**Postgraduate course**

**Virtual Reality and Multimodal Interaction**

# Speech recognition for Virtual Humans

**Report**

Ján Cíger

Lausanne 2002

# Contents

# Chapter 1

# Introduction

“Open the pod bay doors please, HAL.”

“I’m sorry Dave, I’m afraid I can’t do that.”

The idea of machines understanding human speech and communicating with humans in their own language is haunting scientists and sci–fi authors alike since the beginning of modern times.

Interaction between humans and computers evolved from punch cards and teletypes through video terminals into the systems which are in common use nowadays – e.g. mouse and keyboard combination. There were always attempts to make the interaction with the machine less “machine–like” and more “human–like”, which met with various levels of success and acceptance.

The speech recognition was viewed for long time as the ultimate goal of HCI[1] research, because speech is very natural for humans to use for communication. More recently, the focus of the research shifted more towards multi–modal interfaces e.g. combination of speech with gestures or keyboard. But speech recognition still plays a major role in these efforts.

The research in this field was hampered by inadequate computing resources

[1]Human to computer interaction

and unavailability of the good mathematical models of the recognition process and speech itself for long time. Until the advent of modern electronics and computing, together with the progress in science, particularly in signal processing, acoustics and language modeling, machines recognizing the speech and "understanding"[2] it were only possible in sci–fi novels.

The quote from the beginning is from the famous novel "2001: Space Odyssey" by Sir Arthur C. Clarke, first published in 1968 and later made famous by the movie of Stanley Cubrick. The quote itself represents the Holy Grail of speech recognition and language understanding by machines, still unfeasible by the state–of–the–art systems in the most part – continuous speech recognition by a speaker and environment independent intelligent system.

This work is the exploration of the world of speech recognition for Virtual Reality applications. Our laboratory is focusing on the virtual humans, thus one of the natural extensions of the virtual human paradigm is the ability to "understand" speech. The next few chapters describe how such goal may be accomplished.

The first few chapters are general introduction into the speech recognition technology. How it works, what to expect. Later, a system developed as a part of this project, is described, which allows control of the in–house developed VHD++ framework by voice. The final part of this report is a summary of results achieved and lessons learned in the process.

[2]"understanding" is quoted, since even modern systems do not understand meaning of the message spoken in the sense how human does. That is the field of Artificial Intelligence, where the state–of–the–art is still far from the capabilities needed even for simple conversation. But it can be "fooled", see e.g. [1]

# Chapter 2

# History of speech recognition

The first machine to recognize human speech was a toy – a celluloid dog, the "Radio Rex". The simple electromechanical toy from 1920 was capable of jumping, when it's name was spoken [2].

During the late '40s of the 20th century, the U.S. Department of Defense was sponsoring development of the machine capable of automatic translation of intercepted Russian messages. This project failed, but nevertheless sparked interested in the field and started research at MIT, CMU and some commercial institutions.

During the 50's of the 20th century, first system capable of recognizing digits spoken over the telephone was developed by Bell Laboratories. In early 70's, the system HARPY capable of recognizing complete sentences containing limited grammatical structures was developed by CMU.

The main problem of these early efforts was a lack of computing power – the HARPY required so much computing power as in 50 contemporary computers. Moreover, the system recognized discrete speech[1], not continuous speech.

The research continued, the milestones were marked by moving towards sub–word modeling (e.g. phonemes) and better language modeling. As mentioned in

[1]Speech, where words are separated by longer pauses than usual, to make the recognition easier

[5], these years were devoted to solve the “matters of scale”. What worked on few examples in the laboratory, didn’t work in the real world. What worked fine with limited vocabulary, didn’t scale to large vocabularies and a different approach was needed. Nevertheless, the available CPU power remained a problem.

As the available CPU power increased, first commercial applications appeared in the middle of 90’s of the 20th century. One of the first was VoiceBroker deployed by Charles Schwab, stock brokerage, in 1996. Later on, more consumer oriented products appeared - in 1997, Dragon introduced Naturaly Speaking, the first continuous speech recognition package available. Other well known product is ViaVoice (or as originally called – VoiceType) by IBM. It was first distributed with the now almost forgotten operating system OS/2 in 1996. Originally only for English language it gave the operating system first-of-the-kind capability to be controlled by voice and to allow text input by speech.

