

SOUND SOURCE LOCALIZATION USING THE AIBO ERS-7

Bachelor project Computer Science

Marlon Richert 9692077
Tijmen Roberti 1015613
Wouter de Vries 1015842

Faculty of Electrical Engineering, Mathematics and Computer Science



1 TABLE OF CONTENTS

1	Table of Contents	3
2	Preface.....	5
3	Introduction.....	6
3.1	Sound source localization.....	6
3.2	...using the AIBO	6
4	Introduction to the AIBO ERS-7.....	7
4.1	Specifications ERS-7.....	7
4.2	OPEN-R SDK	8
5	Sound Source Localization Techniques.....	10
5.1	Interaural Differences	10
5.2	Interaural Delay Models	12
5.3	Deconfusion	15
6	Project Management	17
6.1	Project Approach	17
6.2	Teamwork	17
6.3	Tools used in the project.....	18
7	Project Execution.....	19
7.1	Requirements Analysis	19
7.2	System Design	19
7.3	Program Design	20
7.4	Implementation	23
7.5	Program Testing.....	25
7.6	System Testing.....	28
8	Project Evaluation.....	30
8.1	Recommendations.....	31
9	Conclusion	32
10	Sources and References	33
11	Appendix A: Programming Guidelines	35
12	Appendix B: Class Diagrams.....	38
12.1	SoundSourceLocalizer	38
12.2	Speaker.....	39
12.3	HeadMover	39
12.4	AibioDemo.....	40
12.5	BlinkAgent.....	40

2 PREFACE

This document is the result of our BSc project Computer Science at Delft University of Technology. A abstract technical paper about this project written by Marlon Richert can be requested from prof. Rothkrantz.

The software described in this document was used in a demonstration made by Wouter Caarls and Stijn Oomes at Robocup 2004 in an Open Challenge. A second place was the result, pre-qualifying the Dutch Aibo Team for Robocup 2005 in Japan!

3 INTRODUCTION

The purpose of the Bachelor project of Computer Science is to apply all gathered knowledge and experience in the field of information technology of the past three years to a big CS-related problem. Ideally this problem contains both research and realization. In this report we will show that our project contains plenty of these two aspects and we will give an in-depth explanation of the realized software, the technical challenges that we encountered, and of course the solutions to these challenges. But first we will start this report with an introduction to our assignment.

3.1 SOUND SOURCE LOCALIZATION...

The problem we chose for our project, as the title of this report indicates, is the localization of sound sources; determining the direction from which sounds are coming from. This project got our interest when prof. Rothkrantz introduced us to Robocup, an international tournament where teams from universities play soccer against each other using robots. Together with other Dutch universities the Delft University of Technology competed in this tournament with their own team: the Dutch AIBO Team [6]. In these competitions almost all teams utilize only one modality the AIBO offers; the camera. However, the AIBO has a range of other modalities including a stereo microphone. Prof. Rothkrantz's suggest that sound could be an interesting extra modality to recognize teammates. Every AIBO would emit a unique sound signal that is received by another AIBO, which can then determine the direction and identity of that teammate. Of course, determining the direction of sounds can be useful in other situations when a camera is ineffective.

3.2 ...USING THE AIBO

The fact that our software had to be developed on an AIBO ERS-7 meant that we got the chance to work with an state of the art piece of technology, but also gave us the opportunity to use our software on a real world platform, instead of the conventional theoretical setups used in sound source localization research. Furthermore the AIBO gave us the extra challenges that working with an embedded system brings, but more on that later. With this interesting assignment the foundation was made for a challenging project, which we would commit ourselves to in the coming three months.

This report will continue with an extensive introduction to the AIBO ERS-7 in chapter 4, and a description of several basic techniques used to localize sound sources in chapter 5. After that we will discuss the project process management in chapter 6. The analysis, design and implementation details are located in chapter 7. The evaluation of the software can be found in chapter 8. The report ends with a conclusion and a list of referenced resources. Appendix A contains our programming guidelines; appendix B contains the class diagrams of our developed software.

4 INTRODUCTION TO THE AIBO ERS-7

The AIBO was introduced to the world by Sony in 1999 as a robotic pet dog. This is reflected in his name, which is deduced from **AI** (Artificial Intelligence) and **roBOT**. It also means pal in Japanese. In the following five years since the original design the AIBO has been improved and its appearance has changed several times.

Initially it was impossible for individual developers to write software for the AIBO because the necessary interface with the hardware was kept secret. In 2002 Sony changed this policy and released the AIBO OPEN-R Software Development Kit (SDK). The AIBO became very interesting for research purposes because it provided a complete robot platform on which people could write software and conduct experiments on at a relatively low price. The most recent version of the AIBO is the ERS-7. We have developed our software for this particular version of the AIBO.



Figure 1: Sony AIBO ERS-7

4.1 SPECIFICATIONS ERS-7

The AIBO ERS-7 is a state-of-the-art piece of technology and has the following specifications:

Dimensions: 180 (W) x 278 (H) x 319 (D) mm

Weight: Approx. 1.65kg (including battery and memory stick)

CPU: 64bit RISC Processor (MIPS R4000)

CPU Clock Speed: 576MHz

Memory (SDRAM): 64MB

Program Storage Media: 16MB AIBO Memory Stick

Wireless LAN Card: IEEE 802.11b (Integrated)

Media: Memory Stick slot In/Out

Camera: CMOS Image Sensor 350,000 pixels

Microphone: Stereo, 16bit, 16 KHz

Speaker: Miniature speaker, 20.8 mm, 500 mW

Sensors: Temperature sensor, Infrared Distance Sensor (head, body), Acceleration Sensor, Electric Static Sensor (head, back (3)), Pressure Sensor (chin, paws (4)), Vibration Sensor.

For our project, interesting specifications are of course the available system resources, like the CPU and amount of system memory. Also key are the microphone specifications. As we will see later in this report the distance between the two microphones and the shape of AIBO's head are of great influence on the determination of the direction of the sound sources.

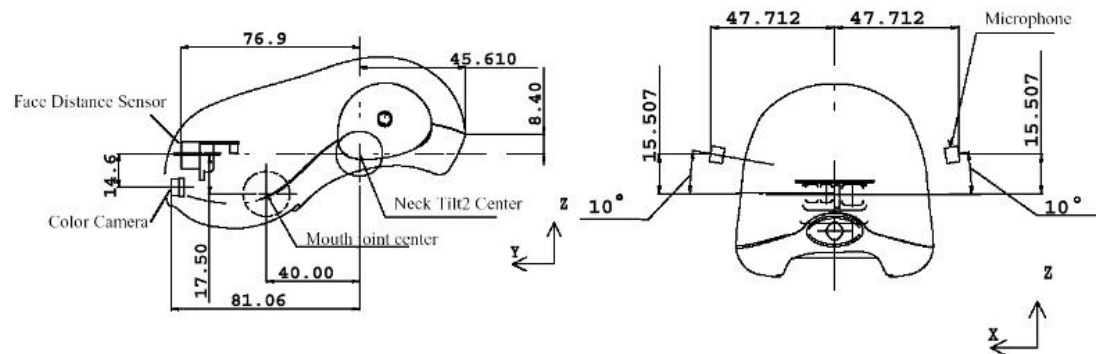


Figure 2 : Head dimensions of the ERS-7

As can be seen from Figure 2 the distance between the microphones is 95.424 mm. In the next chapter we'll see that this distance combined with a sampling rate of 16 KHz results in a maximum accuracy of approximately 13 degrees. As comparison, typical human accuracies are in the order of 10 to 20 degrees.

4.2 OPEN-R SDK

As mentioned the OPEN-R SDK was released to the public by Sony in 2002 and contains everything required to write your own software for the AIBO ERS-7 and its predecessors. The OPEN-R SDK consists of several (C++) libraries, compilers and linkers for the AIBO ERS-7 platform. The libraries provide the layer between the software and the hardware of the AIBO. Sony calls this layer the Level 2 (L2) software. The Level 1 (L1) software contains all software developed by Sony that makes AIBO act as a robotic dog. This software is not publicly available.

The OPEN-R environment was developed as a basis for Sony's future entertainment robots in such a way that changes in the hardware of the robots could easily be incorporated in the environment. The basis is formed by the AIBO Operating System, APERIOS. APERIOS provides basic OS functionality, like memory and file management, threading of running applications, etc. APERIOS is located on a Sony memory stick which is inserted into the AIBO. The user developed applications also have to be on the memory stick in the proper format.

4.2.1 OPEN OBJECTS

In the operating environment certain applications are running, called Open Objects (OObjects). These objects are very different from the well known Objects in Object-Oriented Programming. Objects in Object-Oriented Programming are an abstract method for structured programming; Objects running on the AIBO are applications, like those running on a Windows or Linux system. From now on we will use the words Object and OObject concurrently, meaning OObjects.

By standard there are a number of Objects in the operating environment responsible for several tasks, such as TCP/IP, memory management, disc access, power management etc. These Objects can communicate with each other through a framework provided by APERIOS. Sony has chosen for this highly modular model so that Objects can be easily changed without interfering with other objects (given that the interface remains the same).

The communication system works as follows: one Object is the *Subject*; this Object sends data to other interested Objects, called *Observers*. When an Observer indicates that it is ready to receive data it sends a Ready command to the Subject, which then in turn sends the data to the Observer. See Figure 3. A Subject can have multiple Observers and can of course observe other Subjects.

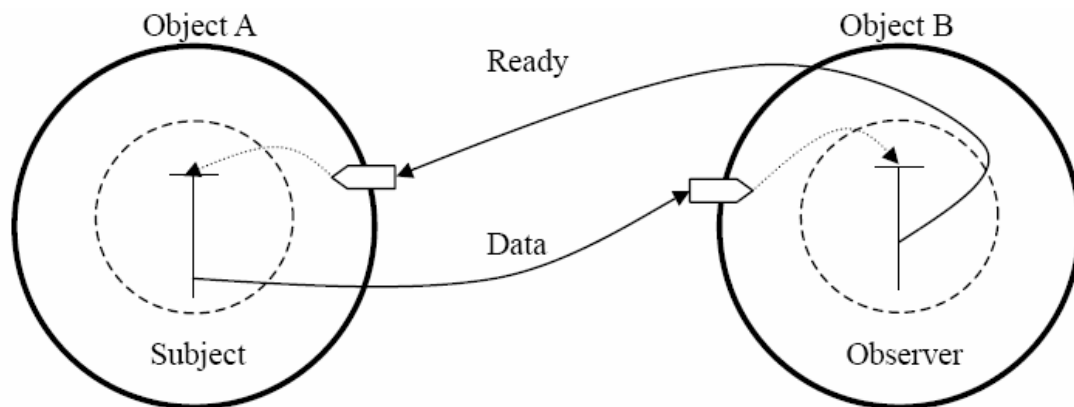


Figure 3: Communication between Objects

4.2.2 CHALLENGES OF WORKING ON AN EMBEDDED SYSTEM

As said in the introduction developing software for an embedded system is quite different from developing software on a normal pc. Aside from having less system resources than on a normal workstation, you also have to deal with the idiosyncracies of the system and it's OS. For example, basic functionality like expanding assigned memory might or might not always work on the AIBO, resulting in unexpected crashes. Also since the OPEN-R SDK uses a different compiler and linker combination than usual, strange errors can occur, especially when using advanced C++ programming techniques like templates. At first these problems took a while for us to fix and get used to, but in the end we were able to find workarounds for almost all problems we encountered.

5 SOUND SOURCE LOCALIZATION TECHNIQUES

In this chapter several techniques are discussed to determine the location of a sound, also called the *azimuth*. The azimuth is defined as being a direction expressed as the angular distance between the observer's orientation (the ERS-7's, in this case) and the direction of the sound source. In most literature about this subject and this report the azimuth is represented by the character θ .

