Responsive Behavior of a Listening Agent

R. M. Maatman,[1,2] Jonathan Gratch[2] and Stacy Marsella[3]

[1]University of Twente, Department of Computer Science

[2]University of Southern California, Institute for Creative Technologies

[3]University of Southern California, Information Sciences Institute

ICT Technical Report No:
ICT TR 02.2005

,
ICT Emotion Project: 53-0820-8640.

Author
R.M. Maatman r.m.maatman@student.utwente.nl

Supervisor
Jonathan Gratch gratch@ict.usc.edu

Institute for Creative Technologies
13274, Fiji Way
Marina Del Rey, California 90292-7008
United States of America

Mentor
Anton Nijholt anijholt@cs.utwente.nl

University of Twente
Postbus 217
7500 AE Enschede
The Netherlands

Acknowledgement
Stacy Marsella suggested the initial idea underlying this work – that some information in the speech signal could drive back channeling behaviors of a virtual character. Sue Duncan pointed us to several references on this literature and provided feedback on drafts. Roz Picard, Shri Narayanan and David Traum suggested several approaches for extracting information from the speech signal and. Raul Galt provided us some matlab code that helped inform our work. Regina Cabrera provided valuable editorial feedback. This work was sponsored by the U. S. Army Research, Development, and Engineering Command (RDECOM). The content does not necessarily reflect the position of the policy of the Government, and no official endorsement should be inferred

1 Preface

An essential part of the Computer Science education at the University of Twente in The Netherlands is a 14-week internship. The university encourages performing this internship abroad and the Institute for Creative Technologies of the University of Southern California offered this possibility.
Thanks to the effort of both Anton Nijholt and Jonathan Gratch, I got the opportunity to work on an assignment at this Institute and this proved quite an experience.
This report contains the results of the professional efforts I did during this internship, while working on my assignment called *Responsive Behavior of a Listening Agent*.

Because of the great time I had here, both professional and personal, I would like to thank Jonathan Gratch for making it all possible and for his guidance during the assignment. Also, I would like to thank Anton Nijholt for setting up this internship and offering the possibility to perform this internship abroad. Furthermore, I would like to thank Ed Fast and Patrick Kenny for explaining the architecture and the communication with the Mission Rehearsal System which contains the implementation of the agent.

Thanks to Stacey Marsella for providing additional insights on gestures and to Kumar Iyer for his assistance on the technical details of the motion tracker.

December 3, 2004
R.M. Maatman

2 Abstract

The purpose of this assignment is twofold. First the possibility of generating real time responsive behavior is evaluated in order to create a more human-like agent. Second, the effect of the behavior of the agent on the human interactor is evaluated.

The main motivation for the focus on responsive gestures is because much research has been done already on gestures that accompany the speaker, and nothing on gesture that accompany the listener, although responsiveness is a crucial part of a conversation.

The responsive behavior of a virtual agent consists of performing gestures during the time a human is speaking to the agent. To generate the correct gestures, first a literature research is carried out, from which is concluded that with the current of the current Natural Language Understanding technology, it is not possible to extract semantic features of the human speech in real time. Thus, other features have to be considered.

The result of the literature research is a basic mapping between real time obtainable features and their correct responsive behavior:

- *if* the speech contains a relatively long period of low pitch *then* perform a head nod.
- *if* the speech contains relatively high intensity *then* perform a head nod
- *if* the speech contains disfluency *then* perform a posture shift, gazing behavior or a frown
- *if* the human performs a posture shift *then* mirror this posture shift
- *if* the human performs a head shake *then* mirror this head shake
- *if* the human performs major gazing behavior *then* mimic this behavior

A design has been made to implement this mapping into the behavior of a virtual agent and this design has been implemented which results in two programs. One to mirror the physical features of the human and one to extract the speech features from the voice of the human.

The two programs are combined and the effect of the resulting behavior on the human interactor has been tested. The results of these tests are that the performing of responsive behavior has a positive effect on the natural behavior of a virtual agent and thus looks promising for future research. However, the gestures proposed by this mapping are not always context-independent. Thus, much refinement is still to be done and more functionality can be added to improve the responsive behavior.

The conclusion of this research is twofold. First the performing of responsive behaviors in real time is possible with the presented mapping and this results in a more natural behaving agent. Second, some responsive behavior is still dependant of semantic information. This leaves open the further enhancement of the presented mapping in order to increase the responsive behavior.

3 Introduction

This document is the result of a three-month internship performed at the Institute for Creative Technologies. The complete assignment will be discussed in chronological order and first, the assignment description is outlined. This is followed by the theory findings with respect to this assignment. These findings result in a mapping from real time obtainable features to responsive gestures which can be performed by an artificial agent.

This mapping then had to be implemented into the available virtual agent. To accomplish that, first a design has been made, which later on has been implemented. This design resulted in two separate programs, one for the voice analysis and one for the physical feature extraction.

After the implementation of this design it has been tested and the results of these tests are presented in the concerning chapter. The implementation has not been described in very much detail and the reasons herefore are twofold. First of all, the way of extracting the various features was not the main purpose of this assignment and second of all, because the text has to be readable, and the describtion of various for-loops and while-statements would not contribute to that.

With the implementation it was possible to perform the responsive behavior according to the mapping rules specified in the theory. However, to integrate this implementation into the target system, named Mission Rehearsal Excersice, an additional program had to be designed to control the communication between the two programs. With the implementation of this ‘controller’, the final implementation of this assignment was ready.

Finally, a conclusion is provided where the findings of this research are outlined and some future work is recommended.

4 Contents

5 Assignment Description

In a face-to-face conversation, not only the verbal aspect of the communication is of importance but also the nonverbal aspect. For example the nodding of the head to illustrate that one still follows the speaker or the pointing of the hand into a direction when giving directions to someone are essential in such conversations. These nonverbal aspects are also called gestures, and are both present by the person who is talking and the person who is listening. Nowadays, more and more computer generated actors (also called agents) take over tasks from humans and this result in conversations between a human and a computer agent. Thus far, the communication between a human and an agent is not experienced in the same way as a human to human conversation and this is partly because of the lack of gestures of the agent in response to the human.

The purpose of this assignment is to investigate whether it is possible to get an agent to perform certain gestures in response to the human who is talking, in order to get the human to perceive a more natural feeling about the conversation.

The focus of the assignment is to extract features of a speaker (verbal and non-verbal) and to generate responsive gestures in in real time. The goal is to create a component that can be integrated into a real-time embodied conversational agent. A challenge is that verbal information is typically unavailable in real-time, as current speech understanding techniques typically only provide semantic information after a complete sentence is known. Thus, our focus is on extracting acoustic and gestural features of the speaker that can be extracted in real-time.

The research is limited to the gestures that should be performed by the listener in response to the speaker during a conversation.

In short, the assignment description would be: ‘Investigate whether it is possible to extract certain features from a human speaker and get the listening agent to respond to these features with certain gestures, so the human perceives a natural behaving agent’.

Keywords in this research are: mirroring behavior, mimicking, social resonance, listener gestures, gestural response, embodied conversational agents, virtual humans

6 Gesture Theory

6.1 Introduction

A need to understand the way responsive gestures can shape the human speaker in his interaction with an Agent is a major impetus for this research. Whether or not these gestures are present during a dialogue can make a significant difference on how a human communicates with the Agent and on emotionally invested a human is in the conversation.

But first, it is necessary to understand the many different gestures that are performed during a natural conversation: gestures of the speaker that accompanies the meaning of his utterance, gestures of the listener that provide feedback to the speaker and gestures that that related to the relation between the speaker and the listener. The latter will be named mirroring gestures in the remainder of this paper.

6.2 Speaker Gestures

Most gesture research has focused on the gestures generated during speech (e.g. Bavelas, McNeill (2000), Zhao) and later we will discuss how these gestures have already been applied in virtual human models.

According to McNeill (1992), there are four different types of speaker gestures: Iconic, Metaphoric, Deictic and Beat. An Iconic gesture depicts a concrete object or event and bears a close formal relationship to the semantic content of speech. Metaphoric, on the other hand, are the same as Iconics but depict an abstract idea. Deitic gestures are pointing gestures to something or someone, either concrete or abstract. Beat gestures are two-phase movement (in-out, up-down) and accompany words that are considered important.

Although these gestures are not always necessary in order for the listener to understand the speaker, these gestures greatly improve the comprehension as often they are closely related to the semantics of the speaker. For this reason, when an Agent to perform these gestures, it is imperative that they are times in advance.

6.3 Listener Gestures

Listeners typically exhibit gestures, and these gestures can have a significant impact on the direction and flow of a conversation (McNeil, 1992; Bickmore and Cassell, 2004). This feedback often consists of simple actions. Although these kinds if gestures are often related to the semantics of the speaker, it is argued that there is also a relation between the manners of speaking. For example, when the speaker raises his voice, this could mean that he is making an important point and a response of the listener with a head nod would be appropriate.

Another type of gesture that is performed during conversations occurs at times of speaker disfluency. In order to keep the flow of the conversation going, the listener often provides certain gestures, often referred to as backchanneling, to signal the speaker that he is willing to wait for him to think about what he is about to say.

6.4 Mirroring Gestures

According to Lakin (2003), people often adapt their behavior to each other, especially during a conversation. Although they are not necessarily aware of doing it, people in a conversation will adjust the rhythm of speech, their body posture and even their breathing to each other. This adaptive behavior is called mimicking or mirroring behavior.

The most basic effect of this behavior is that it affects the emotional state of the persons involved (Baaren, 2000). This generally results in greater affection between the speaker and the listener and can result in the perception of a pleasant, natural conversation (Warner, 1986). It may also be important in synchronizing conversational flow, for example, by providing expectations on when a speaker can be interrupted.

The amount of mimicking is dependant on several aspects, like the subject of the conversation and the relation between the persons involved. For example, there will be little mimicking behavior if the subject of the discussion is formal and people in the conversation have never met before. When two friends are discussing weekend activities, a lot of mimicking will occur.

6.5 Background

Most preexisting research is on gestures that are performed when an agent is speaking, and speaking gestures have already been extensively studied within the contexts of artificial systems. Rea, for example, is an Embodied Conversational Agent that acts as a real estate agent (Bickmore and Cassell, 2004) and she frequently makes use of gestures when she speaks. When visualizing the size of an artifact, she places her hands opposite to each other, with the distance between her hands resembling the approximate distance of the object she is talking about. Because REA knows what she is about to say, these types of gestures can be timed accordingly.

However, when engaged in a conversation with a human, little work has been done on when an Agent should gesture as a response, particularly in the midst of a speakers utterance. The challenge of this paper is to improve conversations between a human and a computer Agent in such a way that the human perceives natural behavior from the Agent.

In the conversation between two humans, this is often accomplished by the performance of gestures that relate to the semantic information of the speaker. Thus, when this semantic information is known, the correct gestures can be determined and performed when appropriate.

6.6 Real time Semantics

In order for an Agent to successfully gesticulate in response to human speech, the Agent must first understand the human's semantics. This technology of analyzing and extracting the semantics of human speech is called Natural Language Understanding (NLU). The NLU performs syntactic analysis, semantic analysis and finally discourse integration.

To date, this technology requires a complete spoken utterance before it can start analyzing the semantic structure. And, although this approach gives a satisfactory outcome, the generation of an appropriate gesture would be delayed at least to the end of a utterance (and up to several seconds later in practice). This deferred response would heavily undermine the perceived authenticity of the Agent by the human.

The below exchange illustrates this.

A: *Where do I go?*
H: *You should take the first right but after that I don't know…*

Imagine that the Agent (A) responds to the human (H) with a head nod one second after the human has finished his sentence. At this point, the nod is performed too late, and this could be perceived as unnatural. However, if the nod would be performed after the pronunciation of 'right', then the nod would indicate that the Agent understood the human (although this might not be the case, but the responsive behavior is correct).

With current Natural Language Understanding technology, it is impossible to extract the semantics from the human speech fast enough to perform responsive gestures in time. Nevertheless, there are other features related to responsive gestures that could be used to create a more natural behaving Agent. The challenge of this research is to isolate these non-semantic features (and the appropriate responsive gestures that correspond to these features) in order to create a more natural interaction between a human and Virtual Agent.

6.7 Summary

There are many different kinds of gestures. But, as discussed, not all are suitable for implementation into an artificial Agent. However, there are certain gestures that can be performed in order to increase the perceived authenticity of the conversation. Gestures such as mimicking the rhythm of speech, body posture and gestures of the speaker can be extracted in real time and performed by the computer Agent. Also certain features of the speech signal of the human can be extracted which could result in feedback behavior.

6.8 Occurrences of Gestures

Now that it is clear which gestures exist and which responsive gestures a computer Agent in real time can perform, the actual occurrences of those gestures should be evaluated. So what triggers the performing of a gesture? To acquire these occurrences, we studied the available literature and we analyzed video data.

6.8.1 Literature

As already discussed, mirroring gestures are triggered by certain qualities of a parties movements. When this person performs a certain gesture or posture shift, the mirroring behavior can be triggered. Specifics of this will be discussed later.