# Chapter 3

# CMU Sphinx

## 3.1 What is Sphinx

The CMU Sphinx[4] is a speech recognition package developed by the CMU Speech group[3] under the funding from DARPA and NSF for more then fifteen years. It is the state–of–the–art speech recognition technology capable of recognizing continuous speech with large vocabularies.

Sphinx evolved since the late 80's of the last century, into a very robust and mature system. In the year 2000, two versions of the software – Sphinx II and Sphinx III were released as open source software under a modified BSD license. The source code is available[1] and the system works on a variety of platforms out of the box.

Sphinx II is the older, a bit less accurate but faster system intended for real–time usage. Sphinx III is slower but more accurate version, developed more recently. There is also Sphinx IV, which is in the early state of development. It is supposed to be Java based, thus platform independent and easily usable on the wide array of Java capable computers.

[1]http://sourceforge.net/projects/cmusphinx/

For the most part, when, speaking about Sphinx in this report, Sphinx II is assumed.

## 3.2 Why Sphinx ?

The decision to adopt Sphinx for this project was not simple and few alternatives were considered.

- **Microsoft Speech API**

  Microsoft Speech API is a proprietary solution of the Microsoft Corp., now available with MS Windows 2000. The original speech recognition code for MS Speech API is based on CMU Sphinx engine.

  **Pros :**

  - Ready made, “pre–packaged” product, working almost out of the box.
  - Nice training and development tools available, the SDK is available for download

  **Cons :**

  - Inflexible – source code is proprietary, not available
  - Only three languages supported – U.S. English, Japanese and simplified Chinese
  - Impossible to train own acoustic model, only to adapt to the existing one
  - Works only on MS Windows

- **ViaVoice by IBM**

  ViaVoice is a product of IBM Corp. Allows both continuous voice dictation and command & control mode.

**Pros :**

- Ready made, “pre–packaged” product, working almost out of the box.
- Nice training and development tools available, the SDK has to be purchased separately
- Supports many languages – English, French, German, Italian, Spanish, Portuguese, Japanese and Chinese

**Cons :**

- Inflexible – source code is proprietary, not available
- It is difficult to get support for it from IBM, their support of the product is rather poor
- Impossible to train own acoustic model, only to adapt to the existing one
- Works only on MS Windows and Mac OS, Linux version is not supported anymore

• **Sphinx II by CMU**

Sphinx II is a product of the CMU Speech group, released as open source software.

**Pros :**

- Source code is available for easy modification and bug fixing
- Has an active development community, so getting support is rather easy
- Default acoustic model is only for the U.S. English, but it is easy to train own models with the tools provided. Even multilingual recognizers (e.g. recognizing English and French words at the same time) are possible.

- Compiles out of the box on many platforms – Windows, Linux, various commercial Unices, even on Mac OS
- It is easy to scale it down to the needs at hand, recognizing of just few words does not require hundreds of megabytes of data

**Cons :**

- Poor documentation, especially for people not working in the speech recognition field
- Non–trivial training process, it is necessary to master a lot of command–line tools to train a usable system
- Out of the box supports just one language – U.S. English. But this is partially mitigated by the option of training own models

In the light of these choices and our requirements (simple, cross–platform system, with access to the source if possible), the Sphinx was the only viable platform to use.

# Chapter 4

# Basics of Speech Recognition

## 4.1 Introduction of terms

In the following parts, some non-obvious terms are going to be used, which will be introduced here.

- **Phoneme**

  It is a basic theoretical unit, which may change the meaning of the word in the spoken language. The amount of phonemes depends on the language.

- **Triphone**

  Phoneme, for which the context of preceding and following phoneme is know. This is used to model coarticulation effects.

- **Monophone**

  Standalone phoneme.

