Master's thesis
MEE0126

Design and Implementation of an Acoustical Transmission Protocol

David Erman, david.erman@bth.se
22nd February 2002

This thesis is presented as part of the
Degree of Master of Science in Electrical Engineering with emphasis on
Telecommunications/Signal Processing.

Blekinge Institute of Technology

Magisterprogrammet
Blekinge Tekniska Högskola
Institutionen för Telekommunikation och Signalbehandling
Examinator: Benny Lövström, Stefan J. Johansson
Handledare: Stefan J. Johansson, Benny Lövström

Abstract

The RoboCup Sony Legged Robot League is an initiative to promote robotics technologies and artificial intelligence in the form of a soccer competition between four-legged robots. The Blekinge Institute of Technology, Royal Institute of Technology, the Universities of Örebro and Umeå, competing in the RoboCup domain as “Team Sweden”, have been participants in the league for three years. To improve the chances of victory in the league, a way to communicate important data between robots is desired. This thesis explores methods for implementing this communication using only the built-in hardware of the robots, i.e. one speaker and two microphones.

Contents

List of Figures

List of Tables

Chapter 1

Preliminaries

This chapter gives a short introduction to the thesis and the time plan for the thesis.

1.1 Introduction

This thesis is the result of a collaboration between the department of Telecommunications and Signal Processing and the department of Software Engineering and Computer Science at the Blekinge Institute of Technology.

It concerns the transmission of digital data using the Sony AIBO entertainment robots, and the implementation of software to facilitate this as part of a Swedish team effort, collectively known as TEAM SWEDEN.

The final functional test was performed during the ROBOCUP world cup in Seattle, USA.

1.2 Problem formulation

What amount of data, given the constraints of the hardware in the AIBO robot, is possible to transmit between two robots under:

- Non-ideal conditions, transmission may be interfered with by either or both of the robots moving or by other background noise such as other robots and audience-generated noise.
- Ideal conditions, i.e. no background noise and with none of the robots moving.

1.3 Time plan

This thesis is for 20 credits, i.e. 20 weeks of 40 hours each, amounting to a total of 800 hours. The dates and times below are intended as approximate guidelines.

1.3.1 Weekly plan

Week 12	Preliminary study
Weeks 13-15	Study of the development environment and hardware, includes the ROBOCUP Summer Camp in Paris on April 10-15. Gather data from robots to be analysed in Matlab.
Weeks 16-20	Protocol– and filter-design. Analysis of the previously gathered data. Supplemental measurements. Preliminary, non-optimised, implementation. Filter algorithms.
Weeks 21-28	Implementation. Possible re-evaluation of chosen algorithms. Optimisation of algorithms and code.
Weeks 29-32	Report and follow–up

1.3.2 Deadlines & Other Dates

2001-03-19 Project commencement

2001-04-10 ROBOCUP Summer Camp, Paris

2001-06-08 ROBOCUP German Open

2001-08-01 ROBOCUP 2001 Seattle

Chapter 2

Introduction to RoboCup and Team Sweden

This chapter describes what ROBOCUP is and, more specifically, the Sony Legged Robot League. It also gives a short introduction to the approach TEAM SWEDEN have used in the league, and a description of the hard– and software of the AIBO robot.

2.1 RoboCup

The Robot World Cup Initiative (ROBOCUP) is an international research and education project, which aims to foster and promote AI and intelligent robotics research. By organising *RoboCup: The Robot World Cup Soccer Games and Conferences* [2], ROBOCUP creates an environment suitable for incorporating multiple technologies such as design of autonomous agents, multi-agent cooperation, high-level strategy decision-making, real-time reasoning and robotics. There are other aspects of the ROBOCUP Initiative, including technical conferences and various educational programmes, but the main focus is the integration of effort and research in the Robot World cup.

The slogan and primary goal of ROBOCUP is:
By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champions.

RoboCup currently consists of six leagues:

Simulation League A software only competition where programmed agents play on a virtual playing field. Not entirely unlike a computer soccer game, albeit with more sophisticated AI capabilities.

Small Robot League 100x100x100 mm sized robots, manufactured and programmed by the contestants. One controlling computer per team and a central overhead cam-

era are used for strategy decisions and localisation, lessening the computational load on the robots.

Middle Size League Larger, vacuum-cleaner sized robots. Much the same as the small size league, with the exception of having removed the central camera. The larger robots have on-board cameras for localisation. This adds the interesting aspect of image processing and identification to the league.

Sony Legged Robot League Fully autonomous robots, navigating by the aid of a range detector and camera. Described further in section 2.1.1

RoboCup Rescue A project for promoting research and development in the domain of disaster rescue. Consists of a simulated league and a league with real robots.

RoboCup Junior A project-oriented educational initiative that sponsors local, regional and international robotic events for young students. It is designed to introduce RoboCup to primary and secondary school children, as well as undergraduates who do not have the resources to get involved in the senior leagues. The focus in the Junior league is on education.

In addition to the leagues mentioned above, the Humanoid League will be part of the competitions starting 2002.

2.1.1 The Sony Legged Robot League

The Sony Legged Robot League (SLRL) uses the Sony AIBO entertainment robots and is the only ROBOCUP league involving hardware that does not include construction of the robot. Modification of the robots is also prohibited. This makes the league more interesting from a software and sensory/perceptual viewpoint, since all teams have a common starting point in the robot hardware limitations.

The competitions are divided into two parts: the soccer matches and the challenges. The matches are played three-on-three with one team wearing blue stickers and the opposing team wearing red, on a 1800x2800mm field. [16] Around the field is placed six landmarks of predetermined colours at fixed positions for use in localisation of the robots.

The challenges are separate tasks to be performed to illustrate the performance of the software of each team. Previous tasks have included localisation, collaboration and simple goal-scoring. For the Seattle 2001 World Cup, the challenges were:

Localisation: The first of the challenges tested the ability of the robots to localise and position themselves on five different markers on the fields. The positions of the markers were known in advance, leaving it up to the software to place the robot on the markers in the shortest amount of time possible.

Collaboration: Last year featured the introduction of a simple one-pass collaboration challenge. This year, this was further expanded to include: passing more than once between teammates, opponent avoidance and goal scoring in a single challenge.

Goalie: For the goalie challenge, the goalie was placed on mid-field and was given thirty seconds to return to the defending goal. Once back in the goal, it was to prevent a ball, launched from a small ramp placed up-field, from entering the goal.

2.2 The Sony AIBO Robot

Developed and built by Sony, the AIBO family of entertainment robots is comprised of three models: the ERS–110, ERS–210 and the new ERS–310-series [1]. The ERS–110 was the first consumer model sold by Sony and the version used in the 1999 and 2000 ROBOCUP tournaments. In late 2000, the ERS–210 was released, with improved CPU and updated operating system. At the time of writing the latest models, the ERS-311/312 have only just been released. [1]

(a) ERS-110 (b) ERS-210 (c) ERS-310

Figure 2.1: The AIBO family

2.2.1 Hardware

The hardware platform of the AIBO robot is a challenging and exciting one. The robot is equipped with several sensors, including – but not limited to – several pressure sensors, a CMOS-based camera, microphones, acceleration sensor, temperature sensor and vibration sensor.

AIBO ERS–210 is modularly built, with all four legs and head easily removable for replacement. The robot can be equipped with a wireless LAN card for debugging. Unfortunately, most of the details of the hardware is classified and available only under a non disclosure agreement

[1] Yet another model was released in early December, 2001.

with Sony. However, some information is publically available: the main CPU is a 200 MHz MIPS, memory capacity is 16 Mb ROM, 32 Mb on-board DRAM and a 4 Mb flash-ROM.

Applications are stored on Sony Memory Stick cards; a PCMCIA slot is located in the innards of the robot for a wireless LAN card. The robot has in total 20 degrees of freedom (DOF), i.e. in total 20 movable joints. Each leg and the head has 3 DOF, the tail two, with the mouth and ears having only one.

2.2.2 Software

The software on the AIBO robots is comprised of a general robot hardware API, OPEN-R [17], [12]. OPEN-R was developed by Sony and released in 1998 to provide a standard API for entertainment robots. The architecture is modular both with respects to software design and the ability to handle different hardware components through the same API.