The occurrence of gestures which respond to certain features of the speech is a bit more sophisticated. Suzuki (2001) argues that the uttering of hummed sounds (like '*uh huh*') is related to the changes in pitch of the speaker. In other words, when the speakers' voice contains a certain (large) shift in pitch, a hummed sound as a response to the speaker would be in place.

Similar research by Kapoor (2001) argues that a hummed sound by the listener in response to a lowering of pitch level of the speaker contribute to the flow of the conversation. This research was done with the use of telephone communication between the speaker and listener, but in face-to-face communications, the hummed sound is often accompanied by a head nod. This is emphasized by the results of the video analysis in the next section.

Another aspect during a conversation that could be enhanced is the gazing behavior of both the speaker and the listener. When there is some disfluency in the speech of the speaker, both participants change their gazing behavior and/or their facial expressions. The speaker is gazing away because he is looking for the right words and the listener often frowns or gazes intensively towards the speaker.

6.8.2 Videos

In order to find more occurrences of gestures, several videos have been studied which contained a role-playing game between two persons. The purpose of this role-playing game was to gain insights in how a conversation between two people with different end objectives could develop. These insights were then used to model an Agent accordingly.

During this role-play, one of the participants plays a military sergeant who is trying to convince the other participant, a medical doctor, to evacuate his hospital because hostile activities might take place in the area.

The main tension within the role-play involved the immediate treatment of the patients versus the possibility of increased danger by doing so.

Although four different sets of actors were recorded, only one set proved useful to study the responsive behaviors. Even though all actors made use of the speaker gestures presented earlier, only one doctor used feedback gestures when responding to the sergeant. The gestures performed by the doctor at these moments consisted of head nods when: he understands the sergeant, when the sergeant raises his voice and the sergeant emphasizes a certain point. The first aspect is related to the semantic information of the sergeant and, as previously discussed, the latter aspect was already pointed out by Kapoor (2001).

The video also confirmed the gazing behavior as it was pointed out in the previous section. But the second aspect is new. This shows that when a voice is raised, the person responds with performing a nod.

Another seemingly remarkable result of the analysis of the video, which is outside the scope of this research, can be found in appendix A.

6.9 Real time possibilities

Now the various gestures are outlined, the real time obtainable features should be pointed out. As mentioned earlier, meaningful semantic information is difficult to obtain in real time (within an utterance), thus other features have to be examined. These features are separated into features related to the speech signal of the human and physical features.

6.9.1 Speech Signal

From the speech signal, we can extract certain features that are not directly related to the semantics of the speech. These features could then be calculated instantly from the input from a microphone. Only the basic features are considered here, because although the computational speed of the current computers is rapidly increasing, it should be kept relatively simple.

According to Milewski (1996), it is possible to extract frequency and intensity information from a speech signal in real time using a Fourier transformation. When these two aspects of the speech signal are known, many useful derivatives can be calculated, such as silences, monotone sounds, et cetera. Thus when using this transformation, there can be much information available in real time concerning the features of the speech signal.

6.9.2 Physical Features

There are two ways to determine physical features from a human in real time: using a camera and image analysis or using 3d-trackers. Image analysis consists of recording the human with a camera and analyzing the data from the camera. The advantage of this method is that the complete human is visible and thus in theory all information could be extracted (with two or more cameras it might even be possible to get a 3D-image from the human). The disadvantages however are that it is computationally intensive and it is much work to create such a system from scratch.

In contrast to the image analysis method, there are tracker devices. These trackers can keep track of their point in space and their orientation towards a certain point. The advantage of using trackers is that they are fast and are not as computationally intensive as image analysis. The major drawback is that the trackers need to be set up and are only operational in a limited area. In addition to this, when using a tracker, only the point of the tracker is known and no other parts of the human body.

For this research however, a space was available where a tracker device was already operational and thus this device has been used to extract the physical features of the human. With this tracker it was possible to extract both head gestures (such as gazes, nodding and shaking) and posture shifts.

6.10 Gesture Mapping

Because the focus of this research is the affect of listener gestures on the perceived authenticity of the conversation, a relatively simple mapping between the extracted features and the gestures would be sufficient. As a result from the previous theory, the following mapping can be made. The mapping is separated for every gesture and is explained briefly.

6.10.1 Head Nodding

According to Ward (1999), there is a relation between the pitch and the possible occurrence of a humming sound. As we depicted earlier, a humming sound and a nod bear the same meaning and so we can use Ward's algorithm to perform a nod and thus the first mapping rule:

1) *Lowering of pitch for more then 110 milliseconds → head nod or humming sound*

The complete algorithm of Ward can be found in appendix B

According to Cassell (2000) and to the video analysis, a head nod could also be a result from the raised voice of the speaker. The accompanying speech signal feature of a raised voice would be a higher intensity or loudness, and thus the second mapping rule would be:

2) *Raised loudness (intensity) of speaker → head nod*

Cassell (2001) argues that the result of higher intensity could also result in other types of gestures like eye gazes, beat and hand gestures, so there is a little room for extension or adjustment here.

6.10.2 Head Shaking

According to Kapoor (2001), a shake of the head can stand for negation or disapproval and thus related to the semantics of the speech. But the shaking of the head also relates to desired mimicking behavior. So if the speaker shakes his head, the listener should shake also, but in a much more discrete fashion. Because the detection of a head shake would be possible with the use of a tracker, this leads to the following mapping rule:

3) Head shake of speaker → Shake head

6.10.3 Posture Shifts

The rules concerning posture shifts are separated into two different parts: one that is related to the desire to mimic the behavior, and the other related to certain speech features which could lead to a posture shift.

6.10.3.1 Mimicking Related

The posture shifts are mainly performed due to the desire to mimic the behavior in order to influence the emotional state of the human (Baaren, 2000). Thus, when the human performs a posture shift, the Agent should detect it and perform also a posture shift. This might mean the change of weight distribution from one leg to another, the folding of the arms and certain characteristic postures. These kinds of posture shifts can be detected when the human has a tracker based on top of his head or in his hand.
Because just the mapping rule is needed, no more detail is specified here but the mapping rule:

4) *Detection of posture shift → Posture shift*

6.10.3.2 Speech Feature Related

According to the video analysis, the listener performs posture shifts when the speaker has certain disfluency (stuttering) in his speech. While awaiting the correct words from the speaker, the listener performs a posture shift (like the change of weight distribution from one leg to the other) or changes his gaze. In addition to these behaviors, the listener also seems to perform some frowns although this was hard to see in the videos.

The meaning of such a posture shift would be that the listener is telling the speaker to take his time (Cassell, 2000). Because a disfluency could be detected in the speech signal with the relation between frequency and intensity (a disfluency could mean a longer period of monotone sounds like uhhhh, or the occurrence of silences), this would lead to the following mapping rule:

5) *Disfluency in Speech Signal → Posture Shift / Gaze shift / Frown*

6.10.4 Gazing Away

When the speaker is mentioning a concrete object within his vicinity, he will often look at it. When this lasts for a certain amount time, the listener could mimic this by either looking in the same direction or looking around. Which one to be done depends on the semantics, but gazing into the same direction would often be sufficient (Bickmore, 200?). Because this gazing of the human can be detected by using the trackers, this leads to the following rule:

6) *Speaker gazes away for longer period → Gaze away*

6.11 Other External Influences

Because we are dealing with humans, there are always external factors that influence the occurrence of gestures during a conversation. Of course, not every person will gesture as much as the other. There are people who almost do not gesticulate at all and there are people who gesticulate like it is a workout. Often, this is related to the speaker's emotions during the conversation. When someone is very excited they need to get rid of a surplus of energy, and that will result in more gesturing than would occur when that person is extremely sad.

Also, the relation of the two people is of importance. People tend to gesture remarkably more when they talk to a friend then when they are talking to a complete stranger (Welji & Duncan 2004). However, when that stranger is asking for directions, there will be more gesturing again.

Thus, the mapping presented here is not a complete coverage of all gestures that at all times are accompanied by the certain speech features, but should be sufficient to increase the perceived authenticity of the conversation.

6.12 Conclusion

During a natural conversation between two people, there are many gestures that occur from both the speaker and the listener. The difference between these two is that the gestures of the speaker are often related to the semantics of the speech, and the gestures of the listener are also related to other features. In other words, not only the meaning of the words is of importance, but also other aspects of the conversation like the intonation and the loudness of the voice and physical properties of the body.

With the current Natural Language Understanding technology it is not possible to extract semantic features from a speech signal in real time and thus other features have to be used to perform responsive gestures.

This research presents a mapping between these real time obtainable features and their accompanying responsive gestures. The mapping is however not limited to these gestures, because many other factors that influence the occurrence of gestures exist, like emotional state and relation of the persons involved.

When implemented into an Embodied Agent, this mapping results in more natural behavior by the Agent in response to the human and thus this mapping is a ground for future research on responsive behavior of an Agent in real time.

7 Prerequisites

Before anything can be done, there needs to be a clear picture of the availability of certain aspects which could be used for this assignment. In the end, the goal is to integrate the results of this assignment into the overall system, which runs in the Virtual Reality Theatre of the ICT. Thus a little background on this theatre and the availability of various useful aspects is provided in this chapter.

7.1 The VR-Theatre

The Virtual Reality Theatre is a room which consists of several aspects to create a virtual reality experience. These aspects are a half round wide screen of 10 by 30 feet approximately, complete surround sound and a possibility to put on a headset with camera projection for both eyes to actually look around in 3D.
This theatre is the home of the Mission Rehearsal Exercise project. This project is funded by the United States Department of the Army and consists of real life scenarios containing multiple virtual humans in order to create an immersive learning environment (Rickel et al., 2003). One of the goals of this project is to create realistic behaving virtual humans and the results of this assignment could be a contribution to that.

7.2 Tracker

As mentioned before, there is a tracker device available in the VR-theatre which consists of a headset and a handheld device. This Intersense IS-900 tracker (Intersense 2004) tracks the 3D location and orientation of both of the devices within the VR-theatre and thus would be usable for this assignment.

7.3 Audio

To obtain the audio from the user of the system, a wireless microphone is used to send the speech signal to a receiver. The default output from this receiver is linked to a Natural Language Understanding module that runs on a Linux system, to process the speech signal. Because the program written for this assignment is based on Windows operating systems, the output signal had to be split so it could be send to an additional Windows machine. To extract features from an audio signal, both standard C-code and Matlab have been used and a comparison of the usability has been made.

7.4 Conclusion

With the described functionality of the Virtual Reality Theatre, all the input needed for this assignment is available: the tracker to obtain physical gestures and the microphone input to filter the audio signal. Because of the availability of both a Windows and a UNIX based machine in the theatre, the input can be processed any favorable way so the Virtual Reality Theatre would be the ideal place to run the implementation of this research.

8 Design

In the previous sections, the theory about the listening gestures is clarified, a basic mapping between real time obtainable features and gestures is provided and the availability of two input devices is described. Thus, all necessary material to implement the listening gestures is available.
Before the design of the complete system can be presented however, a division has to be made between the different aspects of the responsive gestures. The mapping presented in the gesture theory chapter proposes two kinds of responses to gestures, the responses to the gestures of the human (obtained with the tracker device) and the responses to the features from the speech signal from the human (obtained via the microphone). This division will persist in the remainder of this document. This means, that first the design of the tracker gesture program will be presented and after that the design of the audio feature program. In the next chapter, the implementation of both of these designs will be explained in the same division as is made here.

8.1 Tracker Features

According to the theory, there are two basic gestures that can be obtained from the tracker and these are head gestures and posture shifts. First, head gestures are clarified and the design to extract these is presented. After that a definition of a posture shift is given, as is the design to detect posture shifts.

8.1.1 Head Gestures

8.1.1.1 Nodding and Shaking

To detect nodding and shaking using a tracker, the orientation of the tracker is of foremost importance. When the orientation of the head makes the same sort of angle repeatedly (alternating between positive and negative) this means either a nod or a shake. It would be a shake it the movement is along the x-axis (horizontal) and it would be a nod if the movement would be around the y-axis (vertical) as can be seen in figure 1.

So with the orientation of the tracker along these two axes, the nodding and shaking gestures of the head can be recognized using the tracker and the according mimicking rules can be applied to the agent.

8.1.1.2 Gazing

If the human would be gazing towards a certain position, this would mean that his head makes an angle in either the x or the y axis (or both). When this change in orientation holds for a longer period of time, it must mean that the human is staring in a certain direction. Thus this can also be detected using the tracker.