5.1 INTERAURAL DIFFERENCES

About 100 years ago one of the pioneers in spatial hearing research, Lord Rayleigh, developed his Duplex Theory. According to this theory, there are two primary cues for determining the direction of a sound: The Interaural Time Difference (ITD) and Interaural Level Difference (ILD). These are an example of a measure to express the difference between the sounds arriving at the microphones. These measurements are then applied in a certain model to retrieve an azimuth. We first will discuss several measurements and then some models, including the one we developed for our software.

Interaural Level Difference

The ILD is the difference in intensity/magnitude of two sampled signals. It is sometimes also called the Interaural Intensity Difference (IID).

Quote from [7]:

Lord Rayleigh also observed that the incident sound waves are diffracted by the head. He actually solved the wave equation to show how a plane wave is diffracted by a rigid sphere. His solution showed that in addition to the time difference there was also a significant difference between the signal levels at the two ears—the ILD. As you might expect, the ILD is highly frequency dependent. At low frequencies, where the wavelength of the sound is long relative to the head diameter, there is hardly any difference in sound pressure at the two ears. However, at high frequencies, where the wavelength is short, there may well be a 20-dB or greater difference. This is called the **head-shadow effect**, where the far ear is in the sound shadow of the head.

The Duplex Theory asserts that the ILD and the ITD are complementary. At low frequencies (below about 1.5 kHz), there is little ILD information, but the ITD shifts the waveform a fraction of a cycle, which is easily detected. At high frequencies (above about 1.5 kHz), there is ambiguity in the ITD, since there are several cycles of shift, but the ILD resolves this directional ambiguity. Rayleigh's Duplex Theory says that the ILD and ITD taken together provide localization information throughout the audible frequency range.

Quote from [8]:

The most basic principles used to localize the azimuth of a sound source in the horizontal plane involve the inter-aural intensity difference (IID) and the inter-aural time differences (ITD) between the signals received at each ear. The IID is caused mostly by the shading effect of the head, while the ITD is caused by

the difference in distance the sound must travel to reach each ear. [...] Tones at low frequencies (less than 2 kHz) have wavelengths longer than the distance between the ears, and are relatively easy to localize. Pure tones at higher frequencies, however, present ambiguity in the determination of correct time delay, which may be a greater than the signal period, and are much harder to localize. Fortunately, pure tones are fairly uncommon in nature, and high-frequency noises are usually complex and random enough to allow unambiguous interaural delay estimation.

Estimation of elevation and distinction between sounds originating in front of and behind the listener requires more sophisticated processing than IID and ITD. The shape of the pinna and head effect received sounds in a manner dependent on arrival angle and frequency. A model of this process is referred to in the literature as the Head Related Transfer Function (HRTF) and is frequently used in virtual reality and other headphone applications where generated sounds are artificially processed to provide realistic spatial sensations to the inner ears [96]. The theory behind this model is that the brain analyzes the received sound and determines, from the qualities of the signal and learned understanding of the body's HRTF, what direction the sound must have come from.

Interaural Time Delay

The ITD is defined as the difference in time of a signal arriving at two microphones. It is one of the most important attributes for locating a sound source.

Quote from [10]:

These lateralization experiments have shown that the most important attribute for locating an auditory event is the interaural time difference (Blauert, 1997). However, the literature is somewhat confusing on the use of the term, because it generically represents arrival, phase, and even envelope temporal differences. [...]

Interaural Time Difference (ITD): A generic term used to describe any of the above time differences. Typically refers to the one that dominates the frequencies under discussion.

As can be read in the quote, usage of the term ITD can sometimes be confusing. In this document, when we use the term ITD we will mean the delay in time or in samples.

Interaural Phase Difference

Quote from [8]:

By taking the Fourier transform of each microphone signal, the relative phase at each frequency may be compared between the signals. Dividing this phase delay by its corresponding frequency gives the time delay. Phase delay is computed from the discrete Fourier transform of the left and right speech signals, $L(k)$ and $R(k)$, respectively, as described by BrandStein, Adcock, and Silverman [44]. The cross-spectrum $G_{LR}(k)$ is given by

$$G_{LR}(k) = L(k)(R(k))^*$$

where * is the complex conjugate operation. The sample (time domain) delay at a particular frequency ω_k may be calculated as

$$d = (\arg G_{LR}(k)) / \omega_k$$

Note that for wavelengths shorter than the microphone spacing, the phase delay will have an ambiguity corresponding to integer multiples of 2π . The localization system must either reject short wavelengths, or use phase unwrapping to determine all the potential delays and accumulate these in a histogram. [...]

[...] Also, reflected sound tends to shift the phase of the received signal, corrupting the delay calculation.

The fastest way to calculate the IPD is to simply perform a FFT on the samples signal and then use that to determine the interaural phase difference. However, this method is extremely sensitive to reflection and background noise. Thus, we have found it to be of no practical use in a real world application like our own. For this reason, we have ourselves developed a slightly more complex algorithm to calculate the ITD as is explained in chapter 7.

There are much more delay measurements, like the Interaural Envelope Delay. These measures are all more or less variations on the above mentioned measures and we will not discuss them here. More information about them can be find in the papers found in chapter 10.

5.2 INTERAURAL DELAY MODELS

Determining the azimuth from the observed ITD or ILD requires a model that predicts the difference in distance between the left and right ear that the signal of a sound source on a specific azimuth must travel in order to reach them, respectively. Various models have been developed over time. We tested many of them, but none produced satisfactory results. In the end we developed our own model that produced the right results during our testing.

Headless Model

Quote from [8]:

Interaural intensity differences play a significant role in sound localization in animals due to filtering effects of the head and outer ears, especially for determining elevation. However, for two microphones mounted near one another in open air, these effects are minimal, and are ignored here. This dissertation project involves localization in azimuth only, based entirely on interaural time difference. Two assumptions are necessary for this to work properly: (1) The speaker is always assumed to be on the "front" side of the microphone pair, limiting the localization to a range of 180 degrees; and (2) the speaker's location is assumed to exist approximately within a plane parallel with the floor and level with the microphones. This approximation is forgiving enough to allow for differences in height and for both standing and sitting positions, but does not allow the speaker to hover over or duck below

the array. Also implicit in this configuration is the assumption that two speakers will not be positioned one above the other when viewed from the microphone array's perspective. [...]

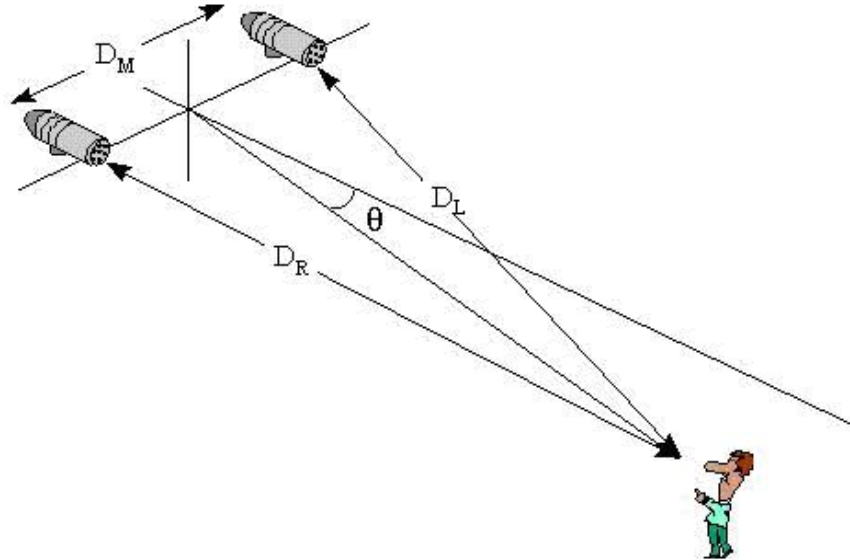


Figure 4: Headless model

Figure 4 shows the geometry used for calculating sound direction based on interaural delay. Calculation of the ITD between two microphones specifies a hyperbolic locus of points upon which the corresponding sound source may reside. For target distances (D_L and D_R) much greater than the microphone spacing D_M , the target bearing angle may be approximated as

$$\theta \cong \sin^{-1} \left(\frac{D_L - D_R}{D_M} \right)$$

Rewriting the difference in target distance in terms of the interaural time delay, one obtains

$$\theta \cong \sin^{-1} \left(\frac{V_{sound} \cdot ITD}{D_M} \right)$$

where V_{sound} for a comfortable indoor environment is approximately 344 m/s.

The headless models assumes an empty space between the microphones. In our case however, we have AIBO's head to deal with. Since most frequencies don't pass through the head they have to go around it, resulting in greater delays. The headless model will therefore generally lead to an underestimation of the azimuth.

5.2.1 DUPLEX THEORY

Originally, we tried to use Lord Rayleigh's (John Strutt's) duplex theory. However, our real world measures with the ERS-7 have definitely shown with great certainty

that it is possible for ITDs to occur that are **greater** than can be predicted with this model. Therefore, we have decided to design and use a new model of our own.

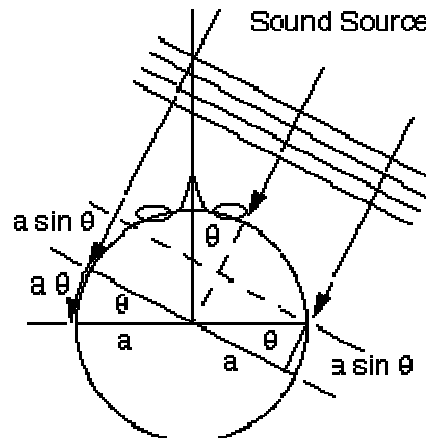


Figure 5: Lord Rayleigh's model

5.2.2 OUR SPHERE MODEL

A serious problem with Lord Rayleigh's model is that he assumes that the left and right signals arrive at the head in two parallel lines. Unfortunately, this is only the case if the sound source is of the exact same size as the head or if the sound source is very far away. In our test environment, however, we also make use of the ERS-7's own speaker (while another ERS-7 fulfills the role of observer), which is significantly smaller than its head. Because of this, the paths of the left and right signal are at an angle to each other. This phenomenon becomes even stronger the closer the sound source is positioned to head.

Our model represents the sound source as a point source, which is the situation most commonly encountered. This is approximated by the following formula

$$ITD \cong \frac{v_s}{r_m} ((2 - \chi)\theta + \chi \sin(\theta)),$$

where v_s equals the speed of sound, r_m equals microphone radius (the distance of either ear to the center of the head), θ is the azimuth, and χ is a function of the distance of the sound source to the head. The properties of χ are that it is always positive and correlates with the distance of the sound source to the head, and that as the sound source moves toward the head, χ converges to zero, while as the distance of the sound source to the head grows toward infinity, χ converges to 1 (which leads to Lord Rayleigh's formula). For computational simplicity in converting the ITD to the azimuth, we assume that the distance of the sound source to the head is effectively zero, so that χ equals zero, which leaves us with

$$\theta \cong \frac{r_m}{2v_s} ITD.$$

Since in practice the distance of the sound source to the head will never be completely zero, it would seem that any θ produced by this formula leads to a slight overestimation of the ITD in this model, and conversely, any ITD would lead to an underestimation of the θ . However, in practice, this bias is only noticeable when the sound source is located a significant distance above the azimuth plane or when the

distance of the sound source to the head becomes larger than roughly three meters. Furthermore, since the speaker of the ERS-7, which serves as the sound source, is located significantly closer to the ground than its ears, interposing parts of the observing ERS-7's body (mainly its shoulders and nose) lead to additional delays that even further obscure this bias.

The accuracy of the azimuth detection is of course limited by the available hardware. In the case of the ERS-7, the limiting factors for the determination of the ITD are the microphone-sampling rate and the distance between the microphones. The smallest detectable delay between the left and right channel is one sample. Since the ERS-7 microphone has a sampling rate of 16 KHz, this corresponds to a time delay of $1 / 16000 = 0.0625$ ms.