- **Word Lattice**

  List of alternative words for each frame of the input signal, together with their probabilities

- **Acoustic model**

  Mathematical model of the acoustic conditions of the speech recognition system. It depends on the characteristics of speaker (e.g. his/hers vocal tract and the way of speaking), on the characteristics of the transmission channel (e.g. microphone, speech capturing hardware) and on the environment (e.g. noise)

- **Cepstrum**

  Cepstrum (pronounced “kepstrum”) is a result of Fourier’s transform on the spectrum of the original signal in decibel scale. Or informally: $cepstrum = FT(log(FT(x)))$. It is often used as a feature set for speech recognition, because it allows good separation of the signal resulting from the vocal cord vibrations from the “distorted” signal formed by the rest of the vocal tract.

- **Transcription**

  Textual representation of the spoken text.

- $N$**–grams**

  $N$–gram gives the probability of the word $X$ occurring in the text given the context of the previous $N - 1$ words. It is used in statistical language modeling.

- **Dictionary**

  In the context of speech recognition, it is the list of all words system recognizes. For Sphinx it is a file containing list of all recognized words together with their phonetic representation.

- **Language model**

  Statistical model of the language the system is supposed to recognize. For Sphinx, it is $N$–gram based. Gives probabilities of words occurring in the text given the context. This is very important for recognition precision, because it weeds out sentences which are not occurring in the normal language.

## 4.2 How it works

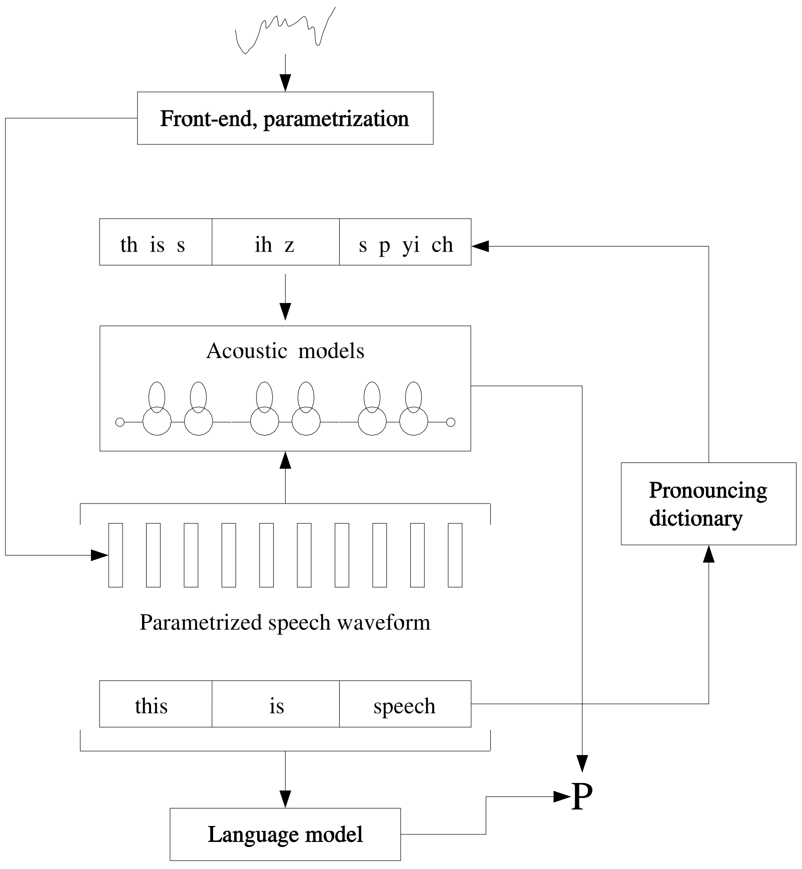

Figure 4.1: Speech recognizer schematics

In the figure 4.1 we see the schematics of the speech recognizer using the same principles as Sphinx II uses. More detailed description can be found in [6]. The processing is as follows:

1. The input waveform is preprocessed by the front end. Usually, signal is nor-

malized and then the feature vector is computed. In the speech recognition, cepstrum is often used, together with signal power and some other values.