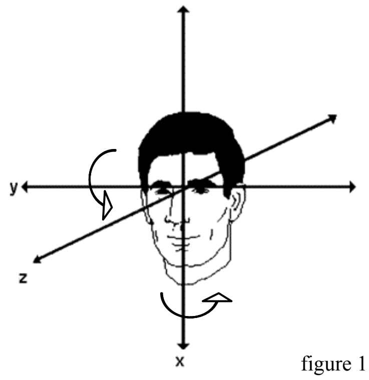
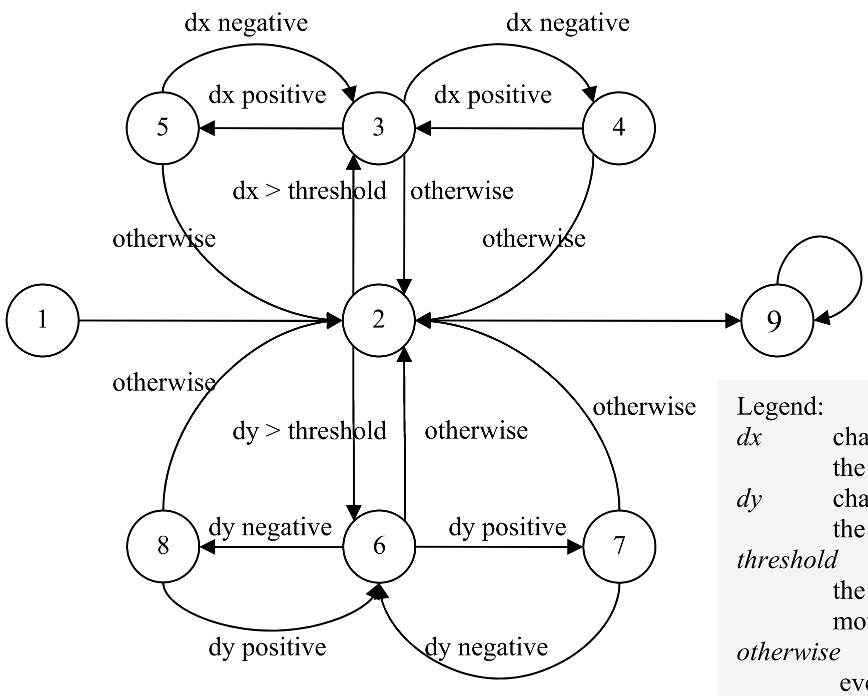


figure 1

8.1.1.3 Head Gesture Detection

We implemented a finite state machine to recognize the head gestures presented above. The machine receives as input the different types of motion that are obtained from the tracker. These types of motion are the change of orientation of both the x and the y-axis of the tracker. The machine looks like this:



Legend:

dx	change of movement along the x-axis
dy	change of movement along the y-axis
threshold	the minimal change of motion
otherwise	every other type of motion

Explanation of the states and the transitions:

1	Initial state. Here the initial position and orientation of the tracker are stored to calculate the relative changes in position and orientation.
2	Basic state. All transitions end up here when there is no matching transition to another state. If there were nods or shakes detected, they will be executed here.
3	X-Movement. There is sufficient movement on the x-axis to qualify for a nod. If the previous state was either state 4 or state 5, then a nod is detected.
4	X-Positive. The head turns towards a positive angle on the x-axis (the head looks up)
5	X-Negative. The head turns towards a negative angle on the x-axis (the head looks down)
6	Y-Movement. There is sufficient movement on the y-axis to qualify for a shake. If the previous state was either state 7 or state 8, then a shake is detected.
7	Y-Positive. The head turns towards a positive angle on the y-axis (the head looks towards the right)
8	Y-Negative. The head turns towards a negative angle on the y-axis (the head look towards the left)
9	If there is no sufficient movement along either of the axis, a timer is started. When there is movement, this timer will be reset, but when the timer reaches a certain minimal value (like three seconds), this will mean a gaze in the current direction. If the angle is greater then a certain threshold, it will result in a gaze (otherwise it could be a

gaze toward the agent, and this is normal). This state can be reached from every other state (because a gaze could start at any moment in time).

Remark:
When the system returns to state number 2 and there were one or more shakes or nods detected, the system will execute these shakes or nods. The effect of doing it in this fashion is that only complete nods and shakes are executed. However, they only will be executed when the human has stopped executing them himself so there is a little delay.

With this state diagram it is possible to extract the different gestures from the human head and thus let the agent perform some mimicking according to the mapping rules that have been specified in the gesture theory chapter. The relevant rules for the head gestures are numbers 3 and 6.

8.1.2 Posture Shifts

The detection of head gestures was not the gesture detected by the tracker. Posture shifts could also be obtained by using the tracker information. In order to do this, first a clear definition of a posture shift had to be stated:

A posture shift is the translation of the center of mass from one leg to the other

In other words, the change from leaning on one leg to leaning on the other (or on both). A simple example proved that this posture shift is accompanied by the translation of the head with respect to a static position between the feet, only when the feet do not move. This way, a posture shift could be detected with just one tracker placed on top of the head of the human with the only requisite, that the human does not move his feet.

To clearly picture the way a posture would be detected, see figure 2.

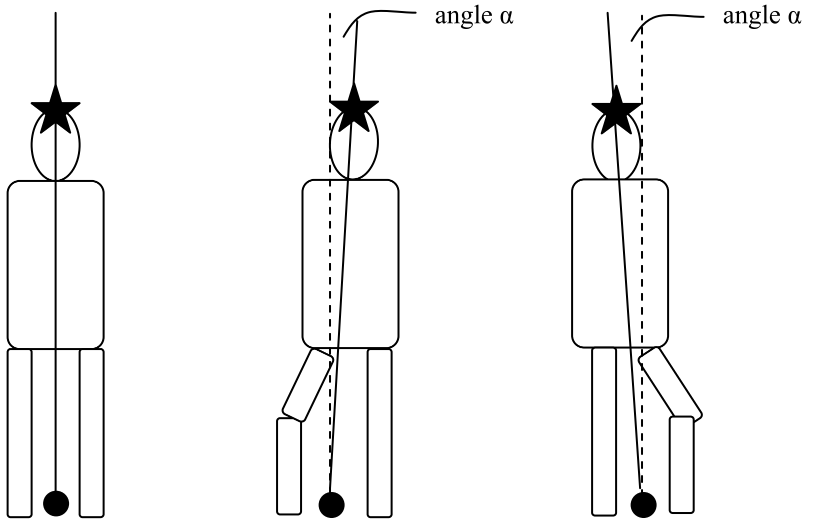


Figure 2

In this figure, it is clear that when the human slouches, an angle α could be calculated between the origin (dotted line) and the position of the tracker, placed on the head of the human. If this angle is greater than a certain threshold, this must mean the human is slouching.

Now it is possible to detect whether the human is slouching, it is also possible to detect if he is changing his posture from one slouch to another (including the posture of standing up straight). To do this, the only necessary elements are the current and the previous type of slouch. If these differ, that must mean a posture shift.

Now it would be possible to detect posture shifts, gesture mapping rule number 4 can be applied.

8.2 Audio Feature Extraction

Besides the different gestures from the trackers, there are also the speech features that have to be extracted in order to implement the remaining mapping rules. According to the theory about the listener gestures, we need to filter the speech signal from the user to extract different features like frequency and loudness. Because the author had no prior signal processing experience, the first thing to do was to read the theory on the internet. Thanks to different sites the basics were learned fairly quickly and thus the obtaining of the sound from the microphone became the first issue. After it was possible to get the signal from the microphone, some filters and algorithms could be applied in order to get to the features needed for the listener gestures. Then the mapping could be implemented and the gestures for the agent could be performed.

8.2.1 C/C++ and Matlab

We developed an initial audio extraction algorithm using C-code, without any additional libraries or tools. During the evaluation of some early versions of the program however, the question rose whether it might be better to use Matlab in order to calculate the features. First of all because this is a mathematical studio which already has many build-in functions and could be accessed in real time, and second because the availability of some existing Matlab scripts which calculate loudness features that could be useful. Before eventually was decided to buy Matlab, the audio features were still extracted using C-code. In the end, an evaluation could be made which compares the use of both and give recommendations about which to use.

8.2.2 Audio Input

Before one can extract features from an audio signal, the signal must first be recorded. This is supported by built-in Windows functions. This functionality is thus to be used to record the human speech. To conveniently test the feature extraction without having to speak into the microphone all the time, it would be useful to use (prerecorded) wave-files that are already on the computer. In order to use these, they need to be read. Finally, is has to be possible to play these wave-files so the agent can respond to the speech in real time during testing.

This described functionality is thus needed before any feature extraction could be performed. The most convenient way to implement such functionality would be to create some kind of wrapper around the build-in Windows functions. This would mean a class which calls the default functions in the right fashion. This class should have the following functions:

- Open a wave-file from the hard disk
- Record a wave-file from the default input (microphone)
- Play a wave file

An instance from this class could then be used to conveniently work with wave-files.

8.2.3 Feature Extraction

According to the mapping rules provided earlier, there is the need to extract features like the pitch and the loudness from the speech signal. With these features, the remaining mapping rules number 1, 2 and 5 can be implemented.

With the ability to record and open wave files, the only aspect that remains is performing some calculations on the audio data in order to extract the features like pitch and loudness. During a literature research on this topic, the Fast Fourier Transform (FFT) was recommended to extract these features from the audio signal, also because it would be fast enough to do it in real time. This is thus the first calculation that has to be implemented.

8.2.3.1 Fast Fourier Transform

The Fast Fourier Transform separates a complex signal into several simpler signals. The exact computation of the FFT is not described here, but can be found in various sources of literature (e.g. Milewski, 1996). With the FFT, an audio signal is divided into an arbitrary number of frequency-intensity pairs and these pairs can be used in the computation of several features of the speech signal. This means, that when the continuous speech signal is sampled in parts of a certain length, from each sample the FFT could be computed and compared. This way, the intensity-frequency pair of each sample is known and this information can be used to implement the algorithm of Arons (1994), which uses the frequency of the speech signal.

8.2.3.2 Intensity Detection

With some minor adjustments, the algorithm by Arons can be used to perform the intensity detection. After determining the average (normal) intensity of a speaker during an initialization phase, the real time intensity can be compared to a pre-computed threshold. This threshold would be the top one percent value of the initial intensity value computed during initialization. When the real time intensity value exceeds the threshold it must mean that there is a raise in intensity. With this information, mapping rule number 2 can be implemented.

Besides this approach to the computation of the intensity of a speech signal, another approach is proposed by Fernandez (2004). His model uses the computation of certain loudness features, which could improve the previously described method. The code to perform these loudness computations in Matlab has been available to us, but unfortunately this proved too slow to be useful for real time computations. Due to the limited amount of time available for this research, no optimizations could be made and thus the loudness detection model was discarded.

8.2.3.3 Pitch Detection

For the C-code implementation, the pitch detection can be done in a somewhat similar fashion as the intensity detection. For each sample, from the resulting frequency/intensity pairs of the FFT, the frequency is chosen that has the highest intensity and this is compared to the previous highest frequencies. This way, a significant drop or raise in the frequency of the speech can be detected.
For the implementation in Matlab, a pitch detection algorithm is available from the Matlab User Community (2004). This algorithm is free and ready to use.

With the ability to detect the pitch or fundamental frequency, it should be possible to implement the backchannel-algorithm by Ward (1999), which can be found in appendix B.

Once the pitch of the speech signal can be detected, mapping rule number 1 can be implemented.

8.2.3.4 Disfluency Detection

The final mapping rule depends on the detection of stuttering of disfluency in the speech signal. An example of this would be the expression 'uhhhh' for a longer period of time. To detect this, the frequency of the signal could be used. If a certain frequency holds for a longer period of time and does not vary much, this could mean disfluency.

This method of detecting disfluency was proposed by Shriberg (Shriberg), and concerns different types of disfluency, like filled/unfilled pauses, false starts and repetition of words. For this research, only the filled and unfilled pauses shall be considered, because these types do not depend on semantic information and thus can be detected using frequency and intensity respectively.

With this information, every mapping rule is covered and thus the implementation of the detection methods can begin.

8.2.4 Conclusion

In this chapter, a brief overview has been presented in which is explained in what way the various aspects needed to implement the mapping rules can be computed. For the tracker related features, it is clear that the position and the orientation of the tracker are needed, and for the audio features there is need to initialize the audio signal of a certain human first. After the initialization phase, the audio signal can be sampled in real time and each sample can be used to compute the various required features.

9 Implementation

Now the design of the different parts of the assignment has been completed, the implementation can begin. Again, the designs are divided into separate parts and the implementation of each of these parts is explained in the following.

9.1 Tracker Features

The design of detecting human gestures from the tracker data can be separated into two aspects and these are the detection of the head movement of the human (the nodding, shaking and gazing gestures) and the detection of posture shifts (like slouching). For both of these aspects a function has been created which processes the data from the tracker in order to extract the gestures from the human, and to handle the sending of the according output to the agent. The implementation of these functions is explained below.

9.1.1 DetectHeadMovement

This function implements the state machine presented earlier in order to detect the nodding and shaking of the human head as well as the gazing of the human into a certain direction. The implementation consisted of four basic steps which will each be described below.

9.1.1.1 Tracker

The first step that had to be taken was the correct data collection from the trackers. Thanks to some sample code from Muller (2004), the connection between the code and the tracker was established fairly quickly. Because of the sample code, it was certain that the correct data was retrieved from the trackers.

9.1.1.2 State Diagram

To correctly implement a state diagram, it is of great importance to test it thoroughly. Unfortunately, the VR theatre where the tracker would operate would not be available much, so another way of testing the state diagram should be examined. Because the data needed to test the diagram is purely orientation based, the testing of the diagram could be done using the mouse coordinates from the computer. The actual data would not differ much from the data that would be retrieved from the tracker.

Thus, a program has been written that receives the coordinates from the mouse inside a console window and sends these coordinates to the state diagram implementation. This way, it is fairly easy to do simulate some tests and thus the correct implementation of the diagram.