The largest possible time delay can be calculated as follows (assuming a round head and the microphones opposite to each other):

$$\delta \cong \frac{\pi \cdot r_m}{v_s}$$

Where δ is the delay in seconds, r_m the microphone radius (the distance between the microphones and the center of the head) in meters and v_s the speed of sound in m/s. Substitution of the proper values results in a maximal time delay of 0.4364 ms. Multiplying this by the number of samples per ms results in a maximum delay in samples of 7.

5.3 DECONFUSION

After determination of the ITD and conversion to an azimuth it is still unknown if the sound source is a front angle or a back angle. Actually, the source can be on an entire cone in 3D space: the so-called *cone of confusion*. The cone of confusion represents all possible locations of a sound source given a certain ITD or ILD. The only way to discover exactly where a sound source is located in this cone is to move the microphones and measure again. Then, by comparing the measurements, the direction of the source can be determined. In order to simplify this problem we assume that the sound source is on a plane level to the microphones and the ground.

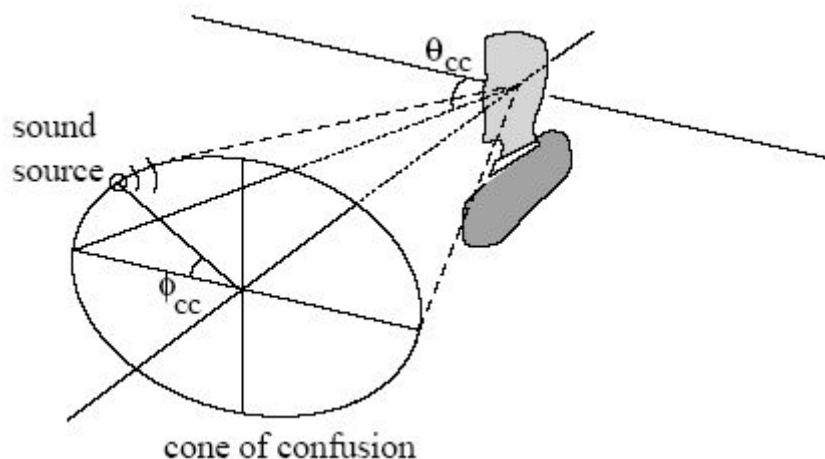


Figure 6: Cone of confusion

Our *deconfusion* used in the software operates on some very simple, but powerful premises. Given our head rotation δ :

- If we previously localized a sound source at a certain azimuth θ_{old} and we rotated our heading **toward** it:
 - If $-\pi/2 + \theta_{old} < \delta < -\pi/2 - \theta_{old}$ or $\pi/2 - \theta_{old} < \delta < \pi/2 + \theta_{old}$, then the sound source must be in **front** our heading.
 - If $|\theta_{new}| > |\theta_{old}|$, then the sound source must be **behind** our heading.
- If we previously localized a sound source at a certain azimuth θ and we rotated our heading **away** from it:
 - If $-\pi/2 - \theta_{old} < \delta < -\pi/2 + \theta_{old}$ or $\pi/2 + \theta_{old} < \delta < \pi/2 - \theta_{old}$, then the sound source must be **behind** our heading.
 - If $|\theta_{new}| > |\theta_{old}|$, then the sound source must be in **front** our heading.
- Otherwise, we assume per default that the sound source is in **front** our heading.

Notice that if we do rotate our head too far, wrapping can occur which can cause the deconfusion to fail. In practice, we avoid this by not allowing the ERS-7 to rotate its head more than $\pm\pi/2$ between observations, which is a safe value for any azimuth.

Obviously, if we do not rotate our head between two periods of observations, no deconfusion is possible. Therefore, in those cases where the ERS-7 would otherwise not move its head enough between observation periods to allow for deconfusion, we force it to make *saccadic* head movements. These randomly distributed motions are relatively small, but significantly large enough motions to allow our software to continue to perform successful deconfusion at all times.

6 PROJECT MANAGEMENT

Until this project, our project experience was limited to the ones in the curriculum of the BSc Computer Science. These projects required an estimated 56 hours of work per person. For this project however, a full three months are scheduled. To ensure a smooth progression of a project of this size we made important decisions regarding process management and teamwork right at the start.

6.1 PROJECT APPROACH

Because the AIBO as well as the subject of sound source localization was unknown to us, we decided to use an iterative design model. An iterative model allows the use of knowledge acquired during development to improve the final design/product by revising parts of the program. We decided to use the V-model (see Figure 7). In this model each step has a matching validation process and when defects are found, cause a loop back to the corresponding development stage. The next chapter's paragraphs will reflect the steps in this model.

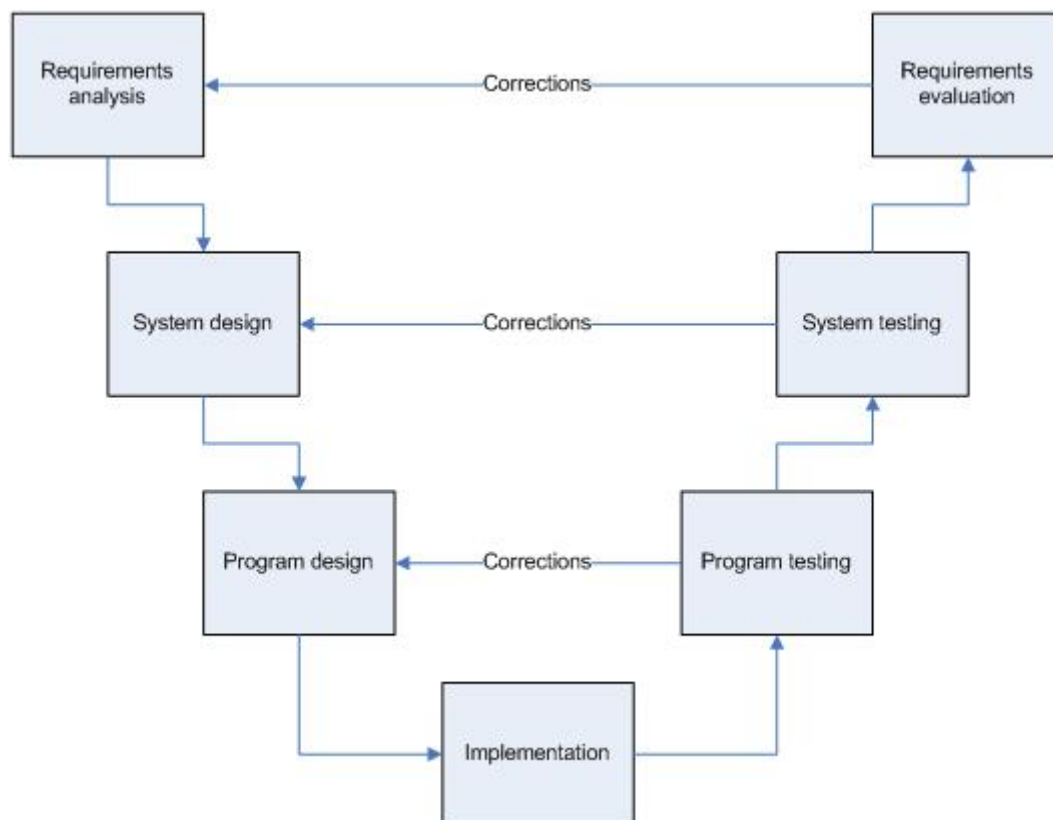


Figure 7: V-process model

6.2 TEAMWORK

For maximum efficiency and productivity during the project, the teamwork has to be excellent. To eliminate the wasting of precious time with logistical problems we decided to use the same working environment and made a clear set of rules to get a concise style during the entire project. At the start of the project we had some bottleneck problems because of missing hardware like AIBOs or WiFi base stations,

but most of these were solved pretty quickly. The following are some of the decisions we took regarding teamwork:

Programming Guidelines

The Programming Guidelines (PG) were decided on at the start of the project to ensure a consistent coding style throughout the project. The PG contain rules for writing code, comments, file names and directory structure. The use of the PG guarantees a uniform style, reducing logistical problems and improving general readability of the code for the other team members. The complete Programming Guidelines (in Dutch) can be found in the Appendix.

Concurrent Versions System

Concurrent Versions System (CVS) [4] is a software configuration management program to support teams working simultaneously at the same set of source files. CVS allows multiple users simultaneous access to files and the merging of any changes they make. It also keeps version information and supports the storage of project related files.

Eclipse

Eclipse was chosen as our main editor [3]. This editor works as well in Windows as in Linux and is because of its standard support for CVS very suited for our project. Having all team members use the same editor also eliminates potential conversion and format problems.

Peer review

To get the code at the highest possible quality we used peer reviews. Originally known as “egoless” programming, peer reviews allow team members to look at and verify another team member’s code. Improvements in the code and/or comments and the occasionally found bugs are discussed and the code is updated.

6.3 TOOLS USED IN THE PROJECT

In addition to the aforementioned tools, we also used MATLAB to design, simulate and tests some of the features in our software. The entire application was written in C++ and compiled in Linux using the tools provided by the OPEN-R SDK. Debugging was done with a WiFi network connection to the AIBO. This document was produced in Microsoft Word 2003, diagrams were designed in Visio 2003 and Rational Rose. Configuration management for the report was done with Microsoft Sharepoint.

Initial testing of the software was done with a pair of Labtec speakers and a program called ToneGenerator. Later on, testing was done with multiple AIBO’s.

7 PROJECT EXECUTION

This chapter describes in detail how the software is produced and what technical solutions are used. The paragraphs will generally follow the phases in the V-process model discussed in paragraph 6.1.

7.1 REQUIREMENTS ANALYSIS

As mentioned earlier, there were two major objectives to fulfill. AIBO should be able to determine the direction of a sound source (for example another AIBO) and recognize different sounds (frequencies). The desired result is that you can have two AIBOs, where one produces a certain sound and the second AIBO can determine from that sound the location of the first AIBO. Also some way of demonstrating the software is required. To provide this functionality the following requirements were set:

- AIBO has to be able to determine the azimuth of a specified frequency or specified frequencies. This requirement allows for a flexible approach for locating sound sources. For example: if you would want to locate human voices, you can configure the software to recognize the characteristic frequencies in voices.
- AIBO needs a way to produce sounds. This is necessary for AIBO-to-AIBO localization.

7.2 SYSTEM DESIGN

To keep the system as modular as possible, it was decided that all sound source localization functions would be contained in one OObject, `SoundSourceLocalizer`. The specific frequencies `SoundSourceLocalizer` will listen to has to be specified by other Objects.

`SoundSourceLocalizer` will receive the byte input stream in an `OSoundVectorData` format from `OVirtualRobotComm`. `OVirtualRobotComm` is an OObject provided by Sony's L2 layer and handles communication with AIBO's sensors. After conversion, `SoundSourceLocalizer` will extract the azimuth from the data. The azimuth then can be sent to all Objects observing `SoundSourceLocalizer`.

To make AIBO-to-AIBO communication a possibility, a way to produce different tones is needed. The Object `Speaker` is designed to fulfill that role. It is responsible for producing sounds of the frequency and duration specified in the given `SoundFile`. Various amplitude modulation functions also need to be available, like triangular modulation.

Finally, an Object to demonstrate the functionality of the other Objects is needed. For this purpose we designed the Object `AibioDemo` (**A**ibo + **A**udio). `AibioDemo` has to be able to specify what frequencies `SoundSourceLocalizer` should listen to and output the results by moving AIBO's head in the direction of the source. To move the head, another Object called `HeadMover` is needed. `AibioDemo` will also give visual cue's regarding the status of the sound source. If AIBO thinks the sound source is right in front of him, it will blink his green LED's. If the sound source is at a direction it is unable to point at with his head (for example behind him) it will blink his red LED's. When tracking sound sources it display a moving bar of white LED's. An Object called `BlinkAgent` will provide the LED display. Figure 8 shows an overview of the system design.