Output of the front–end is the parametrized waveform, divided into frames. Sphinx uses frames $10ms$ long.

2. The feature vectors are passed to the recognition engine itself, in our case it is based on Hidden Markov Models or HMMs for short. The HMMs are trained to recognize triphones[1]. Triphones are also used to model the inter–word pauses, because as the result of coarticulation, the word boundaries are not known in continuous speech.

3. Time–synchronous Viterbi beam search in the forward direction. Time–synchronous means, that all searches are done frame after frame. It searches the full dictionary and language model to produce the first recognition hypothesis and word lattice. For each word recognized it produces the list of possible end times together with their probabilities.

4. Time–synchronous Viterbi beam search in the backward direction. This step is very fast, because it is limited only to the words recognized in the first step. It produces again a word lattice and list of possible beginning times for each word recognized.

5. An $A^*$ search using the word segmentation from the previous two passes and the scores from them together with the language model to produce an $N$-best list of hypotheses with their probability scores. This list is denoted $P$ in the figure 4.1.

[1]Not exact - in case of Sphinx, the triphones are clustered into equivalence classes, these are called in the CMU terminology "senones". HMM's are trained to recognize senones instead of triphones, because it reduces complexity and amount of the training data required. But for our purposes, thinking about triphones is precise enough.

6. List of multiple hypotheses may be post-processed with more complex language model, e.g. to restrict the possibilities to only limited task, when using a general purpose recognition system, or just the hypothesis with the largest probability score is selected.

The process outlined above is a description of a continuous speech recognition system using sub–word units to model speech. If the dictionary is limited to a small amount of words, it is possible to use a word–based model instead of triphone–based model. It may be done by replacing the HMMs recognizing triphones with HMMs recognizing words instead. This has an advantage in very simple training, because only small amount of data is needed to produce reasonable results. The rest of the process remains the same.

Actually, for command & control applications with limited vocabulary and simple commands, the word–based models are easier to deploy. Therefore, for the virtual humans, this approach was selected over the more general, sub–word based, which requires much more extensive training to get a good set of acoustic models[2].

## 4.3 Possible problems

The speech recognition system described in the previous section is supposed to be working perfectly and give results comparable to what humans are able achieve. Unfortunately, this is not the case in the real environments. There are many problems, which have to be taken into account to get a realistic expectation what a speech recognition system of this kind is able to achieve.

[2]The triphone–based models distributed with open source version of Sphinx II used cca. 10 hours of speech per speaker. To gain speaker independency, they used cca 300 different speakers. This amount of training data is very difficult to get unless there is a specialized laboratory at hand. Not mentioning the huge CPU power required to process this amount of data into acoustic models.

- **Speaker dependence**

  The ideal recognizer would speaker independent. This is impossible to achieve, because of different ways how people speak. For example, gender difference plays large role, thus system trained by male speakers may not be able to recognize female voices properly. Most of the speaker–independent systems require speaker adaptation or enrollment where the user trains the system to his own voice by reading texts presented by the system.

- **Acoustic conditions**

  Acoustic conditions play a large role as well. Ideally, the recognizer should be able to recognize human voice in a noisy environment as humans are able to do. The computer is not able of this because of various reasons, e.g. missing visual cues (subconscious lip reading by the listening person) or just because the environment is too noisy. For humans, there is the "party effect", where the brain focuses on the person speaking and we are able to pick up the conversation even in a room with many people talking at the same time. The computer is unable to do this, thus the useful information is "drowned" in the noise picked up by the microphone. This is the reason, why the most speech recognition packages require a good, noise–canceling close talking microphone.

- **Domain dependence**

  Domains with limited vocabulary give better results with speech recognition, than domains with large or unlimited vocabulary. The smaller the domain, the better the results, because there is smaller amount of possibilities to search and therefore lower error rate.

  On the other hand, this also mandates a good language model. If the model is not matching the domain we are trying to recognize, the error rates will be much higher. This is also a reason, why some speech recognition packages offer specialized modules e.g. for medicine or law. Some even offer a possibility to “feed” the system with own data e.g. in the form of documents, which are often dictated, to adapt the models to the real set of data, which is going to be used and thus improve the performance.