9.1.1.3 Communication with the agent

Now it is certain that the state diagram and the tracker both work properly, the only thing that remained was the correct response towards the agent. Commanding the agent has to be done via the communication bus of the system, called Elvin. Because there was little documentation about this communication and about the different commands, there was need for a program to test the communication in order to find the exact usage and whether the communication actually worked.

For this purpose, a small program has been written that asks the user which command to send to Elvin. This way, the communication could be tested. A minor drawback here was that the complete MRE-system would have to be installed on the computer, and that took a while.

9.1.1.4 Bringing it all together

When the separate parts are all finished, the only thing that remained was to combine them and thus creating the actual function. Combining the parts was not too hard, but to get it to compile under UNIX proved more difficult. This was mainly due to my lack of knowledge of linking c-programs under UNIX, but with help from Ed Fast it all compiled and ran eventually.

9.1.2 DetectSlouch

In order to detect a posture shift with just one tracker, we require that the human does not move his feet. To facilitate this, a piece of wood was fashioned depressions indicating where the person is to stand. This way, the posture shifts can be detected with just the angle between the head and the position between the legs. This position can easily be obtained during the configuration of the tracker. Besides this position we also need the height of the tracker (the length of the human) in order to be able to compute the angle.

If both of these variables are known, the angle α can be computed as follows:

$$\alpha = \text{atan}(\, dx / \text{height\_of\_tracker} \,)$$

Where dx is the relative position of the tracker with respect to the initial position where the human is standing straight and *atan* is the inverse tangent function. The resulting angle of the *atan* function is specified in radians, but for more user friendliness, the angle would be better specified in degrees, so the conversion between these two has to be made:

$$\alpha_{\text{degrees}} = (\alpha_{\text{radians}} * 180) / \pi$$

Now the angle is known, the only calculation that has to be made is to compare the current angle to the threshold in order to get the type of slouch (left, right or neutral). Depending on an optional minimal slouch time (the minimal duration of the slouch by the human), the agent can instantly perform a gesture or when the minimal time is exceeded.

To summarize the detection of posture shifts, the first thing to do is to calibrate the tracker and get some initial values. Then every program cycle, the angle is computed, the type of slouch is detected, and depending of an optional delay and the previous slouch type, the agent could perform the slouch.

9.2 Mimicking Program

In the previous section the individual functions are explained that result in the detection of gestures from the tracker. Now these functions have to be called from a main program loop, which also retrieves new data from the tracker. In order to implement this loop and keep the program structure clear, certain classes have been defined for the different aspects of the program. Each of these classes will be explained and the class diagram will be presented to illustrate the structure of the program.

9.2.1 Classes

9.2.1.1 Tracker

The first class that is needed is the Tracker class. This takes care of the communication to the tracker and updates the local values retrieved from the tracker. When new values are to be read, first the Update-function has to be called. This function obtains the new data from the tracker. Then with the GetPosition and the GetOrientation functions, the different kinds of data can be retrieved. The function GetTimeStamp retrieves the timestamp from the tracker at the moment of the last update.

9.2.1.2 Elvin

For the communication towards the agent via the Elvin communications bus, this class has been defined. It opens the communication device with the Open function and registers the necessary operands that could be needed for retrieving data from the Elvin bus. With the Send function a command string can be send to the Elvin bus with the goal of letting the agent perform that command.

9.2.1.3 Gestures

This is the class which contains the different gesture extraction functions like DetectHeadMovement, DetectSlouch (both described above) and some additional functions needed to the correct implementation of these previous ones. The SetElvin function sets the local pointer to the Elvin variable in order to be able to send data to Elvin. The private DetermineState and ResetValues functions are both needed for the implementation of the state diagram which detects the head movement of the user.

9.2.1.4 Mimicking

This is the main program which contains the continuous loop. After initializing instances of the above classes, it calibrates the tracker. This means, retrieving data from the tracker once and store these initial values in order to determine the relative movement of the tracker. After this, the main loop is entered which consists of updating the tracker information, retrieving the relevant values, determining the relative movement and call the two Gesture functions DetectHeadMovement and DetectSlouch.

9.2.2 Class Diagram

The previously described classes can be visualized in some kind of class diagram. This diagram (figure 3) contains the different classes as well as the functions of these classes. Every arrow between the classes represents the information flow between these classes.

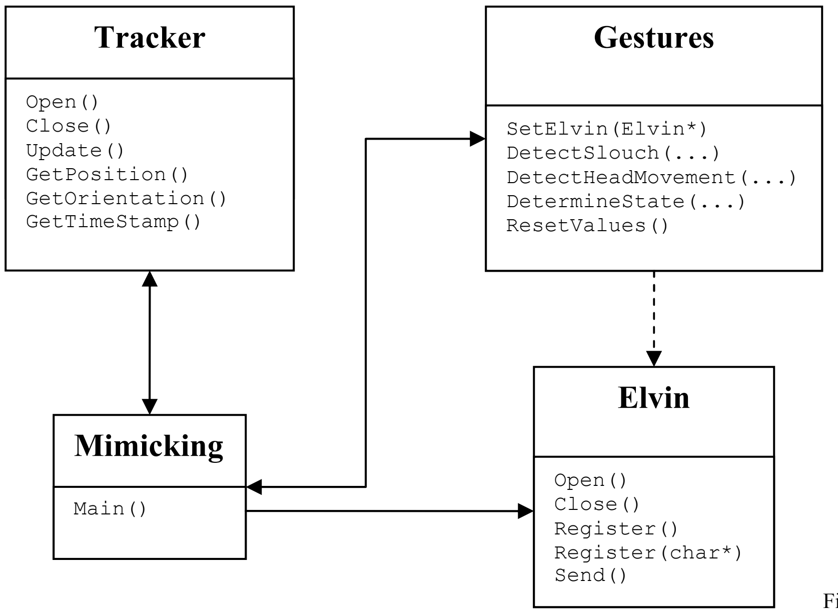


Figure 3

9.2.3 Program Flow

To visualize the complete Mimicking program, it can be explained by using a diagram as can be seen in figure 4. This diagram represents the various data flows and the functions that are called. The dotted arrows represent the call or feedback of a function, the normal/straight arrows represent the necessary data the different methods need to execute.

As can be seen in the diagram, first the Elvin and the Tracker communications are opened. Then the tracker is initialized and calibrated and then the main loop is entered which consists of detecting the slouch and the head movements. Then the response for the agent is sent as an Elvin message and the loop is finished and starts again.

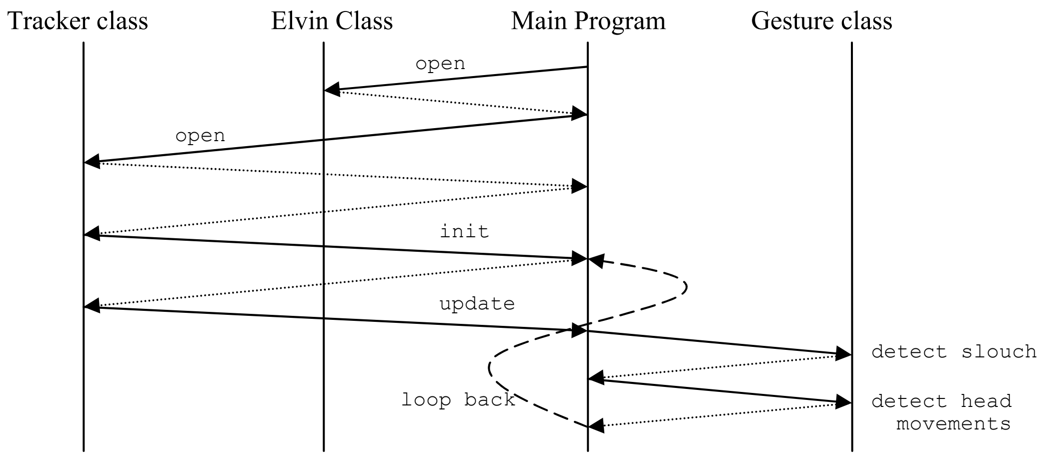


Figure 4

9.2.4 Conclusion

The implementation of the Mimicking program has been described in the previous section. It is a combination of different parts that take care of the various aspects the program consists of, and the main loop calls all these functions in such fashion that the gestures are extracted from the tracker and that the corresponding gesture from the agent is send to the agent via the Elvin communications bus.

9.3 Audio Features

As pointed out in the various previous sections, it is needed to detect certain elemental features from the speech signal in order to perform backchannel gestures. The features mentioned are the pitch/fundamental frequency of a part of speech, the intensity/loudness and the (dis)fluency of the signal. For each of these features, the implemented algorithm to detect them shall be explained, as well as the direct relation between the features and the gesture mapping approach mentioned earlier. This will result in a program that will analyze the audio signal from the user, and will perform the listener gestures according to the gesture mapping. The description of the implementation will not contain very detailed information about programming specific issues (for example array computations and memory allocation) in order to maintain the readability of the text. The main approach to the detection of the several features should be clear.

9.3.1 Fast Fourier Transform

As pointed out in the Theory and Design chapter, the Fast Fourier Transform can be used to extract the frequency/intensity pairs from an audio signal. The implementation of the FFT in Matlab is a standard function of the Signal Processing Toolbox but in normal C-code, the FFT has to be implemented. Fortunately, there has been an implementation available from Milewski (1996) which can be used for this project. A few minor adjustments have been made to get it to work with the data type that is used for the recording of the audio, but these are not worth mentioning.
The results of the FFT are, as said before frequency/intensity pairs, so these can be used to detect the frequency with the highest intensity for each sample. The change in this frequency is the basis for the detection of pitch changes, because this is a simple and fast way to approximate the results.
Because of the fact that human speech basically varies between a pitch of 500 and 8000, the frequencies outside this range are discarded.

9.3.2 Frequency/Pitch Variation

The use of the FFT to measure the changes in frequency is only used in the C-code implementation of the extraction function. In Matlab, another approach is chosen, namely a pitch determination algorithm that uses the Subharmonic-to-Harmonic Ratio. This algorithm has been provided by the Matlab User Community (REF!) and is free of charge usable. The result of this algorithm is a fundamental frequency estimation which can be used to implement the algorithm of Ward (1999) in order to implement gesture mapping rule number 1. The algorithm can be found in appendix B

To actually implement this algorithm, the pitch values of the last 120 milliseconds have to be stored. Then, if all these values are below the 23rd percentile pitch level, an output can be performed after 700 milliseconds when all the other conditions are met. These conditions can

easily be implemented with keeping track of timestamps, and therefore no more attention will be paid to these conditions.
To determine the 23rd percentile pitch level, the voice of the speaker has to be analyzed. This shall be handled later on in the chapter 'Audio Extraction Program'.

In the C-code implementation of the program, a similar approach is chosen to implement the first mapping rule. During the initial analysis of the speaker's voice, a threshold is chosen. If the change in frequency with highest intensity is higher then this threshold, then the mapping rule is evaluated if the same conditions are met which are mentioned in Ward's algorithm.

9.3.3 Loudness / Intensity Detection

To successfully implement gesture mapping rule number two, the loudness or intensity of the speaker's voice has to be determined. As said in the Theory and Design chapter, this can be done also with the results from the FFT. The intensity of each sample is stored and again evaluated against a threshold which is determined during the initial analysis of the speaker's voice. When for a certain amount of time, the intensity of the speaker exceeds this threshold, the raised loudness is detected, and the response can be performed as is specified in the second mapping rule.
There is another approach to the determination of the loudness of the speech which is proposed by Fernandez (2004). He implemented an algorithm which computes the loudness of speech in Barks according to the algorithm proposed by Zwicker (1991). This algorithm, which is implemented in Matlab, has been evaluated and tested, but the result was that this algorithm is not useful for computations in real time. Only when the algorithm would be optimized in such fashion that it runs at least three times as fast as it does now, it can be useful.

9.3.4 Disfluency Detection

The final feature of speech that has to be detected, in order to implement mapping rule number five, is the detection of disfluency in the signal. In the Theory and Design chapter, an approach is proposed according to Shriberg, which argues that a filled pause in an audio signal is accompanied by a relatively low frequency region for a period of at least 200 milliseconds. Thus, to extract this from the audio signal, the frequencies over 200 milliseconds have to be stored and evaluated. When the standard deviation of these frequencies is smaller than approximately one hertz, a filled pause is detected and the correct response can be generated.

The detection of an unfilled pause is even easier. When the intensity of the speech is lower (on average) then the initially recorded silence level, this means there is no speech from the human and thus silence, which equals to an unfilled pause.

These approaches are the same in Matlab and C/C++.

9.4 Audio Extraction Program

In the previous chapter all the different features are explained and the only thing left to do is to actually put them in the main program which could be run. This program has been written in C/C++ as well as in Matlab. Because the implementation of both of these programs consists of almost the same implementation, apart from the actual code, both of the programs will be covered simultaneously.

The computation of the algorithms and the computation of the various features are not discussed in much detail, first of all because it would become a very detailed story with array computations and such things, and second of all, how the actual computations are performed is not really within the scope of this report.