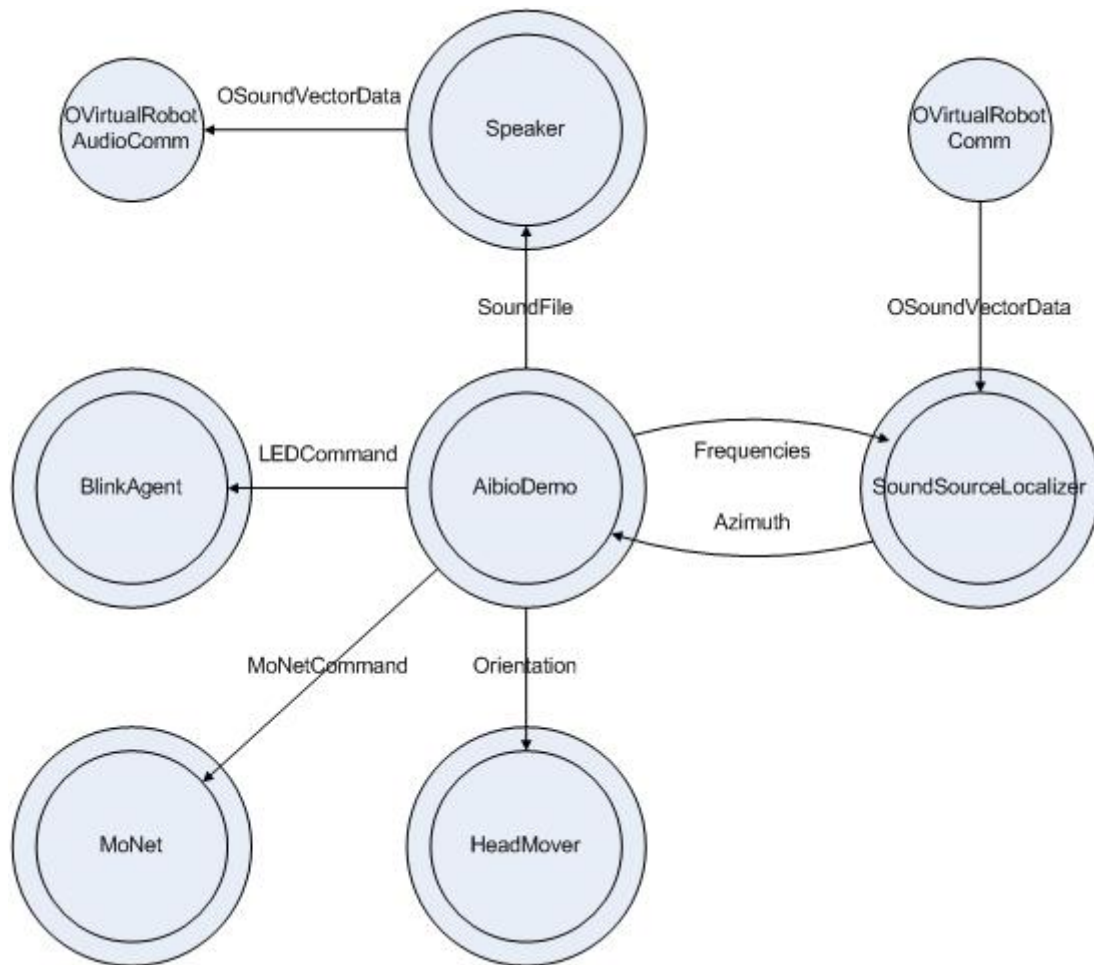


Figure 8: System model

7.3 PROGRAM DESIGN

Now that we had specified the design of all Objects, design of the code structure started. Each Object has to be developed in a certain framework, called a Core Class. The Core Class is like a standard C++ class but with extra functions for startup, shutdown and functions for communications with other Objects (see Figure 9). A Core Class is comparable to a normal program which requires a main() function.

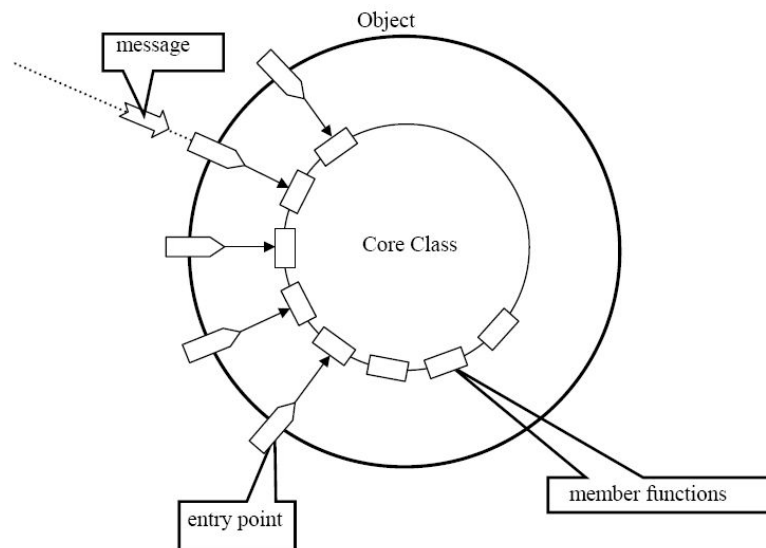


Figure 9: Core Class

7.3.1 SOUND SOURCE LOCALIZER

SoundSourceLocalizer is the Object used to calculate the azimuth and to do so several steps has to be taken:

- Convert microphone byte stream to double arrays so calculations can be performed on the data.
- Apply a Fourier Transform on the data, for frequency and phase information.
- Calculate the azimuth.
- Send azimuth to all observing Subjects.

It was decided that that three of these functions would be performed in their own class. A static class Converter was designed for converting data; a static class FFT was designed for the Fourier Transform and a class AzimuthCalculator was designed for all calculations regarding the azimuth. The Class SoundSourceLocalizer would deal with all AIBO hardware and communications with other Objects. Figure 10 shows the class diagram of the initial SoundSourceLocalizer design.

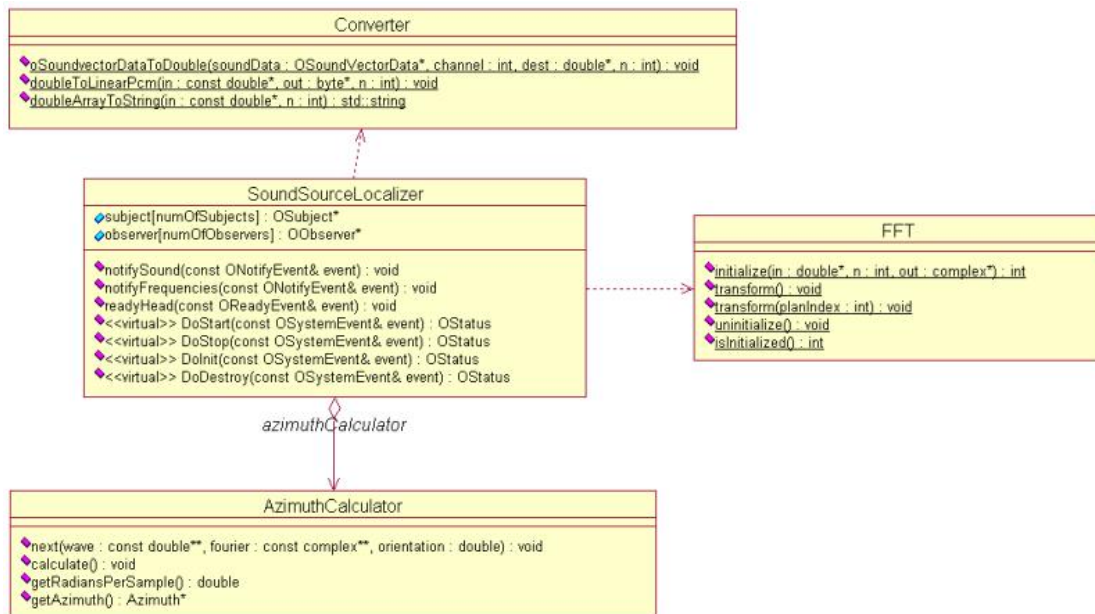


Figure 10: SoundSourceLocalizer Class diagram

7.3.2 SPEAKER

The Speaker OObject is responsible for emitting sounds that would be recognized by another AIBO with our software. The specific type of tone has to be specified in a file. Object Speaker consists of two classes (see Figure 11). Core Class Speaker is responsible for the communication with other Objects and for outputting the sound to the hardware. Class ToneGenerator produces signals with the desired frequency and modulation. Speaker also uses Converter to convert the generated signal to a Linear PCM encoded byte stream and FileParser to read sound settings from stored files.

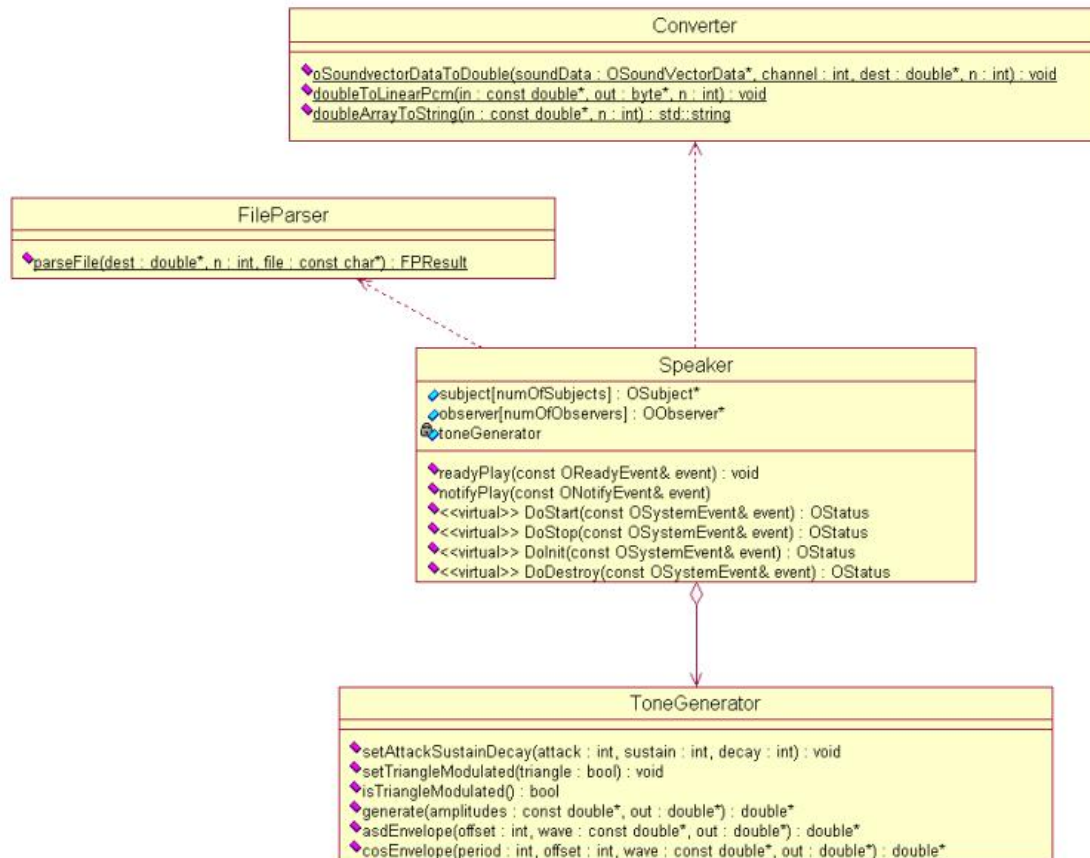


Figure 11 : Speaker Class diagram

7.4 IMPLEMENTATION

7.4.1 SOUND SOURCE LOCALIZER

The SoundSourceLocalizer Core Class is the template for an AIBO OObject and has all necessary functions and attributes necessary to make it run on the AIBO. The class accepts microphone sound input from AIBO's OVirtualRobotComm Object (which is a virtual Object running on the AIBO). The sound data is converted using the class converter, the Fourier Transform is calculated and the finally, the azimuth. The azimuth produced by AzimuthCalculator is the azimuth angle relative to AIBO's head. SoundSourceLocalizer changes the azimuth so that it is relative to the direction in which AIBO's body is facing. The calculated azimuth is then sent to Objects observing SoundSourceLocalizer.