All this almost rules out a system as presented by HAL in the “2001: Space Odyssey”, at least with the current technology. But more modest systems with decent results (e.g. 95% accuracy) are possible, especially when the system is adapted to the speaker and a good microphone is used.

# Chapter 5

# Speech interpretation

The previous chapters described how a speech recognition system works. But it is not enough for the application, to know which text is the most probable match to what was spoken. The application needs to “understand” the content of the message as well, to be able to react accordingly.

## 5.1 Parsing

For the purpose of “understanding” of the spoken commands, as a part of this project, a grammar format and parser were developed. The grammar itself is loosely modeled after Microsoft’s Speech API XML format. An example of such grammar is in the appendix.

The idea is to check, whether the text incoming from the speech recognition system is matching some of the the rules defined by the grammar. If it is, then the corresponding rule name is sent to the application, where the application decides on executing some action in response.

Example of the rule definition :

```
<RULE ID="VID_Play_Sound" TOPLEVEL="ACTIVE">
    <P>play sound</P>
</RULE>
```

This simple snippet of code defines a rule matching the text “play sound”. If the text is matched, the application receives the “VID_Play_Sound” token. The `<P></P>` tags denote, that the text is **required**.

Another example :

```
<RULE ID="VID_Navigation" TOPLEVEL="ACTIVE">
    <O>Please</O>
    <L>
        <P>Switch to</P>
        <P>Select</P>
        <P>Show me</P>
    </L>
    <O>the</O>
    <RULEREF REFID="VID_Place" />
</RULE>
```

This is more complex, the `<O></O>` tags denote **optional** parts of the input (i.e. the rule still matches, even when the text is not present in the input). The tag `<L></L>` denotes a **list** of alternatives to be matched. Finally, the `<RULEREF REFID="VID_Place" />` tag references another rule, which is a continuation of this one.

The result of this parsing is a chain of rules, which matched the speech input or error, if nothing was matched.

## 5.2 Concept of the universal input channel

Together with colleagues Tolga Abaci and Mario Gutierrez, the notion of the universal input channel (or “remote control”) was devised. The main idea of the system is to enable the use of various input devices (e.g. PDA’s such as Compaq iPAQ, or joysticks, position sensors etc.) to control various actions inside of lab’s VHD++ framework.

Figure 5.1 shows the concept. Various devices are connected via “smart proxy”, which translates data from the format the device is able to process to universal message format. The interconnection between the smart proxy and VHD++ is usually

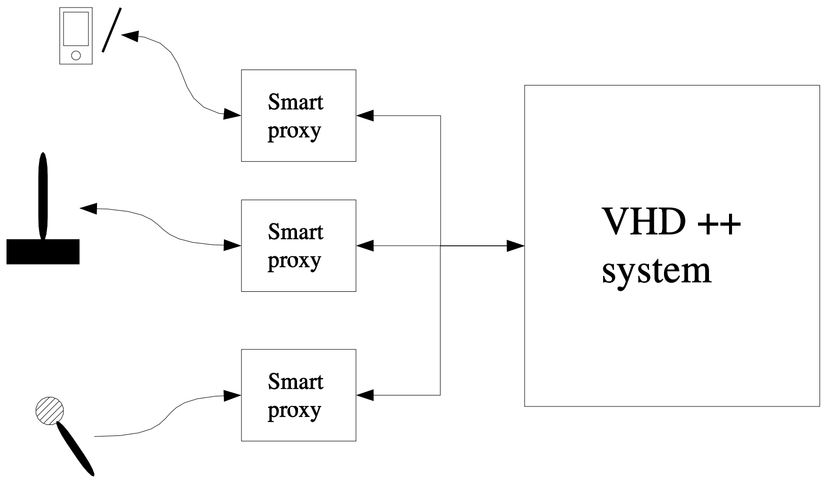

Figure 5.1: Universal input channel concept

over the network, where the proxy acts as a server providing the high–level interface to the device.