9.4.1 Audio Recording

First the speech signal has to be recorded to compute the initial thresholds needed for the various algorithms. To record an audio sample in Matlab, simply the function *wavrecord* can be called, but in the C/C++ implementation it needed some more attention. Because of the existence of default Windows functions to record audio from the default recording device, the only real effort was to call these functions in the correct way. To do this, a wrapper has been written around these functions which made it possible to record audio with just one function. This wrapper is implemented as a class with the name *Wave*. This class consists of the following functions:

- Record( ) – records a sample of specified length
- Play( ) – plays the buffer of the class instance
- Open( ) – opens a prerecorded wave-file
- Set( ) – set the class buffer to the specified data array

The implementation of this class simplifies the wave-operations which lead to clearer code and easier error detection.

9.4.2 Feature Computations in C/C++

To actually implement the functions in C/C++, a separate class has been written in order to simplify the error detection. This class has been named *Intensity* and consists of the following functions:

- InitFourier( ) – initializes the parameters needed for the Fourier transform
- CopyInFourier( ) – copies the buffer data for the Fourier Transform
- Calculate( ) – performs the actual computations

This class computes features of a small buffer containing speech data. The features are highest intensity, highest intensity/frequency pair and average intensity. These features are computed many times per second to achieve the best performance.

The main loop of the program now records a sample of sound, copies the sound data to this class and then calculates all the features. The main loop consists furthermore of two arrays which keep track of the previous values of the features. With these current and previous values the algorithms can be evaluated.

9.4.3 Feature Computations in Matlab

After the recording of a sound sample, the following computations are made: Instead of the pitch determination method used in the C-version of the program, the Matlab version uses a Pitch Determination Algorithm provided by Sun (2001). This function proved to provide more accurate results regarding the pitch of a speech signal. The drawback is however that the function consumes a relatively large amount of time. This delay is about a quarter of the amount of time that was recorded (thus, to compute the pitch for a 20-millisecond recording, it would take approximately 5 milliseconds).

To calculate the intensity of the speech signal, the same method is used as in the C/C++ version, thus with the results of the Fast Fourier Transform. For every sound sample, the FFT is calculated and from the resulting array the intensity is computed, according to the method proposed in Milewski (1996).

The final feature that has to be computed is the disfluency. This is done with the use of the previous pitch values. This *pitch history* of the speech signal is stored in an array, and when the standard deviation of the array is smaller than a certain value (currently 0.05 Hertz, but this is adjustable) it indicates a more or less steady frequency and thus a monotone sound (like a *uhhhh*).

9.4.4 Main Loop

The main loop is for both the Matlab and the C-version very similar. First an audio sample is recorded and then the computations concerning the feature calculation are performed. When the features are computed, the history (array) is updated and the algorithms can be computed. To prevent going in too much detail here, the following pseudo-code is presented which describes the main loop of the program.

```
- record audio sample of 20 milliseconds
- compute the pitch of the sample
- compute the intensity of the sample
- add the current pitch to the history array of length 6 (=120 ms)
- add the current intensity to the history array with a length of 500
  milliseconds
- if for all values in the pitch history array, the value is less then the
  pre-computed threshold, the pitch-timer is started
- if the average intensity history is higher or lower then a certain
  threshold, a intensity or a silence behavior is sent to the agent
- if the pitch-timer exceeds the 700 milliseconds, the pitch feedback
  behavior is sent to the agent
```

9.4.5 Loudness Computation

An additional algorithm for loudness detection of a speech signal was provided by Fernandez (2004), but the implementation of this algorithm proved not suitable to run in real time, because the delay would be too big. In general, to compute the loudness of a 20 millisecond speech signal, the algorithm would take more than these 20 milliseconds to compute. Although these delays could be decreased by optimizing the algorithm or with a multithreaded program, these solutions would take too much time compared to the expected gain in accuracy over the current intensity computation.

9.4.6 Noise Reduction

Because the sound is recorded from a (analog) microphone, there will always be some noise in the signal. This noise however, could influence the pitch detection algorithm and the intensity computations. To reduce this noise in Matlab, two standard functions of the Wavelet Toolbox could be used, *wdencmp* and *ddencmp*. When applied to a (noisy) signal, these two functions will reduce the noise from this signal using wavelets. Unfortunately, due to the limited time to test these functions, it could not be verified that this would work and/or that the use of these functions contribute to the pitch detection algorithm. This is why these functions are omitted in the final version of the Matlab program.

For more information about these functions, refer to the Wavelet Toolbox Documentation (Matlab, 2004).

10 Implementation Issues

During the implementation of the various gesture mapping rules, there were of course some issues. Some of these issues relate to the programming languages and these will not be covered here, because these are basically due to the lack of knowledge of the author about either the programming language or the programming environment. The issues that will be covered here consist of extern problems during running and testing the code and obtaining useful results.

10.1 Virtual Theatre

Because of the limited availability of this theatre, intensive testing was not possible until the end of the assignment and thus it was hard to quickly resolve issues and continue work. First all the necessary adaptations had to be made and after some time they could be tested all at once. If some errors occurred, it was hard to resolve them and test again quickly because the testing environment would not be available for some time.

This was especially the case during the testing of the Mimicking program, because the trackers only operate in this theatre. It was possible to work around some of these problem (as described in the concerning chapter) but the actual (final) tests had to be done in the theatre. This led to some delays during development.

For the testing of the audio, it is also convenient to do this in the theatre because otherwise the tester would be talking (out loud) to the computer and thus disturbing the other people in the room. However, before the sound input was set up in the right fashion (this took a while because the program was written for a Windows based system and in the theatre, the microphone was connected to a UNIX based system), the testing had to be done from the workplace. To do this, as said earlier, the program was tested with previously recorded samples (wave-files).

10.2 Program Variables and Thresholds

Because both the Mimicking Program and the Audio Feature Extraction Program use (the computation of) various thresholds, it is not easy to determine what these thresholds should be and which values would give the best results. To avoid recompiling all the code every time a threshold would be adjusted, these values could be read from a file in real time. This way, only the program had to be restarted in order to adjust the thresholds.

To do this in the C-based programs, a method is applied that has been adapted from a similar method used in the Robotsoccer code from the University of Twente (Verschoor, 2004). To read the values in Matlab, a very simple method is used which reads them in the same order as they are declared in the code and assigns them their value. Although very basic, it is also very effective and fast.

10.3 Readability

The readability of the code became more and more a problem when the code base expanded. Mainly the C/C++ code was hard to read because of many array computations. The solution to this, was to move these computations to different classes which resulted (besides the already described classes) in a specific ‘Array’ class. This was possible because most arrays consisted of integers and this simplifies the computations.

In Matlab, the readability of the code had a different cause. Because Matlab is a computational environment, it is possible to perform many different operations in one line of code. This results in very compact code, but it might prove hard to see what is actually computed. To resolve this, separate functions have been written in order to clearify the meaning of certain computations.

10.4 Elvin Communication

In order to send commands to the agent in the MRE-system, the communication bus called Elvin had to be called. There are several libraries that handle the communication with Elvin, and these libraries have to be present on the computer from which the program is called. The last thing to do is to run the MRE-system from a computer with the same value for the environment variable (called ‘Elvish_scope’) as the value on the computer from where the program is run.
All his sounds trivial, but before this was found out, there were many errors going on and much effort lost due to the configuration of the systems.

When communication with Elvin from the Matlab program, the same libraries have to be called as the C-program does. However, to call these libraries (dll-files) from Matlab is not so trivial. To do that, an additional dll has been written that functions as a kind of wrapper which implements the basic functionality that is needed for the communication. This dll file is then loaded into Matlab and the functions can then be called from Matlab to communicate with Elvin.

10.5 Matlab Compiler

For this assignment, a Matlab license has been bought, but because the program has to run in the Virtual Theatre as well, there was a problem. To avoid purchasing another license, Matlab offers the option to compile the Matlab code into an executable program which can be run from any computer. In order to do this, the Matlab Compiler had to be installed on the computer. Because this is also a commercial product (and thus not for free), a trial version has been downloaded with functionality for one month. When the program has been able to run after that, the full version has to be bought.

With the Matlab Compiler installed, the executable could be created, but still it was not possible to run the program in the VR-theatre, because of the necessary libraries that were needed to run the program. After a major search on the internet, it turned out that there is a program[1] that installs these libraries on any computer. After these libraries were installed, the Matlab program worked.

[1] The program called *Matlab Component Runtime*

11 Synchronization

Because for testing purposes, both the Mimicking program and the Audio Extraction program will send commands via Elvin to the agent on screen. However, when both programs run simultaneously, it could be possible that both programs send a command at the same time. This could lead to overlapping gestures which might result in unnatural behavior. Imagine that the Audio Extraction program sends a command to the agent which tells the agent to nod his head, while at the same time the Gesture program sends a command that the agent should shake his head. Although the agent is capable of performing these gestures at the same time, this results in unnatural behavior.

To solve this problem, some sort of monitoring program had to be written, which regulates all commands that are sent to Elvin. The hard part of this is that both programs operate on a different operating system and thus on different computers. The Audio Extraction program runs on a Windows-based machine and the Mimicking program operates on a Unix-based machine. To prevent writing complex programs which communicate with each other via some sort of network, the Elvin communication bus can be used. It is possible not only to send commands to the agent, but also to retrieve commands from the communication bus. This feature is used to synchronize the agent commands and this will be explained in more detail.

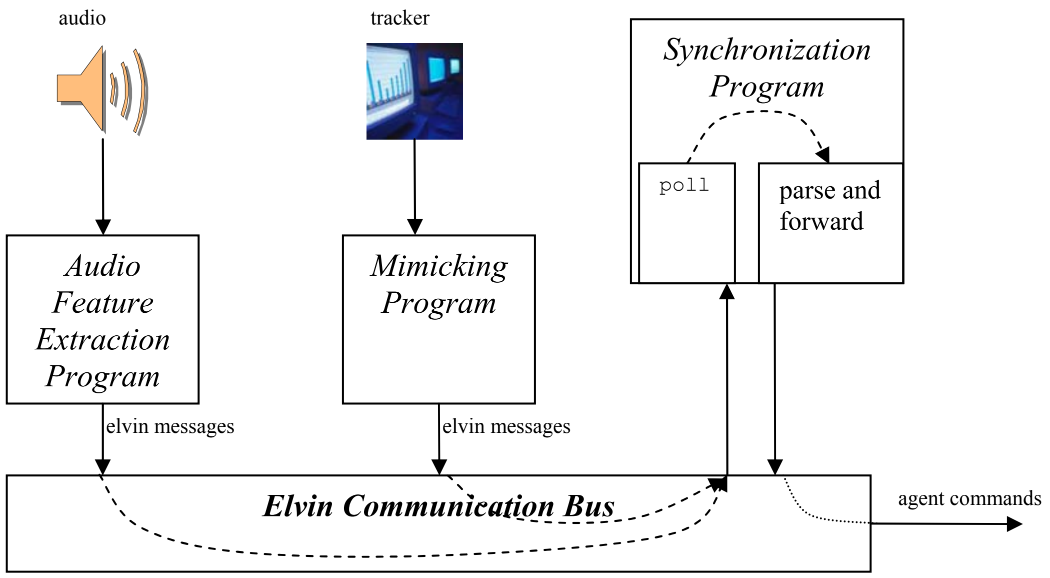
11.1 Design

Because the functionality of retrieving messages from the communication bus, the synchronization program will function as a monitoring program which retrieves all commands intended for the agent, evaluates whether the agent is performing a conflicting command and dependant of this discarding or forwarding the command. To successfully perform these operations, first the command must be parsed in order for the program whether the command is intended for the agent and which type of gesture is contained in the command. Once this parsing has been done, the function accompanying that type of gesture can be called. This function determines whether a previous gesture is still performing, and when this is not the case, a message is created which is to be sent to the agent.

Because not every command that is sent to Elvin contains gestures for the agent, in fact, only the commands sent by either of the two programs written for this assignment should be processed by the synchronization program, some sort of distinction must be provided to filter the messages sent by one of these programs. This can be done by the header of the Elvin message. When a command is sent with the agent as its destination, the header would be *dimr*. Thus, when the two programs written for this assignment change the header of the message, the synchronization program can filter the headers and thus process only the messages with a custom header, i.e. the header used in these programs, which is *gesture*.

11.2 Implementation

In order to retrieve commands from the Elvin communication bus, a callback function has to be set. This function is called automatically when the communication bus is polled for new messages. Thus, this callback function should perform the parsing of the Elvin messages, and the main function of the synchronization program should initialize Elvin in such fashion that the callback function is called, and after that constantly poll the communication bus.
This concept is visualized in the following schema:



In this schema, there are two different kinds of arrows in the Elvin Communication Bus. The two dashed lines, from either one of the programs written for this assignment to the synchronization program, represent messages with the custom *gesture* header. These messages are received and parsed by the Synchronization program. The dotted line, from the Synchronization program to the agent, contains the ‘normal’ *dimr* header which is intended for the agent.

11.3 Additions

Because this assignment proposes a certain mapping between features from the human and responsive gestures by the agent, it could be useful to change the gestures in a relatively easy way. With this Synchronization program, this is possible. This program reads the different gestures to be performed (and some other parameters) from a file when the program is started. Thus, when changing the file and starting the program again, a different mapping could be created. More information about this feature can be found in the manual that has been written and a copy of this document can be found in appendix C.

