work, using multiple open source technologies: ROS, rosjava, and Sphinx-4. The system was developed for the Scitos G5 research robot platform by MetraLabs, but can be used for other robots as well with minimal changes in code.

In order to show flexibility of the system and to avoid dependence from sometimes unreliable speech recognition results from Sphinx-4 (in noisy environments with more than one speaker, on systems with low quality microphones, or when users do not master English pronunciation), two additional input methods were designed.

Using the Android port of rosjava, an alternative input method was developed in a form of application, which can utilize Google cloud speech recognition technologies with a very small word error rate. However, a network connection is not always available (or the end user might have privacy concerns). Therefore, users can provide their input by entering natural language commands in their written form using either a normal keyboard or a virtual keyboard on the touch screen of the Human-Machine interface installed on the robot. The GUI can also be used for debugging purposes.

Together with the existing interfaces such as teleopration realized with joystick and GUI for setting navigational goals (through rviz), the voice user interface developed, forms foundation
for a multimodal human-robot interface, which could for example be enchained with image object recognition to create distinct interaction behaviours for different situations or targeted for specific users.

An innovative approach to syntactic command interpretation has been taken by utilizing grammars (otherwise used primarily for restricting search space of the speech recognition software). The command interpreter is written in C++ and implemented as ROS node, is able to work with different language realisations of the same intent by using underlying semantic information defined by the grammar (i.e. position in the map can be queried by saying “where are you” or “in which room are you”).

Currently the set of voice commands is limited to teleoperation, controlling RoboHead, navigation and small-talk, but can be easily extended by writing new designated grammars. The solution presented is follows paradigms of object-oriented programming, so new functionality outlined by new grammars can be added by extending developed programming interfaces in conjunction with other software without modifying existing code.

Finally, to provide voice output of the generated response sentences, either a wrapper for eSpeak or Festival can be used.

6.3. Outlook

Speech recognition could be improved as suggested in the discussion. Furthermore, the system could be extended to provide a software noise reduction technology.

Currently, the generation of voice output is mainly hard coded for simplicity reasons. However, it is possible to make use of grammars to produce sentences too. Generation of sentences using relevant tags includes creating response grammar, traversing grammar in its graph form with algorithms like depth-first search, saving all possible sentences along with mapping between sentences and the underlying tags. The mapping can be used to find one of possible sentence realisations by looking up for tags, which would add variability to the robot responses making the interface more natural.

Currently, the robot signalizes that it finished an action by changing the blinking modes of the LED diodes. This could be extended to mimic human emotions, which are often subject of how good we are at accomplishing our tasks. To do so, it changing face mimics using architectures like WASABI could be utilized [Bec08]. Additionally, prosody of the voice output could be altered depending on the “emotional” state of the robot (it might be “happy” when it achieved a goal or “sad” if it could not find a path to the target position).
## List of Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface.</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface.</td>
</tr>
<tr>
<td>HMI</td>
<td>Human-Machine-Interface.</td>
</tr>
<tr>
<td>HMM</td>
<td>Hidden Markov Model.</td>
</tr>
<tr>
<td>HRI</td>
<td>Human Robot Interface.</td>
</tr>
<tr>
<td>IPA</td>
<td>International Phonetic Alphabet.</td>
</tr>
<tr>
<td>JSAPI</td>
<td>Java Speech API.</td>
</tr>
<tr>
<td>JSGF</td>
<td>JSpeech Grammar Format.</td>
</tr>
<tr>
<td>ROS</td>
<td>Robot Operating System.</td>
</tr>
<tr>
<td>SR</td>
<td>Speech recognition.</td>
</tr>
<tr>
<td>TTS</td>
<td>Text-to-speech.</td>
</tr>
<tr>
<td>VUI</td>
<td>Voice User Interface.</td>
</tr>
</tbody>
</table>
### Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>differential drive</td>
<td>A differential drive consists of two independently controlled wheels placed on both sides of the axis of the vehicle. A rotation of the vehicle is achieved by difference in the rate of rotation between the wheels. If they are not turning at exactly the same rate the robot will veer to one side, so no additional steering is needed. To balance the robot, a third non-driving wheel or caster may be added. Despite of their slipperiness, such systems are extensively used in robotics because of very simple and inexpensive construction.</td>
</tr>
<tr>
<td>elision</td>
<td>In linguistics, elision is the omission of one or more sounds (usually a phoneme) in a word. For example, the so-called schwa-elision is common for colloquial German, leading to omission of word-final phoneme /ə/ in word &quot;habe&quot; (ich hab' gestern nichts gegessen).</td>
</tr>
<tr>
<td>graph</td>
<td>In graph theory, a graph $G = (V, E)$ is defined to be a mathematical structure that forms a diagram from a set of objects $V$ called vertices (or nodes) and set $E$ of links between some of these objects called edges. Directed edges are called arcs.</td>
</tr>
<tr>
<td>prosody</td>
<td>In linguistics, prosody (from Ancient Greek for “song sung to music”) is concerned with speech properties of prosodic units (syllables, clauses, sentences), such as intonation, tone, stress and rhythm and their informational content (by changing prosodic properties the speaker might hint at the presence of irony, etc.).</td>
</tr>
<tr>
<td>timbre</td>
<td>In music, timbre describes the quality of tone. It distinguishes different tones of same pitch and loudness based on the presence of different non-fundamental frequencies in the spectrum. In simple terms, it describes what we call colour of human voice and that what makes different musical instruments sound differently.</td>
</tr>
</tbody>
</table>
A. List of the supported voice commands

// generated using GrammarTraversal.java (terminals are substituted only once)
close eyes
close your eyes
drive faster
drive right for <number>
drive slower
drive to <room>
drive backward
exit the program
go to <room>
go to the forward for <number> second
drive backwards for <number> seconds
drive in the <room>
drive right
help
hi
listen this is <room>
listen you are in <room>
listen you are in the <room>
mov forward
move the backwards
move the forward for <number> seconds
move to left for <number> seconds
move to right
move to the left for a while
move to the left for <number> meter
navigate in <room>
navigate to <room>
navigate to the <room>
open your eyes
stop
tell me about neighborhood
tell me what can you do
this is <room>
this is the <room>
turn off the lights
turn on the lights
turn on your lights
walk the forward for <number>
walk to the right for <number> meter
where are you
which places are near you
you are in <room>
B. Source code on the CD

This thesis paper comes with a CD containing a digital version of this thesis paper, source code and binary package of Android application, and the full source code of the ROS packages as well as other tools developed as part of the work, and list of dependencies from other packages.

If not otherwise stated, the code is released under the terms of GNU General Public License (GPLv3) and as such distributed in the hope that it will be useful, but without any warranty. See http://www.gnu.org/licenses/ for more details.

Comments, criticism and suggestions are appreciated and can be communicated to the author via smihae@gmail.com.

A copy of the CD content is available at: http://smihael.eu/rosvoice.
Bibliography