In the case of speech recognition, the recognizer is integrated with the parsing engine described in the previous section and connected to the network interface.

The client portion of the system, in our case the VHD++ framework, contains a service, which connects to the servers of devices required by the application, e.g. speech server. After initialization, the messages received from the server are processed and according to the mapping defined by the user and/or application, corresponding actions are executed. For example, the camera moves according to the position of the joystick or the virtual humans executes the order given by speech.

Two simple examples of the messages being passed :

```
@123456|000000|VID_NAVIGATION|*rule*|O|PLEASE|P|SHOW ME|
                            VID_PLACE|*rule*|P|CAMERA|P|ONE#

@123456|000000|SPEECH_RAW_TEXT|HELLO#
```

The first message shows a chain of two rules parsed out of the recognized text “please show me camera one”, the other one is just raw text as recognized by the speech recognition system, in our case “hello”.

## 5.3 Simple application

As a proof of concept, a simple application with one virtual human and speech recognition was built.

The virtual human recognizes commands “forward”, “backward”, “left”, “right”, “stop”, “sit down”, “stand up” and “hello”. Because the set of commands is so simple, it is feasible to use word–based models for training of the recognizer.

The grammar is simple as well, it defines just $1:1$ mapping between the words and tokens. The full listing of the grammar is in the appendix.

The virtual human directly reacts to the spoken orders. Order “forward” makes him walk in the direction he is currently facing. Orders “left” and “right” make him turn to the sides and order “backward” turns him around. The command “stop” stops current action being executed. The orders “sit down” and “stand up” make the virtual human play key framed animation of sitting down and getting up again. And finally, “hello” makes the virtual human bow.

Figure 5.2 shows the virtual human in action.