Designed for generality, the main features of OPEN-R are: the ability to dynamically handle varying hardware configurations without rebuilding the application; interchangeable modular software for easily changing behaviours; plug-and-play connectivity for the hardware modules – each module can notify the system of its capabilities and let OPEN-R react to this.

OPEN-R is built on Sony's object oriented real-time operating system, Aperios, and is programmed using either the 'C' or 'C++' languages.

2.3 Team Sweden

TEAM SWEDEN was formed for the 1999 ROBOCUP World Championships in Stockholm, Sweden as a national collaborative effort. The team consisted of participants from several Swedish universities, [3] most notably the Örebro University (OrU), Royal Institute of Technology (KTH) and the Blekinge Institue of Technology (BTH). KTH has since been phased out, and in 2001 Umeå University (UmU) joined the team.

For 2001, the work distribution was as follows:

UmU Walking styles and joint work on the Reactive Planner (RP). Due to the change of robot from the ERS–110 to the ERS–210, completely new walking styles were implemented.

BTH Communication and joint work on the RP.

OrU Main site; coordination and the rest of the modules. Also responsible for the development of the Graphical Development Platform (GDP) debugging and development platform.

A table of the 2001 TEAM SWEDEN members can be found in appendix B on page 43.

2.3.1 The Team Sweden approach

The TEAM SWEDEN approach is based on the 'Thinking Cap' fuzzy logic system for autonomous robot navigation [13]. The TEAM SWEDEN implementation is a three-layered architecture consisting of an upper, middle and lower layer, as shown in figure 2.2. The upper and middle layers both consist of two modules each, whereas the lower layer is a single module. The modules are: the RP and Global Map (GM) in the upper layer, the Hierarchical Behaviour Module (HBM) and Perceptual Anchoring Module (PAM) in the middle layer, with the lower layer being only the Commander (CMD).

The transmission module, or Sound Module (SND), developed in this project does not fall into any of the layers, but is rather external to the system; it is more or less a bolt-on accessory for added functionality.

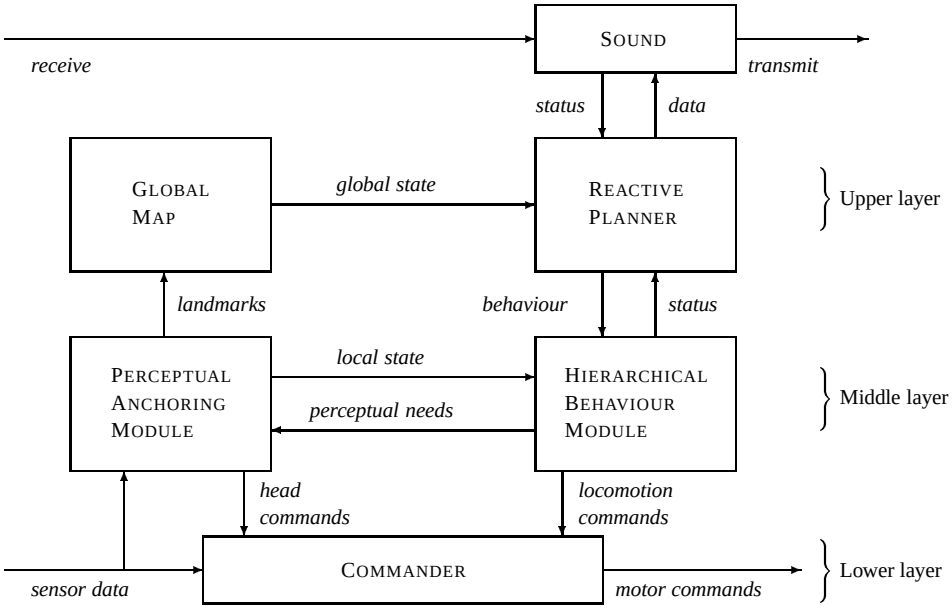


Figure 2.2: The TEAM SWEDEN architecture

GM The GM module is responsible for maintaining a global map of the playing field. This map is updated from the perceptual information received from the PAM module, and is retained in the GM module to build a more robust description of the robots surroundings than is available from the transient information from the PAM.

PAM The PAM uses perceptual information from the robot to create a snapshot of all the objects in the immediate surroundings of the robot. This information is then passed onto the GM, and placed in a *local perceptual space* (LPS) used both by the RP and GM.

RP The second upper layer module makes the real-time strategic decisions, selecting appropriate higher-level behaviours such as OBSTRUCT, GOHOME and SEARCH-BALL. The HBM is then notified of the selection.

The RP is based on the Electric Field Approach (EFA), and a more complete treatise is available in [9]

HBM The HBM is responsible for translating the higher level behaviours from the RP into low level behaviours, or movement commands for the CMD.

CMD The Commander is the Hardware Abstraction Layer (HAL) of the TEAM SWEDEN architecture. It interfaces both the PAM and HBM; the PAM sends commands for scanning for objects and the HBM movement commands. These commands are then converted into motor commands for the robot.

One integral part of the CMD is the implementation of walking styles and kicks. A good walking style can make or break a team, no matter how good the higher level layers are. A bad walking style makes odometry more or less impossible, which in turn makes localisation more uncertain.

SND This is the module developed for this thesis. It implements inter-robot communication by using audio sequences. The SND module is only interfaced to the RP and works closely in conjunction with it. It is described further in chapter 4 on page 20.

In addition to the software that executes on the AIBO, the Graphical Development Platform (GDP) was developed this year to provide a graphical debugging tool for the higher level modules. The GDP can be physically connected to the robots to display localisation and other types of information. When not connected to a robot, the GDP uses either a sensory simulator or previously recorded sensory data from a robot.

The GDP is a valuable tool for evaluating and implementing new behaviours in the RP, since the same source code for the RP is used on both the robots and in the simulator. A complete treatise on the GDP is given in [10].

Chapter 3

Digital Transmission

This chapter gives a very short introduction to the subject of digital transmission; it is by no means intended as a complete treatise, but rather a brief primer. For more elaborate discussions, see Halsall [7] and Haykin [8].

3.1 Digital Transmission Basics

The transmission of digital data over an analogue channel has been the subject of much research. Data is usually carried over the channel in the form of an electrical current in a conductive material, such as a cable or printed circuit board. However, when a cable for some reason or other is not a viable option, such as in sonar applications, using an acoustic channel might be the only available alternative. An acoustic channel utilises pressure waves across some medium, such as water, the ground or air, as opposed to electrical currents travelling in a conductor. The figure below depicts a generic single-channel digital communications system.

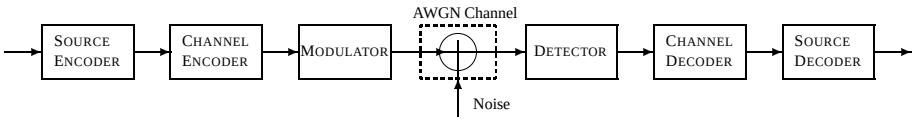


Figure 3.1: A generic digital communication system

The bandwidth of an acoustic channel carried by sound pressure is much lower than its electrical counterpart, due to the facts that air acts a natural low-pass filter, and that the transducers (i.e. speakers and microphones) are very expensive to manufacture for higher frequencies. The relationship between the maximum bit-rate, measured in *bps*, and bandwidth, assuming the channel is noiseless, is given by:

$$C = 2W \log_2 M \tag{3.1}$$

where W is the bandwidth of the channel, measured in Hz and M is the number of levels per signalling element. Assuming a speaker/microphone setup has the bandwidth 18 kHz, with each signalling element representing a single bit, the gross bit rate would be:

$$2 \cdot 18000 \cdot \log_2 2 = 36000\text{bps}$$

A simplified expression for binary transmission, i.e. with only two values possible per transmitted symbol ($M = 2$) is:

$$C = 2W$$

since $\log_2 2 = 1$. This can be interpreted as two bits per period of the carrier frequency.