12 Testing

The previously described three programs are suitable for integration into the Mission Rehearsal Exercise system in order to test the responsive behavior performed by the agent when talking to him. The goals of these tests are twofold: first to test whether the performed behavior would be correct at the time it is performed, and second to observe the influence of the behavior of the agent on the human speaker. The results of the second goal could be used to determine whether it would be useful to implement responsive behavior into future agent models or that this behavior does not add believability to the agent.

12.1 Test 1 - The correct behavior

The first goal of the test would be to determine whether the performed behavior would be correct regarding the semantics of the speech. This test is carried out using simple conversations containing some elements where responsive behavior would be appropriate. These conversations can be found in Appendix C. The tests are performed multiple times, and the results are combined and appended to the conversations in the appendices.

12.1.1 Pitch Detection

As can be noticed in these appendices, most of the nods generated by the Pitch Detection Algorithm are performed in the right place and thus Ward's algorithm does work fairly well. The only drawback here is that this algorithm is dependant of the initial recorded pitch threshold and thus when this initial recording would be modified, the results would be different. So it is important to initialize the voice of the speaker with the correct text so the correct pitch threshold can be determined. But nonetheless Ward's algorithm is very suitable to implement to generate responsive behavior by the agent.

12.1.2 Intensity

The occurrences of nods generated by the raised intensity detection are very dependant of the initial recording and the way a speaker pronounces certain sentences. For example, this sentence:

```
What is that? You want to know the route to the theatre?
```

can be pronounced as if the speaker would be surprised to hear the *threatre* which would lead to a raised intensity towards the end of the sentence and this would result in a nod because of the raised intensity. In contrast, if the speaker would be less surprised, or the emphasis of the sentence would be on *route* rather than *theatre*, this would not result in the occurrence of a raised intensity because the part of the sentence would be too short to be detected and the intensity would be less, although a nod would be correct to perform here too.

To conclude this part, the raised intensity detection would result in correct responsive behavior, but it would be very dependant of the type of speaker and the way a speaker emphasizes certain parts of speech.

12.1.3 Disfluency

The detection of disfluency consists of two parts, which are silences and filled pauses. The detection of silences worked very well although they tended to occur too soon. This can be resolved by extending the buffer which is an easy adaptation. The detection of filled pauses however did not work as well as predicted. The allowed variation in frequency of 0.05 Hertz proved too small and this has to be increased. The results of this increase however are not tested yet because of the limited availability of the Virtual Theatre, although this would probably be the correct solution. With some more testing the correct value should be determined.

12.1.4 Physical Features

To test the correct detection of the head gestures and the posture shifts while using the tracker, the tracker should be operational. However, during testing, the tracker did not seem to respond and due to the lack of knowledge of the technical aspects of the tracker, it could not be determined why the tracker did not work. This resulted in uncompleted tests for the physical feature detections. These features have been tested before but not with an actual person wearing the tracker. The test that has been done consisted of placing the tracker on a static object and moving it around. The movements generated by this test were detected properly and thus the conclusion would be that the tests with a person wearing the tracker device should succeed as well, although this cannot be verified.

The disfunctionality of the tracker also results in the impossibility to test the mimicking program and the effect of the mirroring behavior of the agent on the human.

12.1.5 Conclusion

The testing of the various programs was not flawless. The speech feature extraction program performed fairly well and the results are promising. The initialization phase could be the bottleneck for the speech features because the thresholds are dependant of the correct initialization. Thus, it is of great importance that the initialization of the voice should be performed the right way and that the certain thresholds are computed correctly.

Unfortunately the physical mirroring behavior could not be tested properly and thus the mirroring behavior could not be tested.

12.2 The effects of the responsive behavior

12.2.1 Results

Now that at least the speech feature extraction worked properly, the effect of the responsive behavior on the human could be tested. Unfortunately, there was not much time left to perform extensive testing with many people and the differences between no responsive behavior, random responsive behavior and the behavior suggested by the mapping rules in this research.
However, some informal tests were carried out, and the results are the following:

- The fact that the agent is performing gestures while the human speak improves the natural behavior of the agent.

- The occurrence of head nods improves the believability of the agent, although the timing of the nod could be dependant of the semantics. For example, some sentences could require some thinking of the listener and thus a feedback nod should be performed after a certain delay. In contrast, when the speaker says something like *go straight ahead*, then an instant nod would be more believable than a nod after a delay.
- The proposed delay of 700 milliseconds in the algorithm by Ward proved too long. This has been changed to 200 milliseconds and this led to more accurate head nods.
- The fluency of the agent is of great importance. For example, when a posture shift is concerned, the movement from one posture to another should be without any jolting to be perceived as natural behavior. Thus, not only the fact that gestures are performed is important, but also the way they are performed.
- When the speaker changes his ‘speaking behavior’, the responsive behavior could be fooled. When for example the speaker raises the intensity for a longer amount of time, this would be detected continuously as a raised intensity which is not correct. This could be resolved by re-computing the thresholds when the voice of the speaker undergoes a big change.
- The posture shifts and gazing movement of the agent when the speaker is silent results in believable behavior, although these gestures were not as fluent as possible and sometimes the delay between two behaviors was too short. This is however easily adaptable using the constant files.

12.2.2 Conclusion

Although intensive testing was not possible within the available time, the results of the informal tests seem promising. Especially the performing of head nods and the behavior when the human is silent seem to result in more natural behavior. Even though the speech signal was quite noisy, the responsive behavior improved the natural behavior of the agent.
However, the tests showed that even the performing of simple head nods can be related to semantic information of the speech signal. This means that there is still room to improve the presented gesture mapping. In addition, more results might show up when the intensive testing is done.

13 Matlab versus C++

Because the extraction of the features from the audio signal has been implemented is both Matlab and C++, a comparison can be made between these two. This comparison is based on the experience of the researcher and gives a short summary about his points of view on both the programming environments. At the end a conclusion is provided in which the recommended environment is outlined.

13.1 Advantages C++ over Matlab:

C++ has several advantages over Matlab and most of these advantages are based on the availability of the software and the familiarity researchers may have with it.

First of all, most people that are familiar with programming will have experience in C/C++ style programming because it is one of the most popular languages. This results in a large user community and thus many people that can be asked for help. Also, because there are free compilers, there is no necessary purchase that has to be made. In contrast, the Matlab user community is much smaller and the software is not free. This is a big advantage of programming in C++.

Second, the communication with extern devices is basically straightforward in C++. For the most devices there are libraries which can be used to handle the communication due to the fact that so many people use C++. To communicate with extern device from Matlab is more difficult, because there is often no direct communication possible and thus there is always need to create an intermediate solution.

Also, because C is a very fast language and can be optimized in several ways, the language is more suitable for real time or time consuming processes than Matlab.

The final advantage of C over Matlab would be the reports of errors during development. Where C has detailed error messages and possibly a debug option (when a developer studio is used), the error reports generated by Matlab are sometimes very unclear and occur only in real time (thus only when the specified line is called and that could be very deep inside the code). With C++ there is a smaller chance that these kinds of errors occur and when they occur they are easier to resolve.

13.2 Advantages Matlab over C++

Of course, Matlab has its advantages over C++ also. These advantages are mainly based on the coding and functionality of Matlab.

First of all, Matlab is a mathematic environment and thus comes with much standard functionality which has to be written by hand in C/C++. This result is a big gain in developing time. This includes for instance memory management: where in C every variable has to be declared with enough memory (arrays), this is done automatically in Matlab. This results in less difficulties, but also in more processing time (because adapting memory allocations are time consuming).

Direct result here is that Matlab code is much compacter then C-code, because all the built-in functionality does not need many lines and that with the various toolboxes very specific functions already exists. In C these functions have to be fully understood and implemented in order to use them.

The second advantage of Matlab over C is that it is possible to import C-programs or C-code. This interaction between the two environments makes Matlab very powerful.

The third advantage of Matlab, which was especially useful in this research, is the ability to record and play wave files in an easy and fast way. There is no need to build a wrapper around Windows-based in- and output as was the case in the C-code implementation and this made the implementation in Matlab much easier and faster.

13.3 Conclusion

As is outlined in this chapter, both Matlab and C/C++ have their (dis)advantages and thus it is dependant of the type of program one will write which of the two has to be chosen. When the Matlab environment already is available or the purchase of the studio is not a big issue, then probably this would be the best environment to create mathematically intensive software. However, if the software has to run in real time, this creates some difficulties and then the best way would be (when Matlab turns out to be too slow) to create a C/C++ program which, if necessary, call additional Matlab routines to perform mathematically operations. For this research, Matlab turned out to be fast enough, so this environment is chosen to optimize the results, but it is possible that when more functionality or (computationally intensive) filters are applied, the switch to C-code has to be made. This could mean to convert the Matlab code to C-code, which could be a time consuming process.

14 Conclusion

The purpose of this assignment was to improve the responsive behavior of a virtual agent during a real time conversation. However, because much feedback behavior is dependant of the semantics of the speech of a speaker, and the current Natural Language Understanding technology is not suitable for obtaining useful results in real time, other ways had to be found to perform responsive behavior.

An intensive literature research proved that little research has been done in the area of feedback gestures, although with the combination of video analysis and the available literature it was possible to create a basic mapping between speaker features and appropriate responsive behavior.

This mapping had to be implemented into a virtual agent in order to be able to test the correctness of the mapping and the effect of responsive behavior of the agent on the human interactor.

The design and implementation proved difficult, mainly because of the communication with the target system where the agent was implemented. After these difficulties had been resolved the implementation had to be tested. However, due to the limited availability of the Virtual Theatre it was not possible to perform much testing. To partially resolve this problem, some additional programs have been written to perform the initial tests outside the Theatre. This proved useful and the correct implementation could be assumed.

The development of the speech feature extraction program proved even more difficult. First of all, the author had no experience with signal processing nor audio files. This was the first hurdle to take. The C/C++ programming language seemed best to implement this because of its speed (it had to operate in real time) and the possibility to use the Windows functionality to acquire the microphone input. When some preliminary tests were executed, the thought arose to use Matlab to perform the feature extraction. The conversion of the C-code to Matlab did not prove difficult and the computations in Matlab are easier to perform. In addition to this, there was a Pitch Detemination Algorithm available which provided much better results than the implementation in C/C++. This resulted in the decision to use Matlab for the speech feature extraction.

After the two programs had been written, the synchronization issues between these and the agent had to be resolved to prevent overlapping gestures. A message parser proved to be the solution and after solving all issues this seemed to work.

Now the complete implementation was ready and the behavior could be tested. Unfortunately, there was not much time left to perform extensive testing to obtain the final results. However, the informal tests that have been executed showed a more natural behaving agent in response to the human speaker, but it showed also that many aspects can be improved to extend the responsive behavior.

15 Recommendations

There are several aspects and features that are not implemented and that could enhance the performance of the listening behavior, or that could broaden its usability. These features are highlighted here in order to give a view on some future work. Basically, these extensions affect the interaction between the human and the agent or the internal state of the agent.

- **Filtering of the speech signal**
  There are several filters that can be applied to the speech signal in order to enhance the feature extraction. Because these features are hard to implement using normal C-code, the Matlab implementation is recommended. The features that could be considered are high/low pass filters to filter noise from the signal and to emphasize the frequency range of normal human speech, which is between 500 and 8000 Hertz.
- **Take emotions into account**
  Because emotions have a large effect on the performance of gestures in common, the emotional state of an agent can be used to affect the amount of gesturing by the agent. For example, when the human is sad, he tends to gesture less than when he is excited. This aspect was not included in this research because the inner state of the agent was not reachable from an extern program and the basis of this research was to create an extern, independent program.
- **Extend to multi-person interactions**
  The results of this research come from testing with one agent and one human speaker. When multiple agents are considered, the gazing of the human speaker could initiate individual gestures from agents. For example, when switching the gaze to an agent, this agent could notify the speaker that he is still following (independent of the semantics) with a nod.
- **Add turn-taking behavior**
  The intonation of the speaker also has an effect on the turn-taking behavior of the listeners. The pitch of the speaker tells the listeners whether the speaker is done speaking or that he will be going on (Berg, 2003). This was not included in this research because the focus of this research has only been on the listening gestures.
- **The rhythm of the speaker**
  The rhythm with which the speaker pronounces his words affects the rhythm with which the listener will speak when he will get the turn. This also is a feature of the (implicit) mimicking behavior humans perform and is thus dependant on the liking each other.

The final point that has to be made here is that this research (and all the recommendations) is based on the fact that real time speech recognition is not fast enough to provide useful information in time. Ultimately, the goal is of course to perform gestures in respond to the semantic information of the speaker, but while the current technology does not allow this, we have to satisfy with the current listener behavior in order to create a believable artificial agent.

16 Additional Programs

Besides the two main programs that are developed, a number of smaller programs are written in order to test the main programs or to simulate the Virtual Theatre environment on the actual workplace so testing could be done outside the theatre. These programs are small but very useful to verify whether something is working or not. Each of these programs is quickly mentioned here.

16.1.1 ElvinTest

This program is written to test whether to commands from the current system are received and executed via the Elvin Communication Bus. The setup of this program is very easy: when run, a menu appears which gives a choice between the execution of a head nod, head shake or a gaze. After this initial selection is made, the various parameters are asked for, and after these are provided, the gesture is sent to Elvin, and when the communication works, the gesture is performed by the sergeant character in the MRE system.