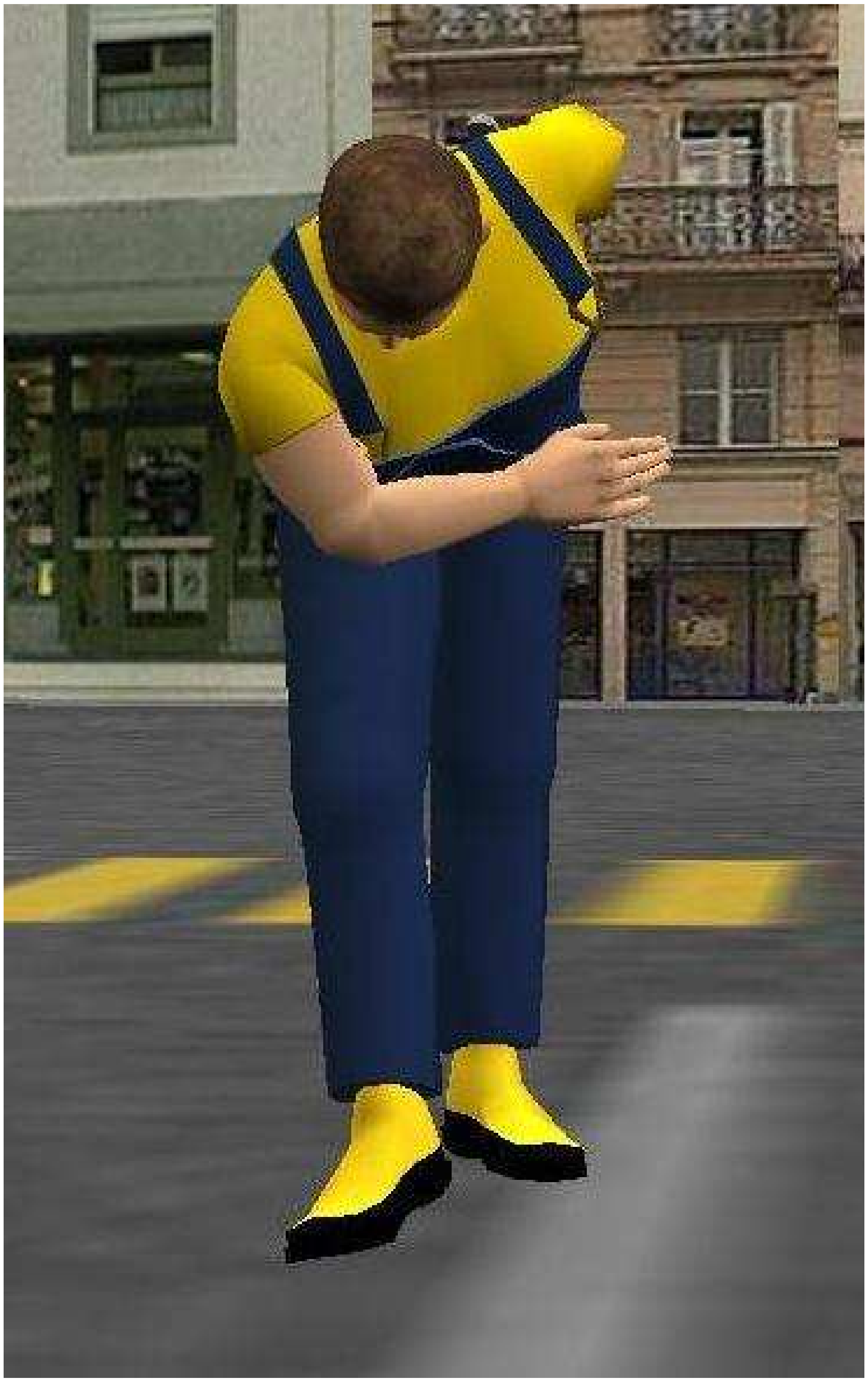

Figure 5.2: Application screen shot

# Chapter 6

# Conclusions

This project achieved the goal to give our virtual humans the limited capability of understanding speech.

Unfortunately, it also proved, that speech recognition *is hard*. To achieve good recognition accuracy required large amounts of training data and good language modeling, even for very simple tasks. Good microphone and low–noise conditions are a must.

Right now, the system is speaker and task dependent, thus requiring a new training for each user and each application. The future goal is to improve this by using the sub–word based models, which allow just adaptation for each speaker and the full retraining will be not necessary anymore. This mandates availability of the good base models, which finally became available from CMU recently.

Another experiment will be an attempt to train the word based acoustic models for recognition of orders in several languages, e.g. English, French and Turkish, for example. Such multi–lingual capability is possible, since the recognition engine itself is language neutral. This could have interesting applications e.g. for exhibitions, where visitors speaking various languages are common. Moreover, Switzerland with it's four official languages is a natural testing place for this kind of system.

On the application side, the future work will focus on more complex behavior models for virtual humans, so they may exhibit complex reactions to the inputs from the user and/or environment. The target is to be able to have the speech input as one of the input modes in the multi–modal environment.

Even when the speech recognition systems are infamous for their poor robustness, it was shown, that it is possible to build a usable system, if the requirements are reasonable and within the capabilities of the recognition system used.