However, there is no such thing as a noiseless channel; and in particular: not an acoustic noiseless channel. Acoustic noise is more tangible than e.g. thermal noise induced in electrical conduits. The influence of noise to a signal is usually referred to be the *signal-to-noise ratio (SNR)*, normally expressed in decibels. It is a measure of how much power the signal in a signal has (S), in relation to the power of the noise (N), and is defined by:

$$SNR = 10 \log_{10} \left(\frac{S}{N} \right) \tag{3.2}$$

The maximum data rate of a transmission channel is clearly related to the SNR, and is described by the expression:

$$C = W \log_2 \left(1 + \frac{S}{N} \right) \tag{3.3}$$

where C is the data rate, W the bandwidth of the channel, S and N as above. This equation is known as the *Shannon-Hartley law*. Thus, if we assume that the 36000 bps link discussed earlier has a S/N ratio of 0.5 — i.e. the total power of the signal is half that of the line noise — we get:$18000 \cdot \log_2(1 + 0.5) \approx 10500\,bps$

Note that this is the maximum *theoretical* data rate, and that signal attenuation in the transmission medium is not taken into account.

As described in section 2.2.1 on page 9, the only forms of transducers[1] suitable for use are the two microphones and speaker on the AIBO robot; others, such as the LEDs on the head, could theoretically be used, but would require an entirely different approach than the ones explored in this project.

3.2 Discrete Sources and Source Coding

A *discrete source* is a source of information that can be modelled as a discrete random variable S, with the possible symbols taken from the fixed finite *alphabet* Φ.

$$\Phi = \{s_0, s_1, \ldots, s_{K-1}\}$$

[1] A device for converting sound, temperature, pressure, light or other signals to or from an electronic signal.

with the symbol probabilities

$$P(S = s_k) = p_k \qquad k = 0, 1, \ldots, K-1$$

Source encoding is the process of selecting replacements for code words in the input that decrease the redundancy in the output. The common method is to replace the symbols more likely to appear in the input with shorter codes, whilst reserving the somewhat longer code words for the more uncommon symbols. For instance, the letter 'e' is more common than the letter 'q' in the Latin alphabet, and would thus be represented by a shorter code word. One example of this type of code is the Morse code, and another common coding scheme is the *Huffman* code, which can be shown to be optimal in the sense that it generates the shortest average code word length.

The collective name for this type of code is *variable-length codes*. To efficiently encode the discrete source, we thus need to know some statistics of the source.

The amount of information gained for each observation of S is given by the expression

$$I(s_k) = \log_2\left(\frac{1}{p_k}\right) \tag{3.4}$$

The *average* amount of information content per symbol, or *entropy*, for the alphabet $H(\Phi)$ is given by the estimate of $I(s_k)$:

$$H(\Phi) = E[I(s_k)] = \sum_{k=0}^{K-1} p_k I(s_k) = \sum_{k=0}^{K-1} p_k \log_2\left(\frac{1}{p_k}\right) \tag{3.5}$$

Assuming that the probabilities are equal, i.e.

$$P(S = s_k) = \frac{1}{K} \qquad k = 0, 1, \ldots, K-1$$

all symbols are equally probable to appear in the input, which yields

$$\begin{aligned} I(s_k) &= \log_2\left(\frac{1}{K}\right) \\ &\text{and} \\ H(\Phi) &= \sum_{k=0}^{K-1} \frac{1}{K} \log_2\left(\frac{1}{K}\right) \\ &= \log_2\left(\frac{1}{K}\right) \end{aligned} \tag{3.6}$$

Since there were no statistical data available about which messages – nor which messages to actually send, for that matter – that were to be transmitted between the robots, equal symbol probabilities was assumed, and a direct mapping of binary values was used.

3.3 Channel Coding

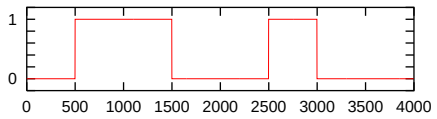
Whereas source coding uses statistical properties of the input to create alternate, smaller, symbol alphabets, channel coding is the process of effectively creating a set of binary transitions in the output so as to minimise the possibility of loss of synchronisation and/or data during transmission.

For protecting against bit errors in the binary stream, error control may be used. Common ways of providing such error control are Cyclic Redundancy Check (CRC), redundant bits and parity bits.

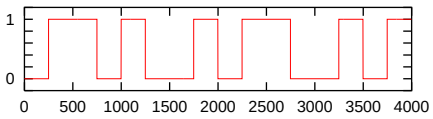
Synchronisation may be obtained by one of several methods; some common common ones are: Manchester, High Density Bi-Polar 3 (HDB3) and Alternate Mark Inversion (AMI). The common denominator with all synchronisation schemes is that they effectively decrease the maximum bitrate in half. This is due to the nessecity for transition extraction in the signal, as shown in figure 3.2

The transitions in the signal makes it possible to extract a synchronisation clock pulse from each bit-time. However, it also requires two discrete levels to be transmitted per bit, thus halving the effective bitrate.

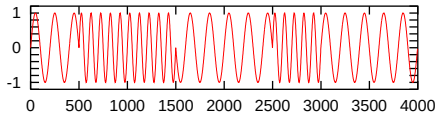
By introducing these extra transitions in the signal, the frequency characteristics are also changed.



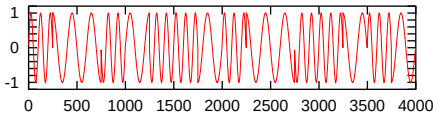
(a) Binary on–off signal



(b) Manchester encoded version of 3.2(a)



(c) FSK version of 3.2(a)



(d) FSK version of 3.2(b)

Figure 3.2: Manchester encoding

3.4 Modulation

Modulation is the process of creating a sequence of digital numbers suitable for feeding to a Digital-to-Analog Converter (DAC) from the binary sequences created previously. The DAC may be connected to some form of transducer such as a speaker and the modulation process is then known as Pulse Code Modulation (PCM).

This section discusses three common methods for generating a PCM sequence: Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK) and Phase Shift Keying (PSK). Common for all three is the notion of a *carrier frequency*. The carrier frequency (carrier for short) is the base frequency for the sine function used in the creation of the PCM sequences. It is denoted below by:

f_{c1} carrier for binary 1

f_{c0} carrier for binary 0

The binary signal we wish to modulate is denoted by c.

The term *shift keying* derives from the fact that each signalling element represents a level shift from some previous state. Though we only discuss binary signals in this section, it is common to use more levels per signalling element, i.e. ternary and quaternary modulation levels. In fact, the safe-mode transmission protocol developed for the thesis employs a 13-level FSK modulation scheme.

3.4.1 Amplitude Shift Keying

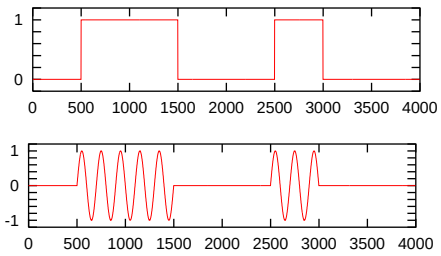


Figure 3.3: Amplitude shift keying

ASK is the oldest and simplest form of PCM modulation. An ASK sequence is generated by multiplying the binary stream with the carrier.

$$v_{ASK} = c \sin(f_{c1})$$

ASK is very sensitive to noise in the channel, since only the positive bits are actually carrying energy across the channel. The acoustic equivalent of the zero bits is silence.