16.1.2 TrackerLog

Because of the limited availability of the Virtual Theatre, the data acquisition for the trackers was simulated to facilitate testing. Two tracker programs have been written, which either log the data in real time (that is, multiple times per second), or log the tracker data when the user asks for it. This latter tracking program could be used to store various static positions and assign a name to them.

When the programs are run, first the communication with the trackers is initiated and then the user is asked for an initial position. This position is then stored, and either the continuous logging is started, or the user is asked for a name for a static position. As soon as the user enters a name and confirms it by pressing enter, this position is written to a log file. When the user presses the ‘q’ key, the program is exited.

16.1.3 MouseInputTest

This program has been used to test the state diagram and it captures the mouse coordinates to simulate the tracker movement. These coordinates are then sent to the implementation of the state diagram which is used for the detection of head nods, head shakes and gazes and the results from this diagram is then sent to the Elvin communication bus. This program has been the final test before the actual tests in the VR Theatre.

16.1.4 ElvinDll

In order to send commands from Matlab to Elvin, a C++ library had to be used. The decision has been made to write a new library which does just that, with no extra functionality. This library is called elvin.dll and the accompanying header file is ‘ElvinDll.h’. It contains function to open and close the communication Elvin and to send commands to Elvin. In Matlab, this library can be loaded and the functions can be accessed in order to send gestures to Elvin.

The creation of this library led to various errors which were due both to the MRE-system and the dependencies there and to the configuration of the target system. On this system, Matlab was not installed and thus the necessary libraries were not present. This was eventually solved by the Matlab Component Runtime program, which installed the libraries on the target machine.

17 References

Arons, B. (1994). *Pitch-Based Emphasis Detection For Segmenting Speech Recordings*, MIT Media Lab, Cambridge Massachusetts.

Baaren, R.B. van, Holland, R.W., Kawakami, K. and Knippenberg, A. van, (2000), *"Mimicry and Prosocial Behavior"*, University of Nijmegen, Nijmegen, The Netherlands

Bavelas, J.B. Bavelas and Chovil, N., *"Visible Acts of Meaning, An Integrated Message Model of Language in Face-to-Face Dialogue"*, University of Victoria.

Bergenstrahle, M, (2003), *"Feedback Gesture Generation for Embodied Conversational Agents"*, Information Technology Research Institute, University of Brighton.

Bickmore, T. and Cassell, J. (2004), "Social Dialogue with Embodied Conversational Agents", In J. van Kuppevelt, L. Dybkjaer, and N. Bernsen (eds.), *Natural, Intelligent and Effective Interaction with Multimodal* ***Dialogue*** *Systems.* Kluwer Academic Publishers.

Cassell, J. (2000). *"Nudge Nudge Wink Wink: Elements of Face-to-Face Conversation for Embodied Conversational Agents",* in Cassell, J. et al. (eds.), Embodied Conversational Agents, pp. 1-27. Cambridge, MA: MIT Press.

Cassell, J., Nakano, Y., Bickmore, T., Sidner, C., Rich, C. (2001). *"Non-Verbal Cues for Discourse Structure."* Proceedings of the 41st Annual Meeting of the Association of Computational Linguistics, pp. 106-115. July 17-19, Toulouse, France.

Cassell, J., Bickmore, T., Campbell, L., Vilhjalmsson, H., & Yan, H. (2000). *"Human Conversation as a System Framework: Designing Embodied Conversational Agents."* In J. Cassell & J. Sullivan & S. Prevost & E. Churchill (Eds.), *Embodied Conversational Agents*. Cambridge, MA: MIT Press.

Fernandez, R. (2004), *A Computational Model for the Automatic Recognition of Affect in Speech*, Massachusetts Institute of Technology.

ICT (2003), *Mission Rehearsal Exercise*, Institute for Creative Technologies, USC, California, http://www.ict.usc.edu/disp.php?bd=proj_mre (October 22, 2003)

Intersense, 2004, http://www.isense.com/, Burlington, Massachusetts.

Kapoor, A. and Picard, R.W., (2001) "*A Real-Time Head Nod and Shake Detector"*, Affective Computing, MIT Media Lab, Cambridge.

Lakin, J.L., Jefferis, V.A., Cheng, C.M., and Chartrand, T.L., (2003) *"Chameleon Effect as Social Glue: Evidence for the Evolutionary Significance of Nonconsious Mimicry"*, Journal of Nonverbal Behavior

Mathworks (2004), *The Wavelet Toolbox Documentation*, http://www.mathworks.com/access/helpdesk/help/toolbox/wavelet/.

Matlab User Community, (2004), http://www.mathworks.com/matlabcentral/

McNeill, D. (Ed.) (2000). *Language and Gesture*, Cambridge, New York, Melbourne and Madrid: Cambridge University Press.

McNeill, D. (1992). Hand and mind: What gestures reveal about thought. Chicago: University of Chicago Press.

Milewski, B. (1996), The Fourier Transform, Reliable Software, http://www.relisoft.com/Science/Physics/sound.html

Muller, T.J. (2004), *Everything in perspective*, Institute for Creative Technologies, USC, California.

Rickel, J., Marsella, S., Gratch, J., Hill, R., Traum, T., Swartout. (2002). Toward a New Generation of Virtual Humans for Interactive Experiences. IEEE Intelligent Systems. **July/August:** 32-38.

Shriberg, E.E, (year of publication unknown), Phonetic Consequences of Speech Disfluency, Speech Technology and Research Laboratory, SRI International, Menlo Park, CA 94025

Sun, X. (2001), "Pitch determination and voice quality analysis using subharmonic-to-harmonic ratio", Proc. of ICASSP2002, Orlando, Florida, May 13 -17, 2002.

Suzuki, N., Takeuchi, Y., Ishii, K., Okada, M., (2001), *"Effects if echoic mimicry using hummed sound on human-computer interaction"*, ATR Media Information Science Laboratories Tokyo.

Verschoor, T (2004?), Robotsoccer Program Code, University of Twente, The Netherlands, http://robotsoccer.cs.utwente.nl

Ward, N. and Tsukahara, W., (1999), *Prosodic Features which Cue Back-Channel Responses in English and Japanese*, University of Tokyo

Warner, Rebecca M., et al. 1987, "Rhythmic organization of social interaction and observer ratings of positive affect and involvement" *J.Nonverbal behavior*, *11(2)*:57-74

Welji, H. and Duncan, S. (2004), *"Characteristics of face-to-face interactions, with and without rapport: Friends vs. strangers"*, presentation at the symposium on Cognitive Processing Effects of 'Social Resonance' in Interaction, 26th Annual Meeting of the Cognitive Science Society, August, Chicago, Illinois.

Zhao, L. and Badler, N.I., *“Gesticulation Behavior for Virtual Humans”*, Department of Computer and Information Science, University of Pennsylvania, PA.

Zwicker, E. Fastl, H., Widmann, U. Kurakata, K., Kuwano, S., & Namba, S. (1991), *Program for calculating loudness according to DIN 45631 (ISO 532B)*, J. Acoust. Soc. Jpn., 12(1), 39-42.

18 Appendices

18.1 Appendix A – Video Analysis

That the emotional state of both persons seems to influence the amount of gesturing. In the first video, the sergeant seemed convinced of his right and thus was not willing to adapt his internal believes to those of the doctor. The sergeant also gestured very little in response to the doctor. The doctor however, seemed aware of the point of the sergeant and was willing to (or knew he had to) adapt his internal believes. And he gestured a lot in response to the sergeant.
In the second video however, the sergeant seemed more uncertain and the doctor seemed very confident of himself. This resulted in more gesturing of the sergeant in response to the doctor then the gesturing of the doctor in response to the sergeant.

The conclusion of this could be that when a person is willing to change his internal beliefs and/or emotions during the conversation he will gesture more then a person who does not intent to change his internal believes. Whether this is true for all conversations, or whether this depends on other aspects than the internal believes is not clear and there has to be done more research in order to get a real conclusion.

18.2 Appendix B – Ward's Backchannel Algorithm

Upon detection of

--- a region of pitch less than the 23th-percentile pitch level and

--- continuing for at least 120 milliseconds,

--- coming after at least 700 ms. of speech,

--- providing you have not output back-channel feedback within the preceding .8 seconds,

→ 700 ms. later you should produce back-channel feedback.

(Algorithm by Nigel Ward (1999))

18.3 Appendix C – Responsive Behavior Manual

Author: R.M. Maatman maatman@cs.utwente.nl
Date: December 1, 2004
Description: This document contains a brief description of the Responsive Behavior Program which extracts features from a human during a conversation with an artificial agent and performs responsive behavior in such way that it should create a more natural behaving agent.

Introduction

The Responsive Behavior program consists of three different modules, one for the speech input from the human, one for the physical input from the user and one for the synchronization between these two and the commands which are sent to the agent.
For every module, a brief description is provided which contains the basic functionality of the program.

Running the Programs

There are two versions of the Responsive Behavior program that can be executed. The first one, *run-speech.bat* only runs the Speech Feature Extraction program and the Communication program. This leaves the possibility to run the Physical Feature Extraction program which uses the tracker device from *caldera*. This setup was necessary, because the tracker device will not work from a Windows-based machine.
When no tracker device is available, it is also possible to simulate the tracker behavior with the mouse. This can be done on a Windows machine and therefore the MouseMimicking program has been written. To execute this program along with the other two, the *run-all.bat* file should be executed.

Speech Feature Recognition

Description

This program analyses the speech input from the user and extracts some basic features from this speech signal which can be used to perform responsive behavior. When a certain feature is extracted, an Elvin message is sent containing the description of the feature that has been found. For the communication with Elvin a dll-file is used, which is called *elvin.dll*.

Functionality

The program is written in Matlab and with the Matlab Compiler an executable file has been generated. This means that in order to run the program, either Matlab should be installed on the computer or the Matlab Component Runtime libraries should be installed.

When the program is executed, first the communication with Elvin is initialized. After this, the background noise level is recorded to determine when there is speech and when there would be silence. Then an initial part of speech should be recorded to obtain the human's default pitch and intensity level. With this initial recording, the thresholds can be computed and the actual loop can begin. This loop consists of recording the speech for 20 milliseconds (adjustable), computing the pitch and intensity levels and comparing these to the thresholds. If a threshold is exceeded, a message is sent to Elvin.
The individual steps of this program are now described in more detail.

Initial Recordings

At the beginning of the program, there are two initial recordings necessary, one for the background noise and one to determine the default speech features of the human.

Background

By default the record time for the background noise level is one second, but this can be adjusted in the *variables.dat* file. The intensity of the recorded signal is computed and the average intensity of the recording is multiplied by a multiplier (3 by default, but adjustable) to ensure a little room for error and to prevent the breathing of the human to be recognized as speech[2].

Speech Features
After the background noise has been recorded, the voice of the human has to be recorded to determine the default pitch and intensity. The recording time is five seconds by default and is adjustable in the variables file. After the recording the pitch threshold is computed as the 23rd-percentile pitch level of the speech. The intensity threshold is the 99th-percentile intensity level. How these thresholds are used is explained in the next section.

Main Loop

When all initial recordings are finished and the thresholds are computed, the main loop can start. This loop consists of recording speech for a certain amount of time (20 milliseconds by default), computing the intensity and pitch levels of this recording and appending these values to an array. When the intensity of the recording is below the background noise level, nothing is computed because this would means that there is no speech from the human. In this case, an Elvin message is generated containing the *silence* string.
However, when the intensity is higher than this level, all the computations are done. When the average intensity over the last second is higher then the intensity threshold described above, this means an increased loudness of the signal and an Elvin message is generated containing the string *high_intensity*.
If the pitch levels of the past 120 milliseconds of recorded speech are all smaller then the pitch threshold, a timer is started[3]. When this timer exceeds the 200 milliseconds delay (again, this delay is adjustable) an Elvin message is generated with the *low_pitch* string.
Finally, the disfluency is computed: when the pitch of the recorded speech over the last 500 milliseconds is fairly constant (by default less variation than 0.05 hertz), this means a hummed sound (*uhhhh*) and a disfluency Elvin message is generated.