Converter

The Converter class performs various data conversion operations. The operation used by SoundSourceLocalizer converts the Linear PCM encoded sound signal from AIBO's microphone to an array of doubles. Converter also has a function to convert an array of doubles to a Linear PCM encoded byte stream, used by the Object Speaker. Other functions include array to string functions used for debugging purposes.

FFT

For the actual Fourier transform calculations performed by the class FFT we used the C library FFTW (Fastest Fourier Transform in the West) [5]. This library is developed

at M.I.T. and won the 1999 J. H. Wilkinson Prize for Numerical Software. Getting this library to work on the AIBO was fairly hard, because the automatic configuration utility only works on Unix platforms. There were also no existing configurations for the AIBO (no surprise here) so we had to configure and compile everything manually.

AzimuthCalculator

When performing our sound source localization with the ERS-7, we are merely interested in determining the *azimuth* of the observed sound source. To be able to discern multiple sound sources from one another and facilitate the distinction between signal and noise, we perform the sound source localization on a number of separate, predefined frequencies only.

Each time frame, the AzimuthCalculator receives the sampled input signal of the left and right ear, the Fourier transforms of these two signals, and the AIBO's head orientation for the current time frame. It uses this information to calculate a deconfused azimuth.

To prevent sound source localization from being performed on frequencies that contain only background noise, we establish a reliability threshold. To this end, we use the Fourier transformed signal of both the left and right channel of the sampled signal. We define a reliability threshold t as

$$t = \mu + k \cdot \left(\frac{|f_i^l| + |f_i^r|}{2} - \mu \right)^2,$$

where

$$\mu = \text{mean}_{\forall i} (|f_i^l| + |f_i^r|)$$

and where

$$|f_i^l| \text{ and } |f_i^r|$$

are the amplitudes of the i -th Fourier components of the left and right signal, respectively. k is a constant which for our purposes has a value of 2, so that the sound from the ERS-7's own speaker does not raise the threshold too much, which would cause it to view the signals of other ERS-7s as unreliable. The signal corresponding to the j -th Fourier component is then considered reliable if and only if

$$(|f_j^l| > t) \vee (|f_j^r| > t).$$

Note that since we perform the FFT on a sampling frame of 16 ms, the j -th Fourier component corresponds with a frequency of $j / 0.016 = j \cdot 62.5$ Hz. For all reliable frequencies the delay is then calculated by determining the Interaural Phase Difference (IPD) as discussed in 5.1.

7.4.2 SPEAKER

To produce a sound Speaker must send an `OsoundVectorData` struct to `OVirtualRobotAudioComm`. This struct contains a byte array which is Linear PCM encoded. To construct this struct, Speaker first reads the file indicated by another Object using `FileParser`, which returns an array of doubles. The n -th element of these

doubles represents the relative amplitude of the $n \cdot 62.5$ Hz tone. Speaker then calls the generate function of the ToneGenerator class to convert the amplitudes into an array of triangular modulated doubles representing the samples to be played. The doubles are converted to Linear PCM encoded bytes by the Converter class, buffered, and then sent to OVirtualRobotAudioComm.

7.5 PROGRAM TESTING

During testing we noticed that the IPD, the measure we initially used to determine delay, is extremely susceptible to reflections. To solve this problem we tried various approaches, like envelope extraction. None were satisfactory. In the end we decided to manually filter out the frequencies we were interested in (contrary to using a Fourier Transform) and use the cross correlation to find the delay in samples between the left and right channel. To perform the signal filtering a new class had to be developed, appropriately named FilterBank. The updated class diagram can be seen in Appendix B: Class Diagram.

FilterBank

A Filter Bank is a combined set of parallel filters all filtering a single input signal. Class FilterBank is designed to do just that. A given input signal is filtered by a number of FIR filters. A FIR Filter performs the following operation on an input signal $x[n]$

$$y[n] = \sum_{k=0}^M b_k x[n-k]$$

where $y[n]$ is the output signal, b_k the filter coefficients and M the number of filter coefficients. (For more information on filters, see [11].) To filter out the desired frequency, bandpass filters were developed in MATLAB. At first, we had 64 filters with a passband of 62.5 Hz and 8 filter coefficients. Preliminary testing and simulation in MATLAB showed that the performance of the filters was sub-par. In order to get adequate performance we determined that we needed to use much more filter coefficients. Since this increases the computational costs of the filtering operation, the number of filters was reduced. The new filter coefficients were developed with MATLAB's Filter Design & Analysis Tool (see Figure 12).

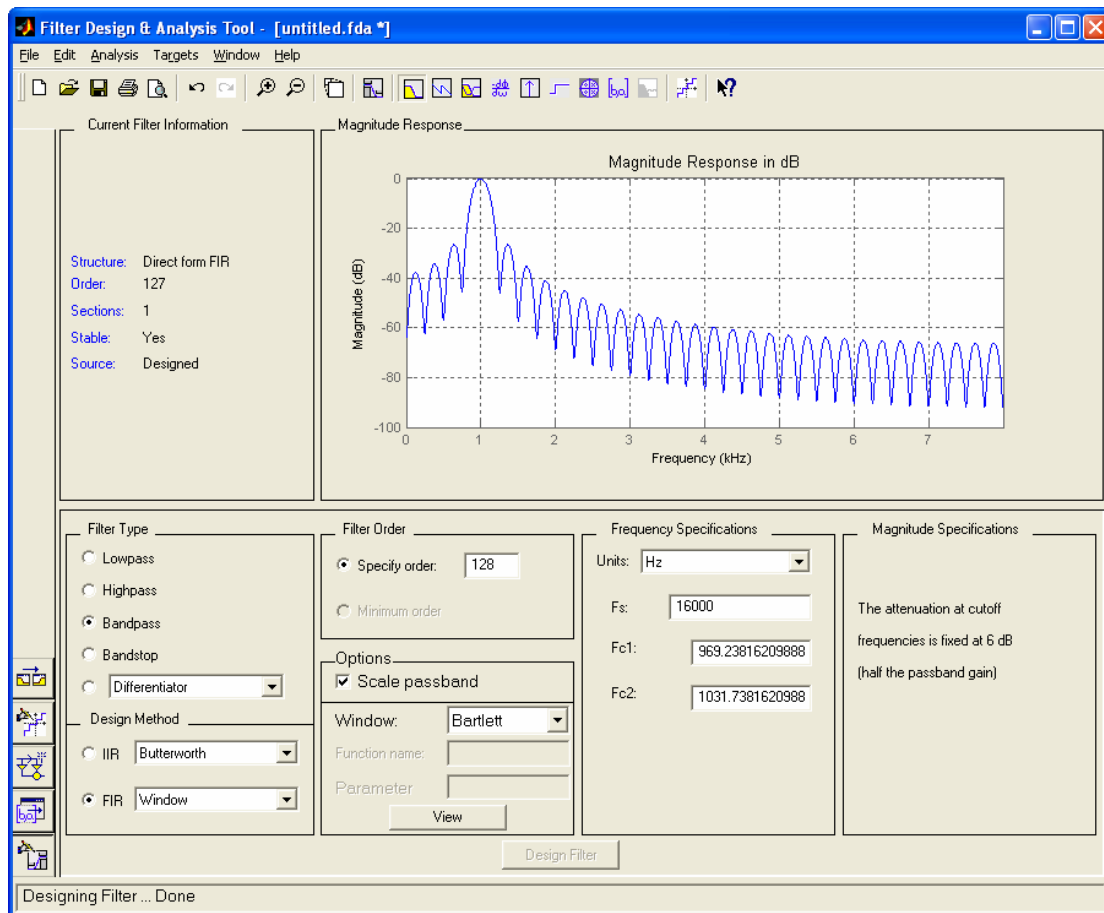


Figure 12: Filter Design & Analysis Tool

With the FDA Tool 8 filters were designed. Every filter is a 128th-order FIR filter with a bandwidth of 62.5 Hz and a central frequency of an integer multiple of 250 Hz. The filters were designed using a 129-point Bartlett window. One beneficial property of this specific type of filter is that it has a magnitude response of -45 dB or less for all frequencies that are multiples of 250 Hz, other than the central frequency (see Figure 13).

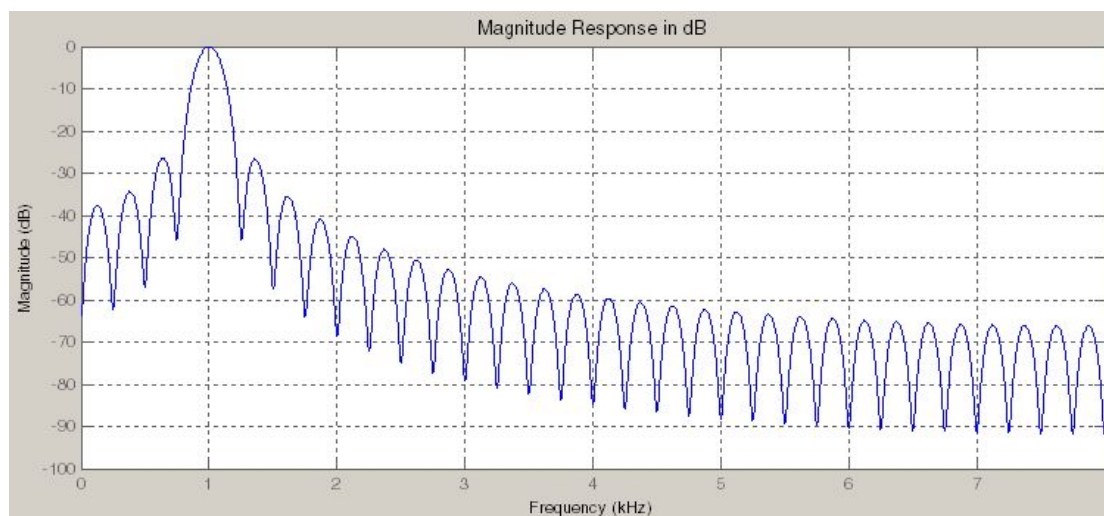


Figure 13: Magnitude response of 128th-order Bartlett FIR Filter

Normalized Cross-Correlation

After filtering the input signal, we want to determine the delay between the left and right filtered signal of that specific frequency. To do so, we try to maximize the cross-correlation between the left and the right signal. The cross-correlation is the preferred measure to determine the coherence between two signals. For our application we make use of the windowed cross-correlation

$$\rho_{lr}[d] = \sum_{n=N_1}^{N_2} l[n]r[n-d]$$

where $l[n]$ and $r[n]$ are the discretely-filtered left and right signal, respectively, and where N_1 and N_2 define a window in time for which the cross-correlation is calculated. The value of the delay d that maximizes $\rho_{lr}[d]$ is the ITD in samples. This is then multiplied by the duration of one sample to acquire the ITD in seconds. Since it is known that the maximal delay in our model can only be 7 samples, we optimize the cross-correlation by only using values for d that are in the discrete interval $[-7,7]$.

A problem with this cross-correlation method is that for signals of constant amplitude it produces more than one maximum at intervals equal to the signal's frequency. In order to overcome this ambiguity, we use an amplitude-modulated signal as our sound source. More specifically, we use triangular modulation, since this always produces the greatest difference in amplitude between periods. Furthermore, when the ITD is larger than one-half the period of the signal, the smallest delay that maximizes the cross-correlation will actually be smaller than the real ITD. To avoid this issue of wrapping, we do not process signals of frequencies that have wavelengths shorter than half the distance between both ears.