3.4.2 Frequency Shift Keying

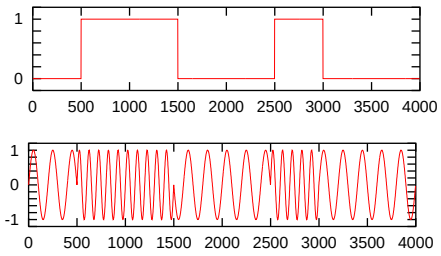


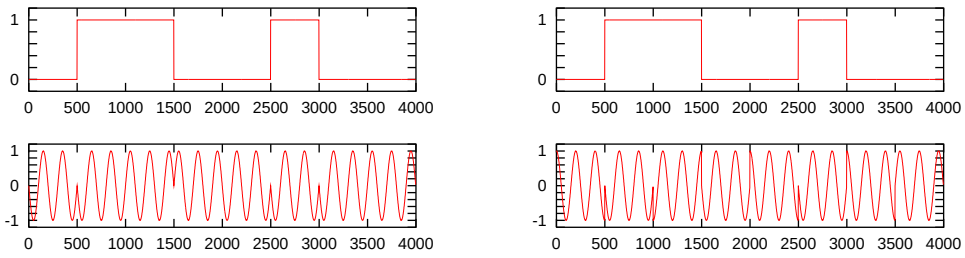
Figure 3.4: Frequency shift keying

FSK uses one distinct frequency per signalling element, with constant amplitude, thus carrying energy in the medium at all times. Each shift level uses its own frequency. This makes FSK signal occupy a larger portion of the available bandwidth than other modulation methods.

An FSK sequence is generated by:

$$v_{FSK} = c\sin(f_{c1}) + (1-c)\sin(f_{c0})$$

3.4.3 Phase Shift Keying



(a) Phase coherent PSK

(b) Differential PSK

Figure 3.5: Phase shift keying

PSK signals use both constant frequency and amplitude, denoting shifts by varying phase. This has the advantage of letting each signalling element carry equal amounts of energy in the signal. This makes it less susceptible to both channel noise and frequency-selective influences.

The phase shifts can be either relative or absolute, and are then called *differential* and *phase coherent* PSK, respectively.

The phase coherent sequence is given by:

$$v_{FSK} = c\sin(f_{c1}) + (1-c)\sin(f_{c1}+\phi)\text{, where } \phi \text{ is the phase shift in radians.}$$

and for differential PSK:

$$v_{FSK} = c\sin(f_{c1} + \gamma_k) + (1-c)\sin(f_{c1} + \phi_k) \quad k = 1, 2 \ldots n$$

where ϕ and γ are varying phase shifts in radians. Each signalling elements implies a phase shift change, whereas the phase coherent sequence does not.

Chapter 4

The Sound Module

This chapter discusses the SND module of the TEAM SWEDEN architecture. The SND module is the main subject of this thesis and each aspect of it is treated in some detail. Two separate protocols were designed and implemented: one high-speed 16 bps protocol and a safe-mode protocol. The safe-mode protocol was provided to ensure some form of reliable, albeit slow, transmission, whereas the high-speed protocol was used for testing what transmission rates could actually be achieved using an acoustic channel with the hardware provided by the AIBO robots.

The high-speed protocol was implemented using 2-PSK encoding with no error checking or control. It was designed for very short distances between transmitter and receiver, with a fixed message size of 16 bits.

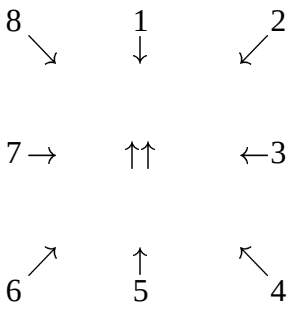
4.1 Preliminary work

Before the actual work of coding the SND module, a program for playing and recording audio with the robots was implemented. The software developed there laid the foundation for much of the module itself, and was used to take measurements and test algorithms.

Without this preliminary piece of software, choosing suitable carrier frequencies would have been more difficult. A DAT-recorder would have offered more possibilities with respect to sample rates and usability, but this approach was chosen to get a more accurate view of the data actually received by the robots.

The first use for the recording program was to try and estimate the combined transfer function of the speaker of the transmitting robot, the space between both robots, and the microphone of the receiving robot. This was done by placing one robot, the playing robot, facing in one direction, and a second robot, the recording robot, facing the first in eight positions and at five distances at each position. The positions were as depicted below. The double arrows indicate the position and direction of the playing robot, and the single arrows represent the recording robot.

By generating and transmitting a PN-sequence on the center robot, and recording this noise



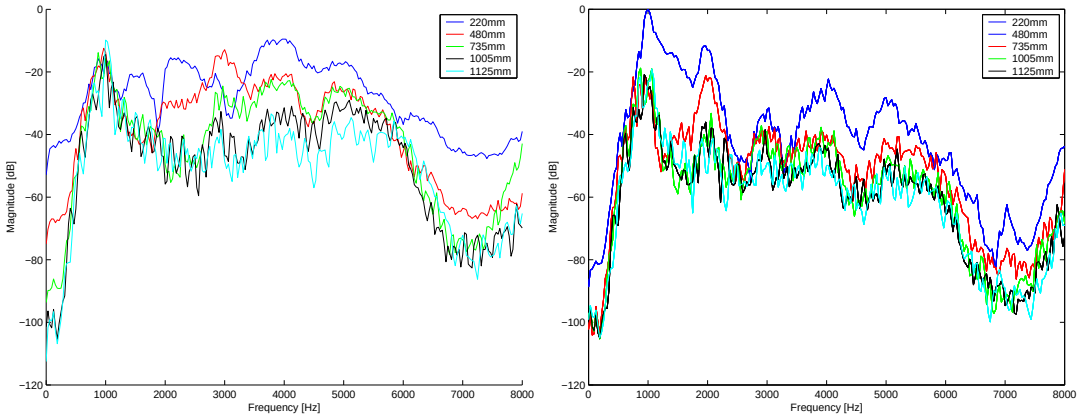
(a) Recording positions

1. 220mm
2. 480mm
3. 735mm
4. 1005mm
5. 1125mm

(b) Recording distances

Figure 4.1: Robot recording positions

on the second robot an estimation of the transfer function between the two robots can be made. This estimation is created by calculating the Power Spectral Density (PSD) of the received signal on the second robot. Figure 4.2 show the PSD of two robot placements.



(a) Position 1

(b) Position 5

Figure 4.2: PSD of positions 1 and 5

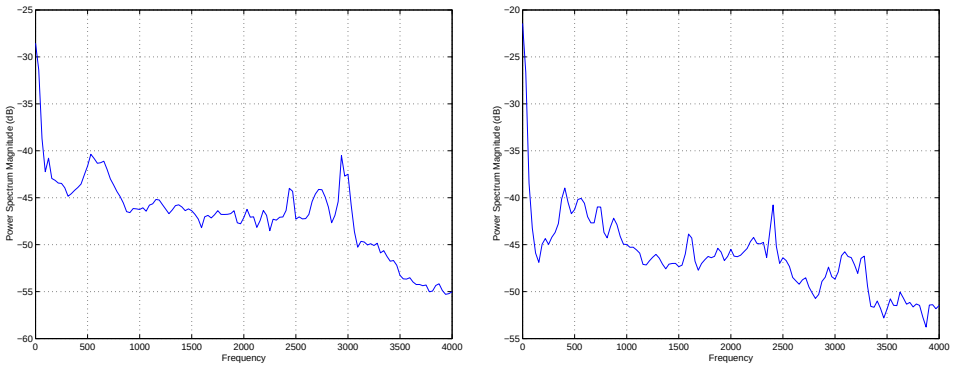
At shorter distances the PSDs are not very similar, being indicative of the influence of the positions and direction of the robots affecting the appearance of the transfer. However, as the distances increase, the PSDs become more similar as the sound reflects off the walls in the room where the recordings were made. The room was fairly small and also filled with furniture, creating a rather complex structure, with several paths that the sound could take. This would not be a problem during use of the sound module, as the matches are played in largely open areas.

4.1.1 Carrier selection

When selecting suitable carrier frequencies, there were two major issues to consider:

1. Background noise. Noise generated by audience, other robots, cameras and such.
2. Robot noise. Noise generated by the robot itself.

Measurements of both were recorded and analysed. By calculating the PSD of recorded data from several sources of background noise, an acceptable estimate of it was obtained. Figure 4.3 shows the PSD of two such recordings, recorded in Seattle and Paris respectively.



(a) Seattle (b) Paris

Figure 4.3: PSD of background noise

As would be expected, most of the energy is contained in the very low frequencies, with a further decrease in magnitude around 3–3500 Hz. The conclusion was drawn that basically any frequencies would be able to carry a signal, but that the higher frequency ranges are more suitable.