Variables File

As mentioned earlier, many variables are adaptable without having to recompile the code. The variables for the Matlab implementation of the Speech Feature Extraction program are located in the *variables.dat* file but because of the way Matlab reads this file, there is no real explanation of these variables, so this explanation is given here:

Variable Name	**Purpose**	**Default Value**
Sample Rate	The sample rate with which the speech signal is recorded in Hertz	44100
Initial Record Time	The amount of seconds the initial recording phase should last to determine the thresholds	5
Background Record Time	The amount of seconds the background noise should be recorded to determine the level of background noise	1
Buffer Size	The amount of milliseconds of each recording	20

[2] Ultimately, this should be done with one or more speech filters, but there was no time left to implement these
[3] According to Nigel Ward's (1999) algorithm

Window Length	Amount of milliseconds the intensity buffer should be	1000
Frame Length	Amount of milliseconds of each frame used for Pitch Determination Algorithm	20
Intensity Threshold Percentage	The percentage of intensity values that should be discarded (by default, the threshold would be the 99th percentile value)	99
Minimum Pitch Delay	Minimum delay in seconds between two occurrences of low pitch according to the algorithm by Ward[2]	0.9
Pitch Performing Delay	The delay in seconds between the detection of a low pitch and the performing of the feedback gesture	0.2
Intensity Gap	Minimum time between two detections of high intensity	0.5
Disfluency Variation	Maximum amount of variation in Hertz the speech signal can contain to qualify for a disfluency	0.5
Apply Filter	Apply the frequency filter, which contains a pass-band of frequencies between 500 and 8000	1
Intensity Multiplier	The multiplier used to multiply the average background intensity to increase this value to be more accurate.	3

Physical Feature Recognition

Description

Besides the speech input of the human, also his physical behavior is analyzed. This consists of the various head gestures a human can make (nodding, shaking, gazing) and the posture of the human. When one of these physical features is recognized, an Elvin message is created containing that feature in order to let the agent mimic this feature. By default, these postures are recognized with the use of the tracker device but due to the fact that this tracker is only available in certain rooms, the tracker data can be simulated with the mouse. A brief description of both of these versions is given below.

Tracker

The normal use of the extraction of physical features would be with the tracker. This device can be placed on top of the human's head and the relative orientation and position of this device can be used to extract the four different features, which are nodding (repetitive up and down motion of the head), shaking (repetitive left and right motion of the head), gazing (static orientation of the head in one direction) and posture shifts (change of the angle between the top of the head and the position between the human's feet).
If one of the head gestures is extracted, the exact same gesture is sent with an Elvin message to the agent. When a posture shift is detected, an Elvin message is generated containing the *posture_shift* string.

Mouse Simulation

With the simulation of the tracker with the mouse, it is not possible to perform posture shifts, and thus only the head gestures are simulated. The resulting behavior is however exactly the same as with the tracker: when a gesture is extracted, the accompanying Elvin message is generated.

Communication Program

Description

The programs intention is to retrieve messages from the Elvin Communication Bus which contain the *gesture* header. These messages are sent by two programs: the *Speech Feature Extraction Program* and the *Mimicking Program*. To prevent the overlap of gestures sent to the Agent, this Communication Program stands between both of these programs and the actual Agent.
With this program, it is (also) possible to manually change the gestures performed by the agent in response to a certain speech feature and change which Agent should be controlled.

Functionality

When Communication is executed, first an Elvin connection is established. This is done with the use of two environment variables called *ELVISH_SESSION_HOST*, which contains the address of the communication, and *ELVISH_SCOPE* which contains the scope of the communication. These variables have to be specified for the program to run.

When the Elvin connection is established, the program polls the Elvin messages every 5 milliseconds. If a message is found which contains the *gesture* header, the message is forwarded to a parser. This parser removes the possible *agent <name>* part of the message and compares the remainder of the message to various possibilities. If a matching possibility has been found, the accompanying gesture is prepared and then sent to the agent.

Supported Messages

Not every possible message is supported, only the messages which would be relevant for the performing of gestures. However, it would be fairly easy to extend the parser to support more messages. The messages that are supported are:

- *nodhead <parameters>*
  This message is just forwarded to the Agent
- *shakehead <parameters>*
  This message is just forwarded to the Agent
- *gaze <parameters>*
  This message is just forwarded to the Agent
- *sig <filename>*
  If a message starts with *sig*, it would be treated as a posture shift. This means that *filename* should be one of the following: (slouch-n-l; slouch-n-r; slouch-r-n; slouch-l-n)

- *low_pitch*
  This message is sent by the Speech Extraction Program when a low pitch sound has been found. The accompanying behavior should be specified in the constants file
- *high_intensity*
  This message is sent by the Speech Extraction Program when a high intensity speech interval has been found. The accompanying behavior should be specified in the constants file
- *silence*
  This message is sent by the Speech Extraction Program when a silence interval has been found. The accompanying behavior should be specified in the constants file.
- *disfluency*
  This message is sent by the Speech Extraction Program when disfluency has been found. The accompanying behavior should be specified in the constants file
- *posture_shift*
  This message performs a posture shift according to the internal state diagram representation of the posture the Agent is currently in. If the Agent is currently standing neutral, a random slouch (either to the left or the right) is generated. When the Agent is already slouching, the new posture would be standing neutral. This message only works if the Communication Program is the only program that controls the posture of the Agent, otherwise, the internal state diagram would be wrong.

Note: every message could be preceded by *agent <agent_name>*, but this part would be discarded anyway.
Note: for the parameters see http://mre.sf.ict.usc.edu/elvin_message/html/.

The Constants File

To change gestures without having to recompile the entire code, the mapping of the Speech Feature to the accompanying behavior is specified in a constants file named *constants.dat*. Besides the different gestures, this constants file contains some more parameters and for every parameter, its use is explained below.

Constants Name	**Description**	**Default Value**
MESSAGE_HEADER	The header of the message which is sent to the Agent	*Dimr*
CHARACTER_NAME	The name of the character to be controlled with the Communication Program	*Doctor*
WRITE_OUTPUT	Whether the messages to Elvin are also printed to the screen	*True*
MINIMAL_GESTURE_DELAY	The minimal amount of seconds between two gestures	*2.0*
MINIMAL_POSTURE_TIME	Minimal amount of seconds a specific posture should last	*4.0*
GAZE_TIME	Time a gaze gesture should last (should at least be the same as in the sequence files)	*1.5*
BROW_TIME	Time a brow gesture should last (should at least be the same as in the sequence files)	*1.5*
NR_OF_LOW_PITCHES	Number of behaviors that respond to a low-pitch detection	*1*
LOW_PITCH_BEHAVIOR_i	The actual behavior(s) that respond to a low-pitch (i = 1 .. NR_OF_LOW_PITCHES	*agent doctor nodhead 1 2 3 -7 0*
NR_OF_HIGH_INTENSITIES	Number of behaviors that respond to a high intensity detection	*1*
HIGH_INTENSITY_BEHAVIOR_i	The actual behavior(s) that respond to a high intensity (i = 1	*agent doctor nodhead 2 3 3 -7 0*

	.. NR_OF_HIGH…)	
NR_OF_DISFLUENCIES	Number of behaviors that respond to a disfluency detection	*3*
DISFLUENCY_BEHAVIOR_i	The actual behavior(s) that respond to disfluency (i = 1 .. NR_OF_DIS…)	*FUNCTION posture_shift* *FUNCTION brow_movement* *FUNCTION gaze_movement*
NR_OF_SILENCES	Number of behaviors that respond to silence detection	*3*
SILENCE_BEHAVIOR_i	The actual behavior(s) that respond to a silence (i = 1 .. NR_OF_SILENCES)	*FUNCTION posture_shift* *FUNCTION brow_movement* *FUNCTION gaze_movement*

Note: The *FUNCTION* keyword specifies that a function should be called instead of a direct Elvin command. This is particularly useful for the posture shift because the elvin command is dependant of the current posture the agent is in. To add a function which can be called, this should be done in the code in the *execute_function_call* function.

Note: If more than one behavior is specified for a certain speech feature, one of these is randomly chosen when that speech feature is sent to Elvin.

Conclusion

This document contained a brief description of the inner workings of the programs written to perform responsive behavior. The feature extractions are based on the theory presented in the theory (Maatman, 2004). With the use of both the physical and the speech feature programs, it is necessary to prevent overlapping gestures and therefore the Communication program has been written. This program retrieves Elvin messages with the *gesture* header and processes them the correct way. The use of the constants file simplifies the possibilities to extend or change the response of the Agent to certain Speech Features. With this constants file, the mapping from speech features to responsive gestures can be refined in an easy way and thus the behavior can be optimized and extended.

References

Maatman, R.M. (2004), *Responsive Behavior of a Listening Agent*, Institute for Creative Technologies, USC, California.

Ward, N. and Tsukahara, W., (1999), *Prosodic Features which Cue Back-Channel Responses in English and Japanese*, University of Tokyo

18.4 Appendix D – Test Conversations

Test File 1:

To initialize the voice:

'Hi, my name is … and I am trying to get some responses from
you. <PDA_NOD>Would you like to help me with that?<PDA_NOD> That would be
very nice…' (approx. 5 seconds).

The actual conversation:

Thank you for listening to me. <NOD> I am currently working on listening
gestures and maybe you could help me with that.<NOD> You look like, uhm,
someone who might like to perform some gestures don't you<NOD?>? What is
that? You want to know the route to the theatre<INTENSITY_NOD>? Well, that is
pretty
easy! You go straight ahead on this road, and after a mile you take
the first street on you right <NOD>. Then all the way down <NOD> and on the end
you should be able to see the theatre<NOD>. Don't Take The First Left There!!
<INTENSITY_NOD>
Then you wont reach the destination. Did you understand all this, <NOD>
and did you gesture a little?<NOD>

Test File 2:

"I am going to tell you some interesting stuff now. <NOD>Have you noticed the
lunar eclipse last month? Now that was really interesting! Just to see how the
earth moves between the sun and the moon <NOD>really fascinates me! It is like,
uhhhhhh, a solar eclipse! What? Have you never seen a solar eclipse
<INTENSITY_NOD>? Now that does not make any sense! Anyone must have seen a
solar eclipse once!!

Now this would be a boring piece of text. This weekend I went to see my
grandma. She is 81 years old and needs a stick to walk. When she would break
her stick she would be sitting on the couch the whole day. <NOD>Isnt that sad?

18.5 Appendix E – Compiling the Source Code

To compile the different programs, here are the commandline specifications.

18.5.1 Tracker Mimicking

This program should be run on the Unix machine (named *caldera*) and it consists of the following files:

Consts.h;
Consts.cpp;
constsFromFile.cpp;
constsFromFile.h;
Elvin.cpp;
Elvin.h;
Gestures.cpp;
Gestures.h;
MimickingBehavior.cpp;
Tracker.cpp;
Tracker.h.

To compile these programs, the following command should be used:

```
c++ *.cpp -L<<tt_utils_lib-dir>> -ltt_utils -L<<intersense_lib-dir>> -lintersense -orun
```

where <<...>> should contain the correct path of the library and '-orun' specifies that the output (executable) is called 'run'.

18.5.2 Matlab Speech Feature Extraction

To compile the Matlab code into an executable, the Matlab Compiler needs to be installed on the system. When this compiler is installed, use the following command:

```
mcc –m –R –nojvm Audio.m
```

where Audio.m contains the main function.

18.5.3 Communication Program

The communication program can be compiled using the Visual Studio Project which is added to the CVS repository. To compile this, the tt_utils.h header file should be known to the compiler, and this file is by default located in C:\mre\Core\Network\tt_utils\src\ directory.

18.6 Appendix H – Constants Files

18.6.1 H1 – Communication – Constants.dat

```
# The header for the Elvin message
MESSAGE_HEADER = dimr

# Character to send the gestures to
CHARACTER_NAME = doctor

# Write the output to the screen?
WRITE_OUTPUT = true

# Minimal time (in seconds) between two gestures
MINIMAL_GESTURE_DELAY = 2.0

# Minimal time a posture should last
MINIMAL_POSTURE_TIME = 4.0

# Time a gaze takes (should be the same as in the sequence file)
GAZE_TIME = 1.5

# Time a brow gesture takes
BROW_TIME = 1.5

# number of low pitch behaviors
NR_OF_LOW_PITCHES = 1

# behavior when low pitch is detected
LOW_PITCH_BEHAVIOR_1 = agent doctor nodhead 1 2 3 -7 0

# number of intensity behaviors
NR_OF_HIGH_INTENSITIES = 1

# behavior when high intensity is detected
HIGH_INTENSITY_BEHAVIOR_1 = agent doctor nodhead 2 3 3 -7 0

# number of disfluency behaviors
NR_OF_DISFLUENCIES = 3

# the actual disfluencies
DISFLUENCY_BEHAVIOR_1 = FUNCTION posture_shift
DISFLUENCY_BEHAVIOR_2 = FUNCTION brow_movement
DISFLUENCY_BEHAVIOR_3 = FUNCTION gaze_movement

# number of silence behaviors
NR_OF_SILENCES = 3

# the actual behavior
SILENCE_BEHAVIOR_1 = FUNCTION posture_shift
SILENCE_BEHAVIOR_2 = FUNCTION brow_movement
SILENCE_BEHAVIOR_3 = FUNCTION gaze_movement
```

18.6.2 H2 – Speech Feature Extraction – variables.dat

```
44100 = Sample_rate
5 = Initial_Record_Time
1 = Background_Record_Time
20 = Buffer_Size
1000 = Window_Length_For_Intensity_Computations
20 = Frame_Length_For_Intensity_Computations
99 = Intensity_Threshold_Percentage
0.9 = Minimum_Delay_Between_Two_Low_Pitches
0.7 = Delay_Between_Pitch_Detection_And_Performing_A_Gesture
0.5 = Minimum_Time_Between_Two_Intensities
0.05 = Maximum_Disfluency_Variation_In_Hertz
0 = Should_PDA_Use_Its_Voiced_Parameter
1 = Apply_The_Speech_Frequency_Hertz_Filter
3 = Intensity_Multiplier_To_Increase_Silence_Threshold
```