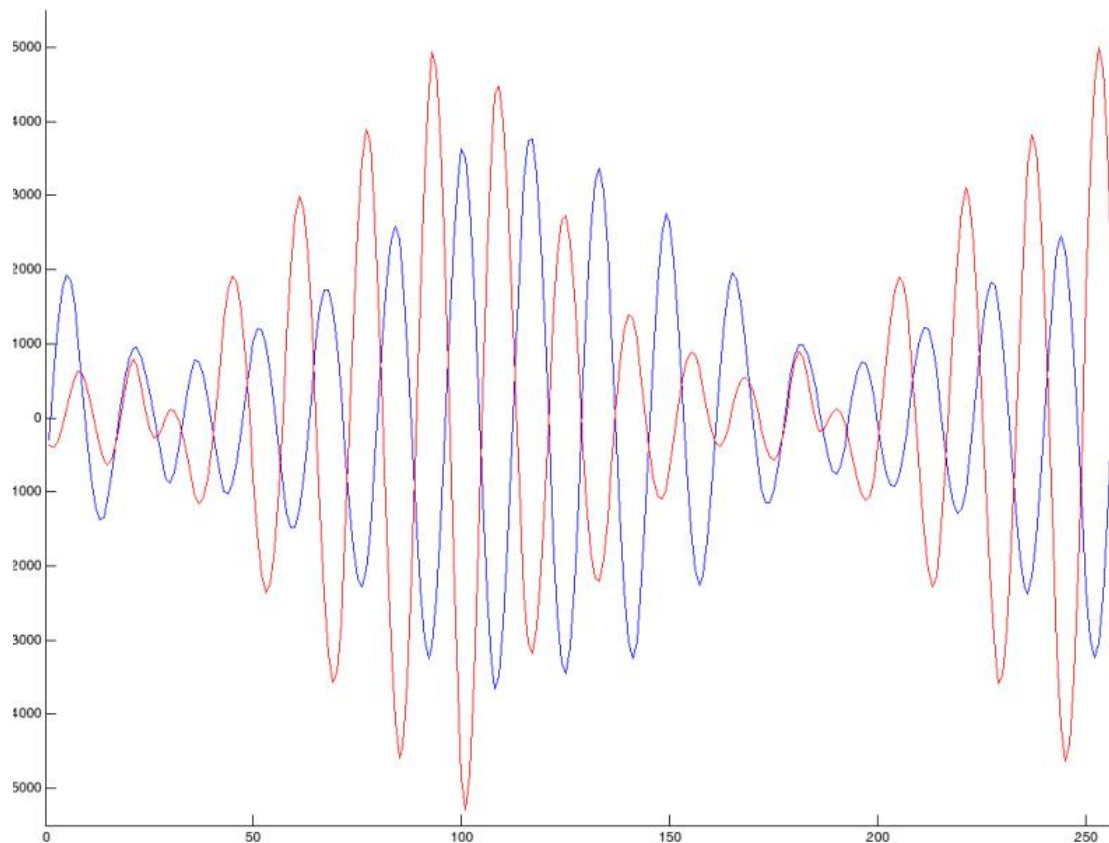


Figure 14: 16ms, 1000Hz signal sample captured with AIBO's microphones

Figure 14 shows an actual signal captured from the AIBO's microphones when a 1000Hz, triangular modulated tone is generated. The red line shows the input on the left microphone, the blue line shows the input on the right microphone. A plot of the calculated cross-correlation can be seen in Figure 15. The results show that a delay of 7 samples produces the highest correlation, which means that the source must be entirely to the left of the AIBO.

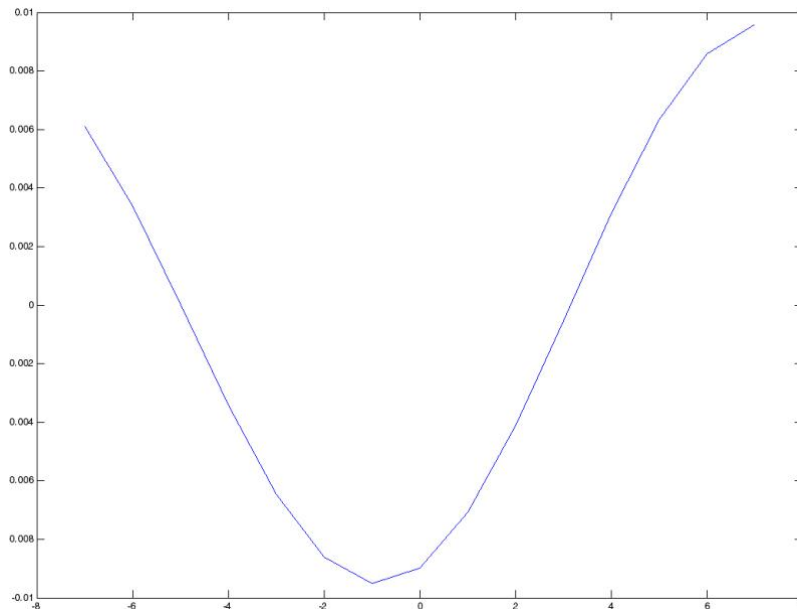


Figure 15: Cross-correlation of the 1000Hz signal for various values of d

7.6 SYSTEM TESTING

At first, SoundSourceLocalizer was designed as a passive Object. It would only receive data from the AIBO and not control any parts of the robot itself. The necessary movement of the head for deconfusion would have to be done by other applications. After testing however, we noticed that the calculated azimuths were fairly inaccurate. This could be attributed to the sound produced by moving the head while sampling. It was decided that we would also add an active mode of operation to SoundSourceLocalizer, in addition to the already existing passive mode. In this active mode SoundSourceLocalizer would listen for a sound, move the head towards the perceived source using the Object HeadMover, listen again, and repeat. This new approach is illustrated in Figure 16.

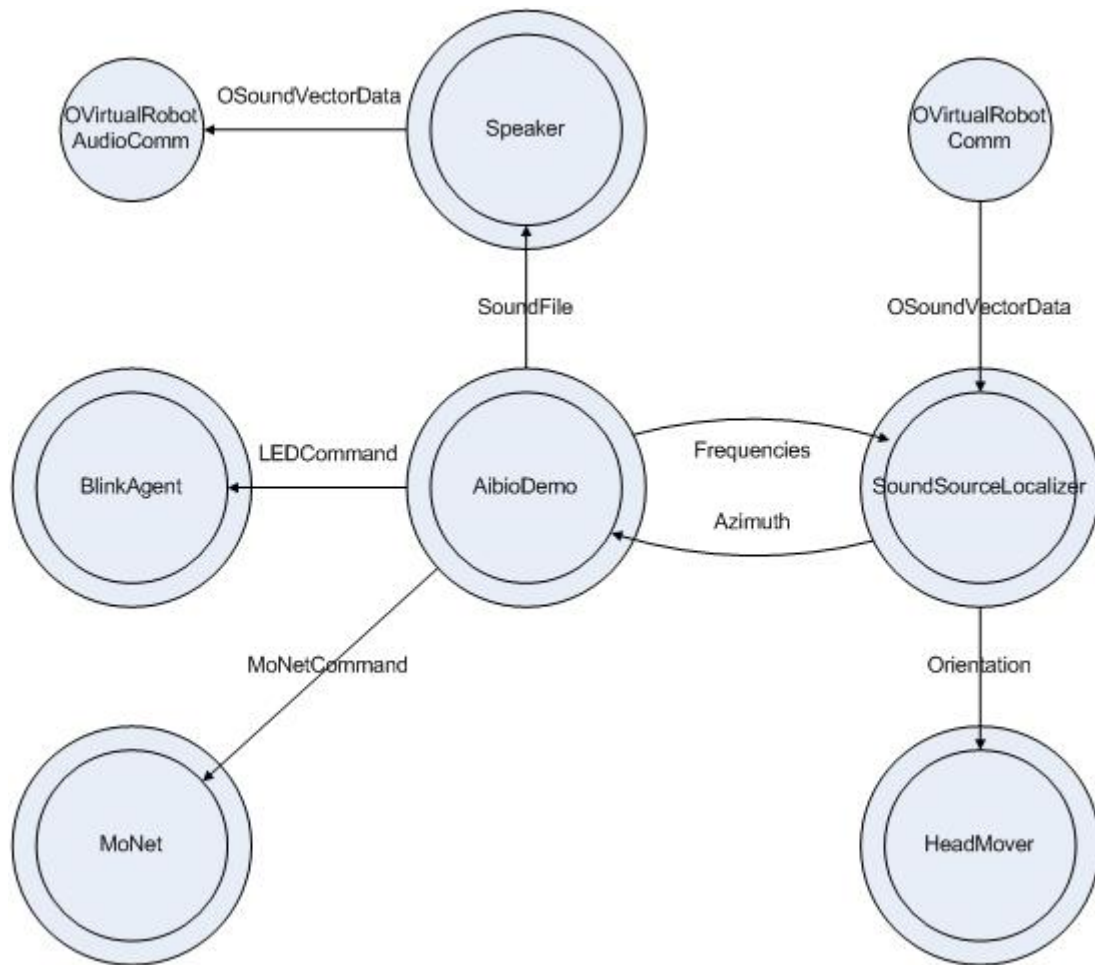


Figure 16: System model with SoundSourceLocalizer running in active mode

Testing showed that the new approach produced far better and more accurate results. Listening times are set to 160ms, equivalent to 10 sampling frames. To smooth the azimuth and reduce the effect of noise we also apply the following formula:

$$\theta_{n+1} = (1 - \alpha)\theta_n + \alpha \cdot \theta$$

where θ is the latest observed, deconfused azimuth, θ_n is the smoothed azimuth at the n -th sampling frame, and α is a constant for which $0 < \alpha < 1$. In our application, we have chosen $\alpha = 0.1$, which gives the last 10 sampling frames an influence of about 65% on the current value of the smoothed azimuth. A high value of α would lead to a very fluctuating azimuth, while a low value would produce a very slowly changing azimuth. By testing, it was determined that a value of 0.1 provides the best balance between noise reduction and a quick response.

8 PROJECT EVALUATION

The software was tested using PC speakers and a tonegenerator application. A different triangular modulated frequency would be generated on each channel simultaneously. The AIBO would then be set to determine the azimuth of one of the frequencies. Figure 17 shows an example of such a setup. After one azimuth has been determined, the AIBO would be set to determine the azimuth of the other frequency.



Figure 17: AIBO test setup

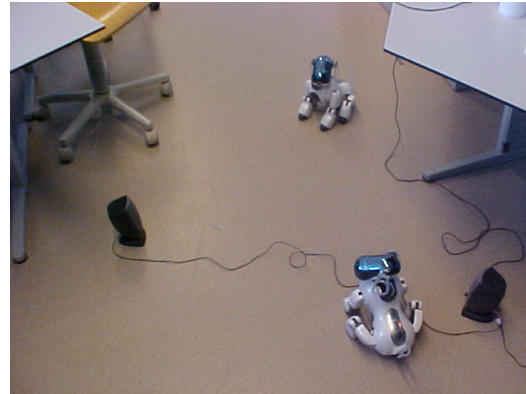


Figure 18: Multiple AIBO test setup

Figure 18 shows a test setup with multiple AIBO's and two speakers. The AIBO's both produced certain frequencies, which then had to be recognized by the other AIBO.

Deconfusion was tested by letting AIBO track a certain source. Then the sound source would be shut down and moved in the mirror position relative to the microphone axis. Without deconfusion, the azimuth of the new sound source would have been exactly the same and AIBO would continue to look in the old direction. With deconfusion however, AIBO found the new location of the sound source.

The performance of the software in all tests exceeded our expectations. Looking back to the requirements, we can safely say we fulfilled all of them. With our software the AIBO is able to:

- Determine the azimuth of a specified frequency or frequencies.
- Determine the azimuth of that frequency in noisy environment, within reasonable limits.
- Determine whether the azimuth is a front or back angle.
- Produce triangular modulated frequencies that can be detected by other AIBOs.