4.2 Module API

The software interface to the SND module is a two-way channel as shown in figure 2.2 on page 11. In addition to the two channels used, a third channel was specified, but remained unimplemented. The third channel was intended to be used for sending configuration commands to the SND module, and might be used in later versions of the system.

The SND module receives a data transmission request from the RP, which is parsed and transmitted. The RP is notified of the current status of the SND module in much the same way. The message passing is handled by OPEN-R internals.

4.2.1 Data structures

The structures carrying data to and from the RP and SND modules was specified so as to follow the general style of the TEAM SWEDEN software. The specifications were made general enough so as not to need re-implementation if and when a new communications framework was developed. Thus, there are some fields of the structures that are defined, but not currently used.

Transmission request format

The Transmission Request Packet (TRP) is the data structure given to the Snd module from the RP, when a transmission is desired. It is also used for incoming messages in the other direction.

A packet diagram of and the 'C' source code to the transmission request format is given in figure 4.4 on the next page. The fields are described below.

time: Each packet is time-stamped as an unsigned integer in this field. This time-stamp is only used internally and is not transmitted.

opcode: This field is the main data field of the packet. These opcodes are defined in the RP, and are transmitted 'as is' by the SND module.

arg1...arg3: These fields are for optional opcode arguments.

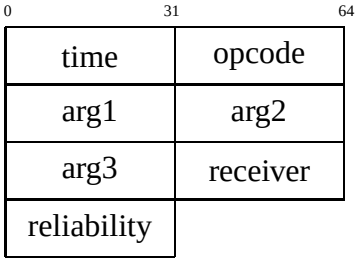
When sent from the Snd module, these fields contain information on which frequencies were sent, opcode variability and the internal PSD size.

receiver: A recipient field to be used with addressed transmission. Currently unused, but added to the packet for generality.

reliability: The TEAM SWEDEN architecture makes heavy use of fuzzy logic, and this field is a value $[0,1]$ indicating a desired level of transmission reliability when being sent from the RP, and a calculated reliability when sent from the Snd module. It was not used in the version of the software used in Seattle.

Status event format

The status event received by the RP is a unsigned integer wrapped in a separate structure for consistency. The actual states it refers to are specified in table 4.1 on page 25 as external states. This structure is used by the RP to determine whether transmission is possible or not. It is depicted in figure 4.5 on the next page.

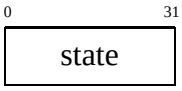


(a) Transmission request packet format

```
typedef struct {
    unsigned int time;
    unsigned int opcode;
    int arg1;
    int arg2;
    int arg3;
    float reliability;
} SndMsg;
```

(b) Transmission request C structure

Figure 4.4: Transmission request packet



(a) Status event packet format

```
typedef struct {
    unsigned int state;
} SndStatus;
```

(b) Status event C structure

Figure 4.5: Status event packet

4.3 Module states

The SND module maintains two separate state sets; one internal and one external. The internal state is a control for whether or not the SND module is allowed to perform certain actions, whereas the external state is an indication of which action is actually being performed at a given time. These actions are described below.

SND_STATE_LISTENING Indicates that the SND module is idle, i.e. it is listening for some message to be transmitted. This state is the only one in which the RP is allowed to make a transmission request.

SND_STATE_RECORDING When in this state, the SND module has detected a transmission header and is recording the entire sample sequence of the message.

SND_STATE_DECODING Denotes the fact that an entire sequence has been recorded and is being decoded. After decoding, the RP is notified of the new message.

SND_STATE_PARSING_TX This state is entered upon reception of a transmission request from the RP. The request is decoded and prepared for transmission.

SND_STATE_PLAYING A sample sequence has been generated and is being sent to the DAC on the AIBO for playback.

The state transitions are depicted in figure 4.6 on the following page, and the formal states are given in table 4.1.

```
INTERNAL_STATE := (
   SND_STATE_PLAY |
   SND_STATE_RECORD |
   SND_STATE_OFF
)
```

(a) Internal states

```
EXTERNAL_STATE := (
   SND_STATE_LISTENING |
   SND_STATE_RECORDING |
   SND_STATE_DECODING |
   SND_STATE_PARSING_TX |
   SND_STATE_PLAYING
)
```

(b) External states

Table 4.1: SND module states

Though not a part of the SND module itself, we list the messages sent by the RP for transmission. These messages reflect the *intended* state of the RP, and make up the symbol alphabet of the designed communication system. The symbol alphabet is also given in table 4.2. The use and meaning of the messages are discussed elsewhere. However, the *number* of symbols/messages is tightly tied to the safe transmission mode, being the determining factor on how much bandwidth the transmission is allowed to occupy, as in this mode, each message uses a separate carrier frequency

```
RP_STATE := ( RP_IAMDISABLED = 1 | RP_IGOFORBALL_LEFT = 2 |
              RP_IGOFORBALL_RIGHT | RP_IGOFORBALL_BACK |
              RP_THEYHAVEBALL_LEFT | RP_THEYHAVEBALL_RIGHT |
              RP_THEYHAVEBALL_BACK | RP_BALLISFREE_LEFT |
              RP_BALLISFREE_RIGHT | RP_BALLISFREE_BACK |
              RP_MEANDOPPHAVEBALL | RP_E_LITE_BALLISFREE = 13 )
```

$$\Phi_{RP} = \{s_0 = 1, s_1 = 2, \ldots, s_{12} = 13\}$$

Table 4.2: RP states/message alphabet

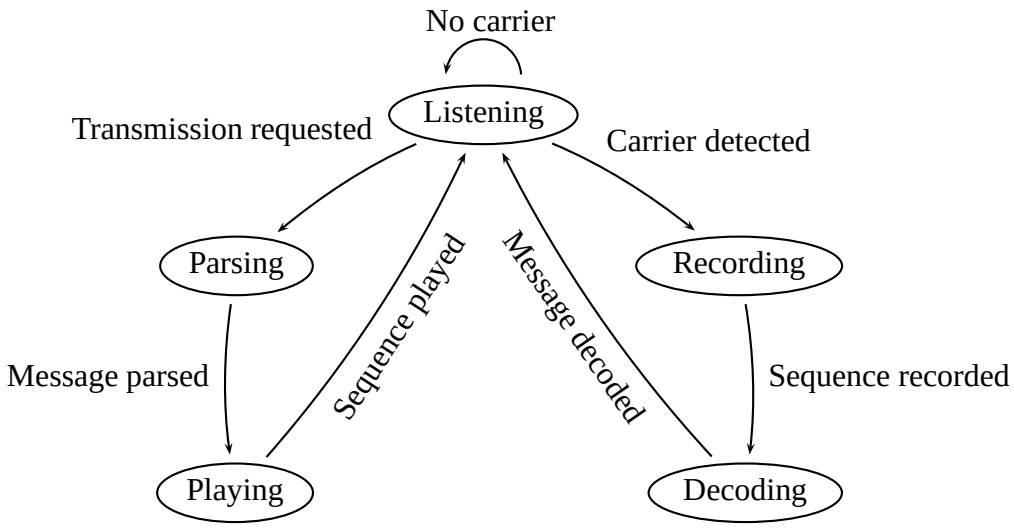


Figure 4.6: SND state transition diagram

4.4 The generic DSP library

As stated previously, two separate protocols were created; one safe-mode protocol and one high-speed, or rather, high-datarate protocol. High-data rather than high-speed because of the amount of redundancy used as described in 5.4 on page 33. Both protocols utilise a common underlying layer of generic Digital Signal Processing (DSP) routines, which also hade be be made from scratch, as no such software was made available on the AIBO platform.

This common library contains routines for several common DSP functions, and was created to act as a platform independent codebase having no coupling to the AIBO software interface.

The library was implemented in 'C' and the functions all use double precision arithmetic unless noted otherwise. Not all of the functions were used in the production code that ran on the robots. The functions are discussed briefly below.

`gen_pn`

Generates a random value of 1 or -1 using a maximum-length feed-back shift register sequence. This was used for channel identification and to measure signal attenuation in the early stages of the project.

`hanning`

Calculates the Hanning window function:

$$W = \frac{1}{2}\left(1 - \cos\frac{2\pi n}{M-1}\right) \qquad n = 0, 1 \ldots M-1$$

where M is the length of the window. The 'C' function may also optionally window data supplied to it.