# Chapter 7

# Appendix

Example grammar.

```
<GRAMMAR LANGID="409">
    <DEFINE>
        <ID NAME="VID_Viewpoint" VAL="3"/>
        <ID NAME="VID_Exit" VAL="4"/>
        <ID NAME="VID_Play_Sound" VAL="5"/>
        <ID NAME="VID_Stop_Camera" VAL="7"/>
        <ID NAME="VID_One" VAL="101"/>
        <ID NAME="VID_Two" VAL="102"/>
        <ID NAME="VID_Three" VAL="103"/>
        <ID NAME="VID_Four" VAL="104"/>
        <ID NAME="VID_Five" VAL="105"/>
        <ID NAME="VID_Place" VAL="200"/>
        <ID NAME="VID_Place_Index" VAL="201"/>
        <ID NAME="VID_Navigation" VAL="254"/>
        <ID NAME="VID_Command" VAL="255"/>
    </DEFINE>

    <RULE ID="VID_Exit" TOPLEVEL="ACTIVE">
        <L PROPID="VID_Exit">
            <P VAL="VID_Exit">exit application</P>
            <P VAL="VID_Exit">quit application</P>
        </L>
    </RULE>

    <RULE ID="VID_Play_Sound" TOPLEVEL="ACTIVE">
        <P>play sound</P>
    </RULE>

    <RULE ID="VID_Stop_Camera" TOPLEVEL="ACTIVE">
        <P>stop</P>
        <O>camera</O>
    </RULE>
    <RULE ID="VID_Command" TOPLEVEL="ACTIVE">
        <O>Agent</O>
        <L>
              <P>Go to</P>
              <P>Walk to</P>
        </L>
        <P>hotel</P>
    </RULE>

    <RULE ID="VID_Navigation" TOPLEVEL="ACTIVE">
        <O>Please</O>
        <L>
            <P>Switch to</P>
            <P>Select</P>
            <P>Show me</P>
        </L>
        <O>the</O>
        <RULEREF REFID="VID_Place" />
    </RULE>

    <RULE ID="VID_Place" >
        <L PROPID="VID_Place">
            <P VAL="VID_Viewpoint">viewpoint</P>
            <P VAL="VID_Viewpoint">camera</P>
        </L>
        <L PROPID="VID_Place_Index">
            <P VAL="VID_One">one</P>
            <P VAL="VID_Two">two</P>
            <P VAL="VID_Three">three</P>
            <P VAL="VID_Four">four</P>
            <P VAL="VID_Five">five</P>
        </L>
    </RULE>
</GRAMMAR>
```

Grammar used for the demo application.

```
<GRAMMAR LANGID="509">
    <DEFINE>
        <ID NAME="SPEECH_FORWARD" VAL="3"/>
        <ID NAME="SPEECH_BACKWARD" VAL="4"/>
        <ID NAME="SPEECH_LEFT" VAL="5"/>
        <ID NAME="SPEECH_RIGHT" VAL="6"/>
        <ID NAME="SPEECH_STOP" VAL="7"/>
        <ID NAME="SPEECH_SITDOWN" VAL="8"/>
        <ID NAME="SPEECH_STANDUP" VAL="9"/>
        <ID NAME="SPEECH_HELLO" VAL="10"/>

    </DEFINE>

    <RULE ID="SPEECH_FORWARD" TOPLEVEL="ACTIVE">
        <P>forward</P>
    </RULE>

    <RULE ID="SPEECH_BACKWARD" TOPLEVEL="ACTIVE">
        <P>backward</P>
    </RULE>

    <RULE ID="SPEECH_LEFT" TOPLEVEL="ACTIVE">
        <P>left</P>
    </RULE>

    <RULE ID="SPEECH_RIGHT" TOPLEVEL="ACTIVE">
        <P>right</P>
    </RULE>

    <RULE ID="SPEECH_STOP" TOPLEVEL="ACTIVE">
        <P>stop</P>
    </RULE>

    <RULE ID="SPEECH_SITDOWN" TOPLEVEL="ACTIVE">
        <P>sitdown</P>
    </RULE>

    <RULE ID="SPEECH_STANDUP" TOPLEVEL="ACTIVE">
        <P>standup</P>
    </RULE>

    <RULE ID="SPEECH_HELLO" TOPLEVEL="ACTIVE">
        <P>hello</P>
    </RULE>
</GRAMMAR>
```

# Bibliography

[1] Inc. A.L.I.C.E. Artificial Intelligence Foundation. Alicebot, the chat bot based on aiml. http://www.alicebot.org/.

[2] B.Christensen, J.Maurer, M.Nash, and E.Vanlandingham. Accessing the internet via the human voice. http://www.stanford.edu/˜ jmaurer/history.htm.

[3] CMU Speech group homepage. http://www.speech.cs.cmu.edu/.

[4] CMU Sphinx home page. http://www.speech.cs.cmu.edu/sphinx/index.html.

[5] P. Jacobs. Natural language processing: A brief history for skeptics. *Unisys World Print, January 2001 Issue*, 2001. http://www.unisysworld.com/monthly/2001/01/nlp.shtml.

[6] M. K. Ravishankar. *Efficient Algorithms for Speech Recognition*. PhD thesis, Carnegie Mellon University, 1996.