Our attempt to make one isolated Object that gets raw stereo sound data and returns azimuths has only partly succeeded. To prevent noise interfering with the sound we had to introduce an active mode to SoundSourceLocalizer in which it has control over the head movements via HeadMover. Although it doesn't work stand-alone in active mode it does yield significantly better results. Passive mode still is available and is automatically used if the HeadMover Object is not available.

8.1 RECOMMENDATIONS

Even with all the current features, there is certainly still room for improvement. The following are suggestions of improvements that can be made:

- Using IIR (Infinite Impulse Response) filters instead of FIR filters in the filter bank. IIR filters produce better bandpass filtering results than FIR filters but are harder to implement in software. Also with IIR filters the passband frequency can be much smaller, allowing for the recognition of more frequencies.
- Testing and improvement of the passive mode of SoundSourceLocalizer.

9 CONCLUSION

When we started this project we decided to use an iterative approach to the problem, following the V-process model (Figure 7). This approach was very useful to us, because it allowed us to use the knowledge that we gained during the whole process to improve certain parts of the system.

In the end, after numerous steps back and forth, we achieved an impressive result. The AIBO can sit listening to one particular frequency and move its head towards the sound source, or else indicate that the sound source is beyond the turning range of its head. Other sounds, nearby objects, positioning of the sound source behind the head or talking people did not seem to have a significant influence on AIBO's ability to localize the sound source.

Observing the performance of our software, we can conclude that sound source localization is very possible on an embedded environment such as an AIBO like the ERS-7. It has enough system resources to process the necessary operations and calculations fast enough to follow a sound source in real-time.

Sound source localization can be made fairly accurate and reliable to the point where two AIBOs can blindly find each other by using sound. The Dutch AIBO Team demonstrated this during the Robocup Tournament 2004, in a demo using this sound source localization software. They got 2nd place in the Open Challenge, pre-qualifying for Robocup 2005 in Japan!

10 SOURCES AND REFERENCES

- [1] www.eu.aibo.com, “AIBO Europe”.
- [2] www.openr.org
- [3] www.eclipse.org, “The Eclipse Project”.
- [4] www.cvshome.org, “Concurrent Versions System”.
- [5] www.fftw.org, “FFTW Home Page”.
- [6] <http://aibo.cs.uu.nl>, “Dutch AIBO Team 2004”
- [7] http://interface.cipic.ucdavis.edu/CIL_tutorial/3D_psych/azimuth.htm, “Azimuth Cues”.
- [8] <http://www.ie.ncsu.edu/kay/msf/sound.htm>, “Sound Localization”.
- [9] François Serra and Jean-Cristophe Baillie, “*AIBO Programming Using OPEN-R SDK*”, June 2003.
- [10] Rob Hartman, Ken Pohlmann, and Kevin Heber, *High Frequency Effects on Localization and Sound Perception in a Small Acoustic Space*, 2002.
- [11] James H. McClellan, Ronald W. Schafer, Mark A. Yoder, *DSP First. A Multimedia Approach*. Prentice Hall, 1998.
- [12] Bob Hughes and Mike Cotterell, *Software Project Management. Third Edition*. McGraw-Hill, 2002.
- [13] Kazuhiro Nakadai, Hiroshi G. Okuno, and Hiroaki Katano, *Real-time Sound Source Localization and Separation for Robot Audition*, 2002.
- [14] Kazuhiro Nakadai, Hiroshi G. Okuno, and Hiroaki Katano, *Auditory Fovea Based Speech Separation and Its Application to Dialog System*, 2002.
- [15] Keith D. Martin, *Estimating Azimuth and Elevation using Auditory Differences*, 1999.
- [16] http://en.wikipedia.org/wiki/Bandpass_filter, “Band-pass Filter”.
- [17] http://www.biols.susx.ac.uk/home/Chris_Darwin/Perception/Lecture_Notes/Hearing5/hearing5.html, “Binaural Hearing and Localization”.
- [18] <http://www.ee.surrey.ac.uk/Personal/D.Jefferies/powerac.html>, “Complex Numbers”.
- [19] http://www.circuit-magic.com/complex_numbers.htm, “Complex Numbers for AC Circuits Analysis”.
- [20] http://en.wikipedia.org/wiki/Discrete_Fourier_transform, “Discrete Fourier Transform”.
- [21] Kenzo Obata, Kentaro Noguchi, Yoshiaki Tadokoro, *A New Sound Source Localization Algorithm Based on a Formant Frequency for Sound Image Localization*, 2003.
- [22] <http://www.u-aizu.ac.jp/~j-huang/Localization/localization.html>, “Sound Localization in Reverberant Environments”.
- [23] <http://www.ie.ncsu.edu/kay/msf/sound.htm>, “Sound Localization”.
- [24] Jie Huang, Noboru Ohnishi, and Noboru Sugie, *A Biomimetic System for Localization and Separation of Multiple Sound Sources*, IEEE Transactions on Instrumentation and Measurement, vol. 44, nr. 3, June 1995.
- [25] Ehud Ben-Reuven and Yoram Singer, “*Discriminative Binaural Sound Localization*”.
- [26] Michael S. Brandstein, John E. Adcock, and Harvey F. Silverman, *A Practical Time-Delay Estimator for Localizing Speech Sources with a Microphone Array*, 1995.

- [27] Hiromishi Nakashima, Toshiharu Mukai, and Noboru Ohnishi, *Self-Organization of a Sound Source Localization Robot by Perceptual Cycle*, 2002.
- [28] Ea-Ee Jan and James Flanagan, *Sound Source Localization in Reverberant Environments using an Outlier Estimation Algorithm*, 1996.
- [29] Kazuhiro Nakadai, Daisuke Matsuura, Hiroshi G. Okuno, and Hiroaki Katano, *Applying Scattering Theory to Robot Auditioning System: Robust Sound Source Localization and Extraction*, Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robot and Systems, Las Vegas, Nevada, October 2003.
- [30] Tokuji Okada, Naoyuki Takahashi, and Hideyuki Sato, *Compact Sensor for Localizing a Sound Source Using Differences of Sound Levels Sensed by Condenser Microphones*, 2002.
- [31] Keith D. Martin, *Towards Automatic Sound Source Localization: Identifying Musical Instruments*, 1998.
- [32] Harald Viste, Gianpaolo Evangeliste, *On The Use of Spatial Cues to Improve Binaural Source Separation*, Proceedings of the 6th International Conference on Digital Audio Effects (DAFx-03), London, UK, September 2003.
- [33] Jean-Marc Valin, François Michaud, Jean Rouat, Dominic Létourneau, *Robust Sound Source Localization Using a Microphone Array on a Mobile Robot*.
- [34] Vikas C. Raykar, *Optimization Methods for Sound Source Localization Using Microphone Arrays*, CMSC 660 Project Solutions, December 2002.
- [35] <http://homepages.inf.ed.ac.uk/rbf/HIPR2/zeros.htm>, “Zero Crossing Detector”.
- [36] http://www.cs.cf.ac.uk/Dave/Vision_lecture/node29.html, “Second Order Zero Crossing Methods”.

11 APPENDIX A: PROGRAMMING GUIDELINES

Programmeer Guidelines voor Aibio

Code structuur:

-
- Include files hebben de extensie .h
 - h-files declareren eerst de public methoden en dan de private.
 - h-files declareren per access modifier (public/private) eerst constanten, dan variabelen, dan constructors en dan de methoden.
 - Source files hebben de extensie .cc
 - Maximaal 1 klasse definitie in een include file.

File naamgeving en plaatsing:

-
- <classname>.h en <classname>.cc (dus inclusief hoofdletters).
 - Alle files die nodig zijn per Core class Object in een aparte directory in /src. De naam van de directory dient hetzelfde te zijn als de naam van het Object.
 - Alle config files voor objecten (.ocf, stub.cfg) moeten in de /src/<Objectname> directory.
 - Alle files voor libraries die door meerdere objects gebruikt (kunnen) worden in de /src/common directory op volgende wijze:
 - * Library source files in de /src/common/lib<libname> directory.
 - * Library interface header files in de /src/common/include directory.
 - * Gecompilede libraries in /src/common/lib<libname> directory (samen met de sources dus). Zorg ook voor een makefile om de library te compileren, archiveren en ranlib-en.

File Header:

-
- Zet de onderstaande header aan het begin van alle nieuwe software files die je toevoegd aan het project. Vervang de comment signs door de juiste tekens voor die file. Header is onder voorbehoud.
- ```

/*
 * Copyright 2004 Aibio Project Group
 * Copyright 2004 Delft University of Technology
 *
 * Aibio Project Group is: Marlon Richert, Tijmen Roberti and Wouter de Vries
 *
 * Permission to use, copy, modify, and redistribute this software for
 * educational purposes is hereby granted.
 *
 * This software is provided "as is" without warranty of any kind,
 * either expressed or implied, including but not limited to the
 * implied warranties of fitness for a particular purpose.
 */

```

Algemene code eisen:

- 
- Maak bij if-statements ALTIJD gebruik van accolades {}.
  - Het weglaten van {} speelt latere fouten in de hand.
  - Dus NIET zo:
 

```

 if (condition) //FOUT, Tijmen!! GEEN ACCOLADES {}!!
 STATEMENT;

```

Maar ZO:

```

if (condition)
{
 STATEMENT;
}

```

- Klassenamen beginnen met een Hoofdletter.
- Functienamen beginnen met een kleine letter (dus niet volgens die vage Sony manier).
- variabelen beginnen ook met een kleine letter.
- Gebruik GEEN underscores maar wel hoofdletters:
  - bijv: mijnMooieVariabele ipv mijn\_mooie\_variabele
- constanten in FULL\_CAPS.

- verzin ZINNIGE en logische variabelenamen!! Dus geen bla1, temp1, temp2 etc. De variabelenaam moet het gebruik van de variabele zo goed mogelijk verklaren.
- Idem voor functienamen. Er mag best wat tijd besteed worden aan het bedenken van een goede functienaam.
- Schrijf niet teveel inline code.
- Gebruik geen afkortingen die niet algemeen bekend/geaccepteerd zijn.
- Vermijd ambiguïteit.

## Debugging:

- Gebruik voor debugging de OSYSDEBUG macro van OPEN-R, wanneer we eindelijk de wireless console tot onze beschikking hebben! Gebruik tot die tijd de eigen TOLOG macro op dezelfde manier als OSYSDEBUG. TOLOG schrijft per default naar de file /MS/OPEN-R/DEBUG.LOG.
- Om deze macro te kunnen gebruiken moet je <log.h> includen.
- Je moet in je Makefile ook de Aibio library meelinken. Dus voeg aan je LIBS toe:

```
-L../common/libAibio -lAibio
```

- Begin elke debugmessage met "<klasse-naam>::<methodenaam>() : ". Gebruik van de % operator is mogelijk. Dus bijvoorbeeld:

```
TOLOG("MyClass::myBuggyMethod() : state = %d\n pointer = %x\n", state, ptr);
```

- Gebruik voor run-time debuggen het BlinkAgent OObject. Zie voor een handleiding Debugging with BlinkAgent.txt in de aibio/docs folder.