`hamming`

Calculates the Hamming window function:

$$W = 0.54 - 0.46\cos\frac{2\pi n}{M-1} \qquad n = 0, 1 \ldots M-1$$

where M is the length of the window. The 'C' function may also optionally window data supplied to it.

Both the `hamming` and `hanning` functions are used for creating Finite Impulse Response (FIR) filters and for affecting the accuracy of amplitude and frequency when calculating the Fast Fourier Transform (FFT).

`fir_lp, fir_hp, fir_bp`

These three function all use the windowing method to create FIR filter coefficients for a low-pass, band-pass and high-pass filter respectively. This type of basic filter design is discussed in any basic book on digital signal processing, e.g. [14].

`convolve`

Implements the common convolution formula

$$y(n) = \sum_{k=0}^{M+N-1} x(k)h(n-k)$$

where M and N are the number of samples in $x(n)$ and $h(n)$ respectively. The function $h(n)$ is the impulse response of a given Linear Time-Invariant (LTI) system, and the convolution of $x(n)$ with $h(n)$ is the *response* of the system with $x(n)$ as input.

`convolve_long`

Also implements the above formula, using only integer arithmetic to make the operation faster.

`fsk_encode,ask_encode,bin_encode,psk_encode,psk_encode_diff`

Implements the modulation schemes discussed in section 3.4 on page 17.

All the encoding schemes receive an array of bytes, which is encoded most significant bit first. The function `bin_encode` creates a binary on–off sequence as seen in the plots in section 3.4 on page 17.

The functions `fsk_encode`,`ask_encode`, and `psk_encode` are all phase-coherent versions of the respective encoding schemes.

`psd`

Estimates the Power Spectral Density (PSD) of a given input, using the Welch averaging method. The PSD is normally calculated using the FFT. The Welch variant of the PSD calculates a modified periodogram as described in equation (4.1).

$$P_{xx}^{(i)}(f) = \frac{1}{MU}\left|\sum_{n=0}^{M-1} x_i(n)w(n)e^{-j2\pi fn}\right|^2 \qquad i = 0,1,\ldots,L-1 \tag{4.1}$$

where x_i is the momentary input, w an optional time window, M the length of x_i and w, L the number of averages, and U a normalisation factor for the influence of the window. U is given by

$$U = \frac{1}{M}\sum_{n=0}^{M-1} w^2(n)$$

The full Welch power spectrum estimate is given by calculating the average of the L number of periodograms from (4.1) by

$$P_{xx}^{W}(f) = \frac{1}{L}\sum_{i=0}^{M-1} P_{xx}^{(i)}(f) \tag{4.2}$$

As the periodogram shown in (4.1) is in essence a Discrete Fourier Transform (DFT) with a windowing of the input data, we can rewrite it using the Fast Hartley Transform (FHT) as

$$P_{xx}^{(i)}(f) = \frac{1}{MU} |wFHT(x_i)|^2$$

where $wFHT$ is the combined windowing and Hartley transform operation.

The implementation for this project uses a similar transform, the FHT, which utilises the fact that the signal is real valued, and thus only works for such signals. The FHT is on average 50% faster than the corresponding FFT. The source code for the FHT used for this thesis was written by Ron Mayer.

The PSD is discussed further in [6] and [14], and the FHT is described in full in [5].

`carrier_detect`

Uses the `psd` routine for detection of a given frequency's presence in the input signal.

This routine is used for determining whether a transmission has been initiated by another robot, and represent – together with the `psd` routine – the core of the SND module.

`fft_convolve`

FFT version of a circular convolution.

In addition to the functions described above, a few trivial functions and data structures were also implemented for convenience.

Chapter 5

Protocol implementations

This section discusses the parts of the SND module implementing the two codecs for the protocols.

Much of the source code used for both codecs is the same, and we will discuss the differences when nessecary.

The receiver and transmitter models used are simplified versions of the ones in figure 3.1 on page 13. Due to the fact that the safe mode only uses one signalling element, the synchronisation and symbol changes usually performed in the channel codec stage is not needed. The high-speed mode does not perform any channel coding either.

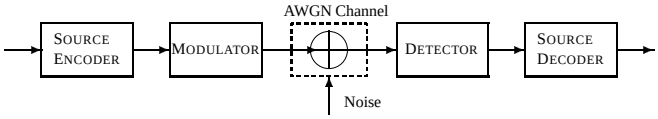


Figure 5.1: The modified system overview.

5.1 Transmitter

The transmitter is a very simple double-buffering wrapper around the OPEN-R notification mechanism tied to the sound primitives on the AIBO robot.

The double-buffering routine handles the copying of previously generated sample data from a internal shared buffer. The generation of these sample data is discussed in sections 5.3.1 on page 32 and 5.4.2 on page 34.

5.2 Detection and Header Creation

A message transmitted by the Snd module consists of two distict parts: a header, and the payload or data. The header is common for both procotols, whereas the payload differs in both modulation technique and encoding scheme.

5.2.1 Frequency allocation

The final version of the Snd module uses five separate frequency bands, modifiable at compile-time. The bands used for the tournament in Seattle were:

- 2738 - 3113 Hz
- 2910 - 3285 Hz
- 3066 - 3441 Hz
- 3300 - 3675 Hz
- 3488 - 3863 Hz

Valid carriers were selected by calculating a PSD of 2048 bins in the frequency interval 0–4000 Hz, and then selecting – starting with the lowest frequency in the band – every fourth frequency.

This was done so as to not have any interference due to frequency leakage in the FFT. The leakage could have been remedied by using a Hanning window, but that would have forced additional multiplications in the detector and decoder.

5.2.2 Header

The header consists of a single sine transmitted for a given period of time, which is configurable at both run– and compile-time.

For the high–speed mode, the header is followed by silence for 0.5 times the length of the header. This silence is used for synchronising the start of a message

When using the safe mode protocol the header is continuously transmitted without interruption, and forms part of the message data itself.

5.2.3 Detector

The detector of the Snd module uses the modified PSD routine briefly discussed in section 4.4 on page 28. The detection process is very simple.

The detection routine is automatically called via the OPEN-R notification mechanism as soon as the input sample buffer is filled. The data is converted to mono and is normalised. A PSD with frequency resolution of 512 bins is calculated. The bin with the highest amplitude is compared to the pre-defined valid frequencies, and if the bin matches any one of the pre-defined frequencies ± 1 bin, a carrier is considered to be detected.

As the carrier frequencies are all in the audible range, there is the possibility of other sources – such as cellular phones, pagers, PDA alarms etc – generating the same frequencies. This problem is alleviated by requiring a carrier to be detected in three consecutive sample sequences for a signal to be considered a valid transmission header.

Once the received signal has been deemed valid, the system enters recording mode. A total of two seconds of data is recorded and the system subsequently enters decoding mode.

A flowchart of the detector is shown in figure E.2 on page 47.

5.3 Safe mode and common codec functionality

This section discusses the safe mode codec.

The safe mode is the transmission mode used for the Seattle competitions and has proven itself to be very robust, even in very disadvantageous environments such as the relatively open spaces during the soccer matches.

We also discuss the parts of the codecs that are common for both transmission modes.

5.3.1 Coder

The encoding is a very simple one–to–one mapping of the 32-bit binary opcode from the TRP value to a modulated sample sequence. This step is performed while the Snd module is in the `SND_STATE_PARSING_TX` state

Since the safe mode only consists of a single sine wave transmitted during a full second, or a 13–level FSK modulated signal with a 1 second signalling element length, the encoder is basically a control for making sure the opcode value is a valid one. In case the opcode is invalid, it is substituted for the largest valid opcode.

The opcode is then used as an index into an array of frequencies for selecting a carrier frequency, and a sine sequence is generated from this. The sequence is scaled to fit into $[-127, 127]$ as an 8-bit sample, and converted to 8-bit integer arithmetic.

The external state is changed to `SND_STATE_PLAYING` and the internal state to `SND_STATE_PLAY`, and the OPEN-R notification mechanism is told to start playing.

5.3.2 Decoder

The safe mode decoder is an extension of the detector in the sense that it uses a PSD for detecting frequencies in the input signal, with an increased frequency resolution of 1024 PSD bins. The additional functions performed by the decoder is basically filling in the various fields of the TRP to be returned to the RP.

The opcode is calculated by comparing the frequency bin with the highest magnitude to all of the valid transmission frequencies, and defaults to `0xffffffff` if no valid opcode is found. A flowchart representation of the opcode assignment is given in figure E.1 on page 46.

The argument fields all contain various extra decoding information that could be used by the RP to determine the validity of the message.

arg1 Contains the frequency bin with the highest value, which should ideally be the same as arg2. It may be ± 1 bin off.

arg2 This field is filled with the *ideal* value of a given frequency bin. This is the reference value which arg1 is compared to.

arg3 The final argument contains the total number of bins in the PSD.

The reliability–field of the SndMsg structure is calculated by the following expression:

$$rel = \frac{x(m-1) + x(m) + x(m+1)}{\sum_{n=0}^{M-1} x(n)}$$

where x is the calculated PSD, M is half of the number of PSD bins and m the index of the largest value in x. The reliability field is not used in the Snd module itself, but is left for the RP to use at its discretion.

5.4 High speed mode

This section discusses the high speed mode codec. The high speed mode uses a 2-shift PSK modulation method with bit-duplication for error detection and correction. It can transmit a total of sixteen bits, out of which the first bit is for phase synchronisation at the receiver, leaving fifteen bits for data.

5.4.1 Coder

As stated above, the protocol allows for a total of sixteen bits to be transmitted; the first bit of these is defined to always be zero for extracting a reference phase from the message.

The final version of the coder only considers the two lowest bits of the opcode received from the RP. The low opcode bit is mapped to the seven bits following the synchronisation bit in the

message, and the high opcode bit is mapped to the final eight bits. Figure 5.2 shows the structure of a complete two-byte packet.

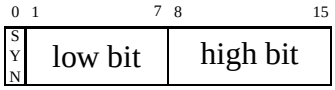


Figure 5.2: High speed bit allocation

If the low bit of the opcode is a 1, bits 1–7 in the message are also set to 1, otherwise they are set to 0. Bits 8–15 are similarly mapped to the high opcode bit. This redundancy is used for calculating the reliability of a message in the receiver.

After having generated the two-byte message packet, a sample sequence is generated and played in much the same way as with the safe mode, with the main difference that only one frequency is used. Both FSK and PSK were implemented in the generic DSP library.

The carrier frequency is taken from the values in the same array of frequencies as the safe mode protocol. The first frequency in the array is selected. This carrier is sent un-shifted for binary 1 and shifted by π radians for binary 0.

5.4.2 Decoder

The PSK decoder is somewhat more complex than the FSK decoder. It is based on a conversion of the received signal from PSK to a bi-polar ASK signal.

As the received sample data contain both the header and a period of silence, as described in section 5.2.2 on page 31, the first step in decoding the signal is discarding this header and locating the start of the actual message. This is done be using the detector function to keep identifying the header as a valid signal. As soon as the detector reports a non-detected signal, the system tries to locate the start of the synchronisation bit. The synchronisation bit is considered detected when the absolute value of the input exceeds a configurable pre-determined value. The time between the end of the header to the location of the synchronisation bit is measured, and if it exceeds the length of the header itself, the transmission is discarded.

PSK to bi-polar ASK conversion

The need for a synchronisation bit as part of the data is to be able to extract a known phase from the recevied signal. This phase is needed to be able to convert the received PSK signal to a bi-polar ASK signal. The conversion is described by the following expression:

$$v_{ASK}(n) = x(n)\sin(2\pi fn + \phi) \qquad n = 0, 1, \ldots M-1 \tag{5.1}$$

where x is the recorded signal, f the normalised carrier frequency and ϕ a phase adjustment.

If we assume that the signal x is a phase coherent PSK modulated signal with a zero-bit phase shift of π radians, as is the case here, the expression for x is

$$x(n) = c(n)\sin(2\pi fn) + (c(n)-1)\sin(2\pi fn+\pi) \qquad n = 0,1,\ldots M-1 \tag{5.2}$$

Substituting (5.2) for $x(n)$ in (5.1), we get:

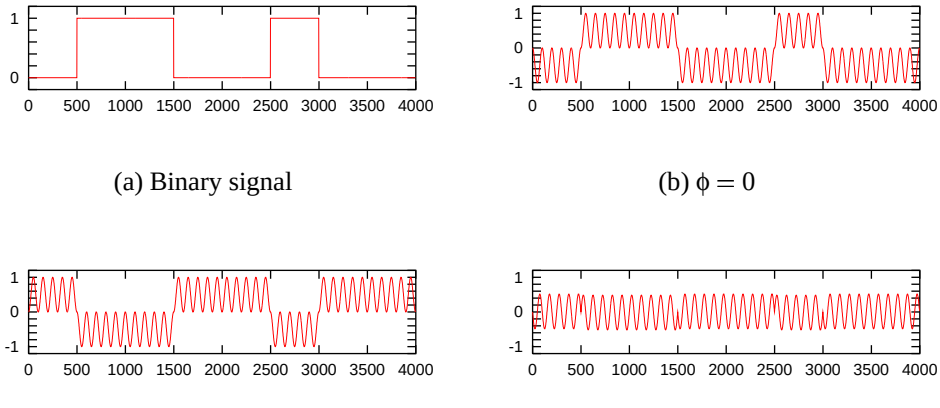
$$\begin{aligned} v_{ASK}(n) = \quad & c(n)\sin(2\pi fn)\sin(2\pi fn+\phi) + \ldots \\ & \ldots (c(n)-1)\sin(2\pi fn+\pi)\sin(2\pi fn+\phi) \qquad n = 0,1,\ldots M-1 \end{aligned} \tag{5.3}$$

The term containing the factor $c(n)$ describes the positive bits in the data carried by the sample sequence, the term containing the complement of $c(n)$ describes the zero bits.

The phase adjustment term ϕ makes it possible to decide the polarity of the extracted bi-polar ASK signal. If we let e.g. $\phi = 0$, then the bits will be given by:

$$\begin{aligned} 1: &\quad \sin^2(2\pi fn) \\ 0: &\quad \sin(2\pi fn+\pi)\sin(2\pi fn) \end{aligned}$$

This shows that all the positive bits would be represented by only positive samples, and the zero bits would be completely contained in the negative range. If we would use $\phi = \pi$, the opposite would be true. Figure 5.3 shows the resulting sequence for three different phase adjustments.



(a) Binary signal (b) $\phi = 0$

(c) $\phi = \pi$ (d) $\phi = \frac{\pi}{2}$

Figure 5.3: PSK to bi-polar ASK conversion

Bit extraction

The standard way of decoding an ASK signal would be to pass it through a low-pass filter with a cut-off frequency of $1/4$ times the bitrate of the original signal, in Hz (this is also known as the

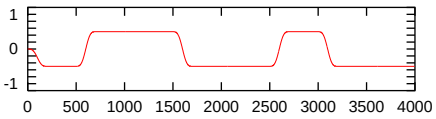


Figure 5.4: Filtered ASK sequence

fundamental frequency). Applying this to the sequence in figure 5.3(b) on the page before would yield the sequence shown in figure 5.4.

One would then calculate the sum over the length of one bit, and make a decision of 0 or 1 based on the polarity of that sum.

However, filtering is a fairly expensive operation in terms of CPU cycles, and it would be preferable to avoid it. In addition to this, there is no pressing reason for re-creating the binary signal for making a decision. It is quite possible to follow the same scheme for decisionmaking as usual, with the exception of the filtering process. Since all the sample representing zero bits are in the negative range and the one bits are in the positive range, the summing operation suffices. If we denote the summation

$$B(n) = \sum_{k=nT_b}^{(n+1)T_b-1} v_{ASK}(k) \qquad n = 0 \ldots N-1 \tag{5.4}$$

where T_b is the the number of samples for one bit and N the total number of bits in the message, the decisionmaking can be expressed as:

$$b_n = \begin{cases} 1 & : \quad B(n) > 0 \\ 0 & : \quad B(n) < 0 \end{cases}$$

This summation is performed over all the 16 bits in the received message.