## Comments:

- Alle comments in het Engels.
- Alle top-level comments graag in mooie volzinnen.
- Een /\* \*/ comment refereert altijd aan het GEHELE volgende codeblock.
- Een // comment refereert ALLEEN aan de volgende zin.
- Een // comment mag ook achter de zin in kwestie staan, na een of meer tabs.
- Iedere klasse begint met een comment waarin uitgelegd wordt hoe die klasse heet en wat hij doet:

```
/* A FirstClass is a thing that does incredible things.
 * Some might say it is destined for greatness.
 * Others beg to differ.
 */
class FirstClass
{
 //I'm awesome!!
}
```

- Zet comments er DIRECT bij als de code af is (en werkt uiteraard) en niet later want dat gebeurt meestal niet. :)
- Alle niet standaard functies beginnen met een comment waarin uitgelegd wordt wat die functie doet op de volgende wijze:

```
/* Does amazing things because we want it too.
 * Has some embarrassing side effects that we didn't want,
 * but they're here anyway so deal with it!!
 */
<return type> <class name>::<function name>(<args>)
{
 //...
}
```

- Voeg comments toe bij potentieel onduidelijk stukken code.
- Vertel ook WAAROM je iets doet.
- Bij het committen naar de CVS ook comments toevoegen aub.

## Code style:

- Wrap alle lines (netjes!) op 80 characters.
- Als je line-wrapt, indenteer per default met twee tabs (en niet met één), om goed onderscheid te maken met block indents.
- Functies/constructors definitie:

```
<return type> <method name>(<args>)
{
```

```
 //insert code here
 }
```

- Zet, bij het line-wrappen van een functie met veel argumenten, de argumenten onder elkaar. Bijvoorbeeld:

```
int mijnIngewikkeldeFunctie(int myInt,
 int* myIntPtr,
 double myDouble,
 double* myDoublePointer);
```

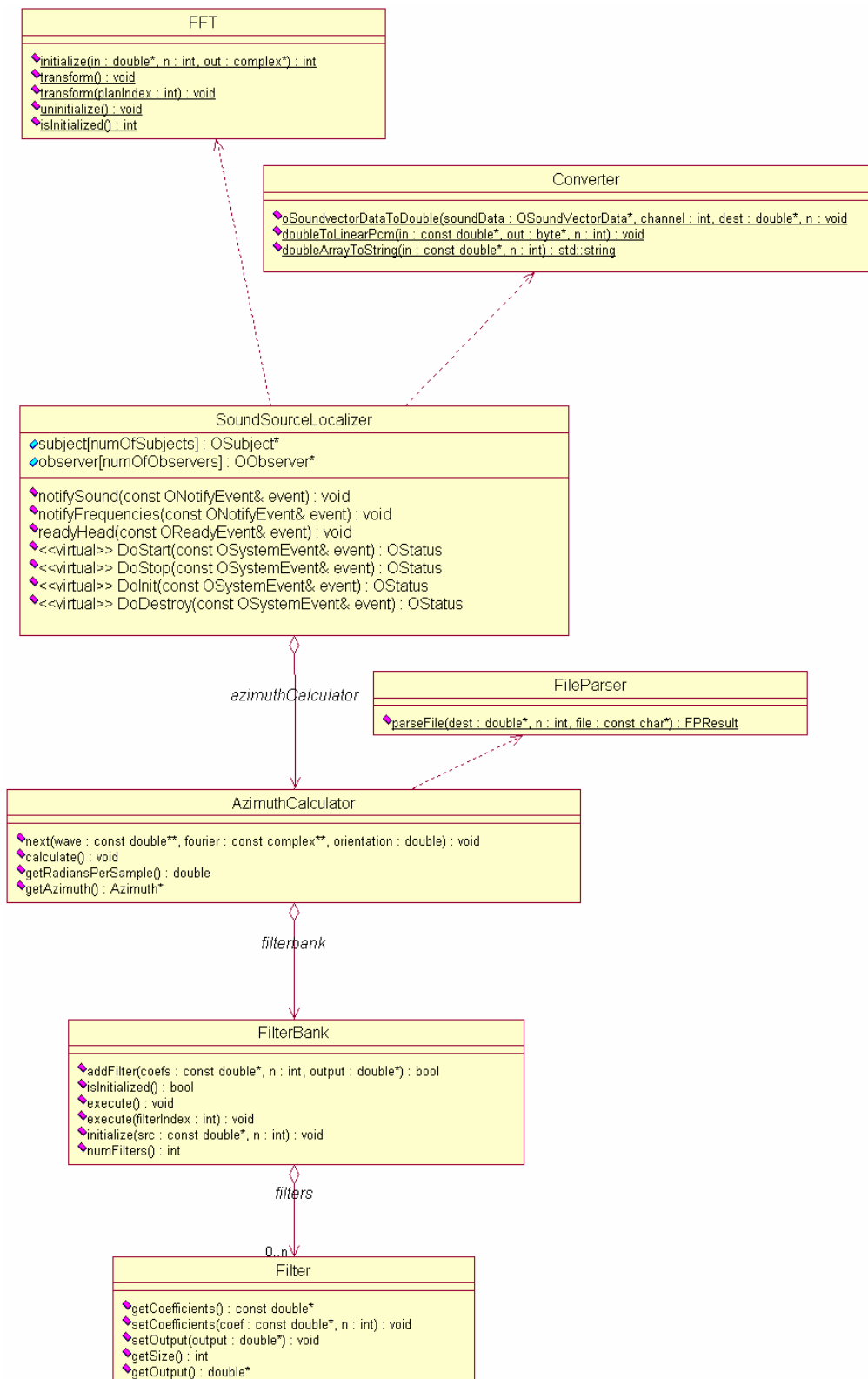
- Bij functies met als argumenten input en output array pointers gebruik de volgende volgorde :
  - 1) source array
  - 2) parameters die iets over source zeggen
  - 3) overige parameters
  - 4) destination array

bijv:

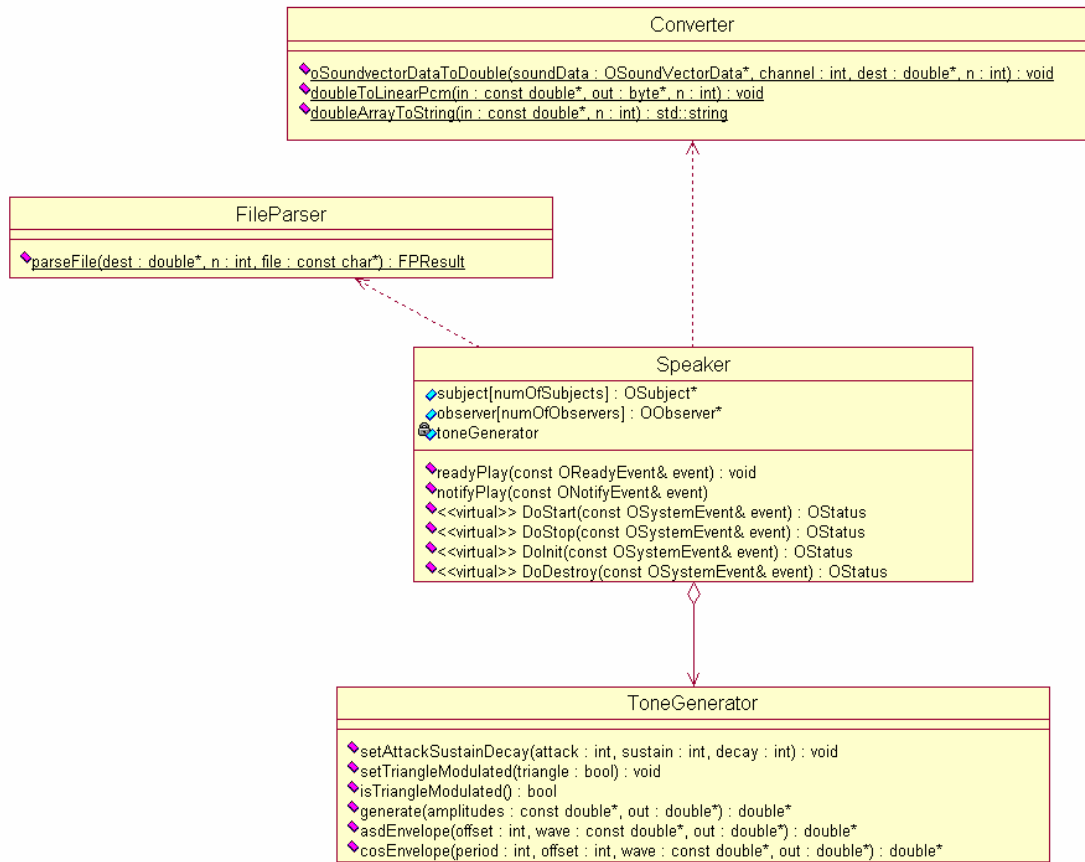
```
copyArrayTo(double* source, int length, int mask, double* destination);
output(double* source, int amount);
```

## 12 APPENDIX B: CLASS DIAGRAMS

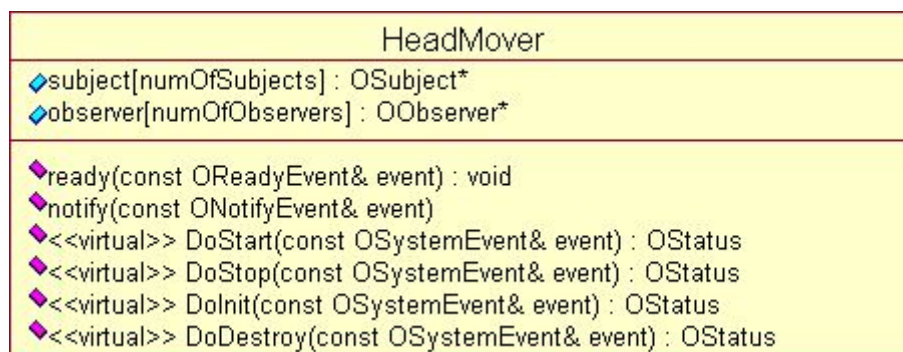
### 12.1 SOUND SOURCE LOCALIZER



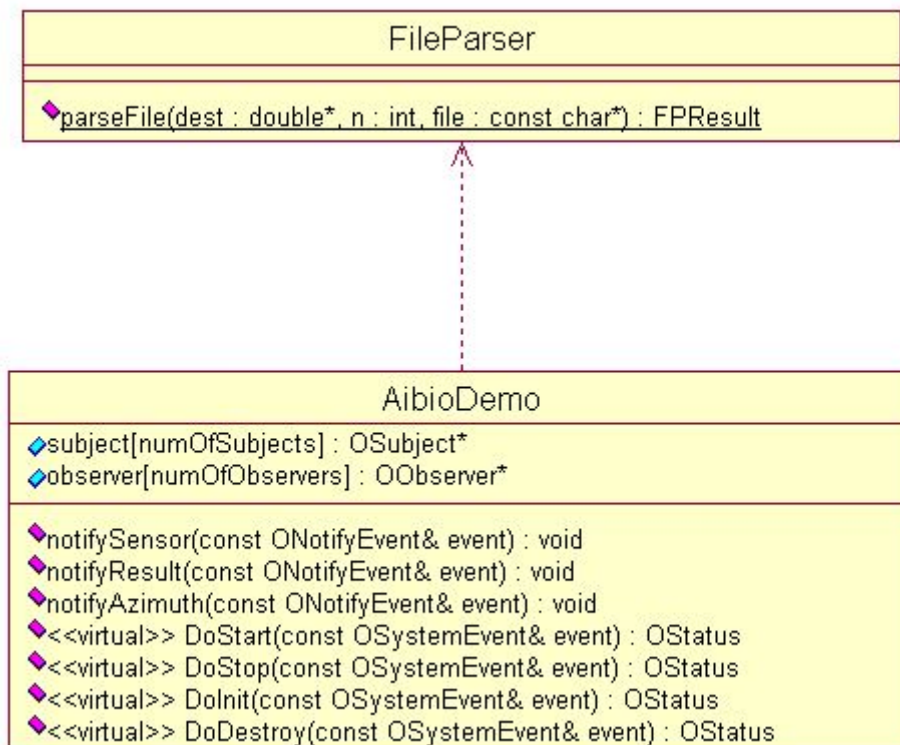
## 12.2 SPEAKER



## 12.3 HEADMOVER



## 12.4 AIBIODEMO



## 12.5 BLINKAGENT