Phase extraction

After having received a sample sequence, the current phase of the signal is not know. This phase is needed for the conversion discussed in section 5.4.2 on page 34.

Fortunately, since we know that the first bit of the transmission *always* is zero, we can use this for extracting the phase of the entire signal.

The synchronisation bit of the signal x can be defined as

$$x_s(n) = \sin(2\pi f n + \pi + \theta) \qquad n = 0 \ldots T_b$$

where θ is a random variable. The randomness derives from the fact that it cannot be determined exactly at which sample the actual data in the signal starts. Additionally, there may be phase variations in the signal during the transmission over the acoustic channel.

Then the ASK converted variant would be:

$$v_{ASK}(n) = \sin(2\pi f n + \pi + \theta)\sin(2\pi n f) \qquad n = 0 \ldots T_b$$

If we rewrite this using (5.4) and take into account that $n = 0$ we get a modified version of the summation depending on θ instead of n:

$$B(\theta) = \sum_{k=0}^{T_b-1} \sin(2\pi f k + \pi + \theta)\sin(2\pi k f)$$

Since the samples we are now handling are a zero bit, this sum should be as negative as possible. Thus, the θ that minimises the sum $B(\theta)$ is the phase adjustment needed to convert the PSK signal to ASK.

This was done in software by calculating several values of $B(\theta)$ and selecting the lowest. The 'C++' sourcecode is featured in table 5.1

```
 1: double
 2: Snd::ExtractPhaseNeg(double *data, unsigned int length)
 3: {
 4:   static double scratch[32768];
 5:   double step = 0.1;
 6:   double sum=0;
 7:   double min=0, minphase;
 8:   unsigned int n = MIN(32768,length);
 9:   double fc = (double)m_carrier / (double)SND_FS_REC;
10:   memset(scratch,0,32768*sizeof(double));
11:
12:   for(double phase=0.0; phase<M_2PI; phase += step)
13:   {
14:      sum = 0;
15:      memcpy(scratch,data,n);
16:
17:      for(int i=0;i<n;i++)
18:        scratch[i] *= 2*sin(M_2PI * fc * (double)i + phase);
19:
20:      for(int i=0;i<n;i++)
21:        sum += scratch[i];
22:
23:      if(sum<min) {
24:        min = sum;
25:        minphase = phase;
26:      }
27:   }
28:   return minphase;
29: }
```

Table 5.1: Phase extraction source code

TRP fields

The opcode– and reliability-fields of the TRP are calculated in a different manner when using the high-speed mode than the safe mode. Recall that the actual data available to transmit is only two bits, with the low bit having six redundant bits, and the high bit seven.

The opcode is calculated by counting the number of ones and zeroes in the databits. If bits 1-7 contain more ones than zeroes, it is considered to be a positive bit, and analogously for the last eight bits.

The reliability-field takes into account the varying number of ones and zeroes in the databits. If the low bit contained three ones and four zeroes, it would be considered a zero, with four correct bits. The number of correct bits from both low and high bit are summed up. The synchronisation bit always account for one correct bit. The reliability-field is then calculated by:

$$rel = \frac{N_c}{16}$$

where N_c is the total number of correct bits.

The argument-fields are not used in the high-speed mode.

Chapter 6

Discussion and Conclusions

This chapter briefly discusses other approaches to audio communication in the ROBOCUP domain, and ends the report with some conclusions drawn from the experiences of the project.

6.1 Other approaches

The 2000 ROBOCUP competition sported the first use of the audio capabilities of the AIBO robots, as simple beeps were used by one team during one of the challenges. The 2001 ROBOCUP tournament was the first time communication was being used for anything more than this type of simple notifications.

Besides TEAM SWEDEN, three other teams used a communications system [4]:

- Cerberus, the Turkish-Bulgarian team
- University of New South Wales, Australia
- University of Pennsylvania, United States

Out of the three teams, the Turko-Bulgarian team had designed the most complex system. It consisted of a six-frequency protocol, where three frequencies form a combined command/ID group, and the last three represent data. The system was never put into use during the tournament due to the system's inability to handle the high-noise environment.

From what information could be gleaned over the shoulders of the two other teams, their approach was much similar to the one of TEAM SWEDEN. In fact, during the time in Seattle, TEAM SWEDEN was asked on a few occasions to please refrain from testing the communication software during the matches, as it interfered with the other teams transmission software.

6.2 Conclusions

Much of the project has been devoted to learning not only the software of the AIBO robot and TEAM SWEDEN architecture, but also digital transmission in general.

From the onset of the project, it was not clear what the specifications of the hardware of the AIBO were, and much time was spent trying to coax this information from the documentation and Sony representatives. As the documents were delayed in translation from Japanese to English, there was a considerable slowdown of the work during the first half of the project.

Another issue was the fact that the amount of data to be transmitted using the system was difficult to estimate. There was much discussion within TEAM SWEDEN about what data to transmit and how to encode this data in the number of bits available.

The trial of fire for the communications system was the 2001 ROBOCUP competition in Seattle. Several fieldtests were made previous to that, but as TEAM SWEDEN did not have neither the possibility to activate more than three or four robots at once nor an audience of several hundred people, a real-world test could only be preformed during the competitions. The system worked flawlessly during both the fieldtests and, more importantly, the competitions.

The problem formulation in chapter 1 was set to examine what amounts of data would be able to transmit under various conditions. By using the high-speed protocol in benign environments, i.e. no interference and a distance of about 30 cm, 16 bits during one second have been successfully decoded, and it is likely that 32 or even 48 could be achieved without much trouble. However, as the distances increase and other interferences arise, the safety of this transmission mode decreases dramatically. It is possible to remedy the noise by adaptive means, but the hardware limitations of the AIBO robots prohibit this. To acheive increased data rates, the safe mode is probably better suited.

6.3 Further work

As the rules for the ROBOCUP competitions have changed for 2002, allowing wireless LAN cards to be used, the discussion of further work on the Snd module is a moot point. However, there are a few issues that could be mentioned as possible improvements:

- The module should be split into smaller modules. There should be at least a `Transmit` and `Receive` module, and perhaps a controlling `Communicator` module.
- The dsp library (section 4.4 on page 27) is using naïve implementations of the routines. These should be optimised, and ideally be implemented using hand-crafted assembly for the tighter loops, such as in the FHT routine.
- The safe mode should be expanded to use several signalling elements by shortening the length of each element. This would also imply some form of error correction or at the

very least error detection. By just adding two additional signalling elements, the number of total messages would be 2197 (13^3).

- Convert the double precision arithmetic routines to 32-bit integer arithmetic. This would improve performance dramatically on the MIPS RISC CPU used on the AIBO.
- Re-work the detection algorithm to get rid of the PSD calculation.

Appendix A

RoboCup 2001

The 2001 installation of ROBOCUP was held at the Washington State Convention and Trade Center, Seattle USA betweeen august 1st and 10th. It was the fifth time the ROBOCUP competition and conferences were held, and there were four new teams in the SLRL, making a total of 16 teams competing. The conference was co-hosted with the International Joint Conference on Artificial Intelligence (IJCAI)

A.1 The competitions

The competitions consisted of three challenges in addition to the soccer tournament. Each challenge and the tournament awarded the winners, second and third place with points. These points were summed up to yield the final results.

The results of the competitions are given below.

1. UNSW United'01, University of New South Wales, Australia
2. CM-Pack'01, Carnegie Mellon University, USA
3. UPennalizers'01, University of Pennsylvania, USA
4. LPR'01, France
5. BabyTigers'01, Osaka University, Japan
6. Kyushu, Japan
7. SPQR'01, Rome, Italy
8. USTC, China
9. ARAIBO'01, Tokyo, Japan
10. Essex Rovers'01, United Kingdom
11. German Team'01, Germany
12. McGill RedDogs'01, Canada
13. Cerberus, Balkans, Turkey and Bulgaria
14. **Team Sweden'01, Sweden**
15. Undecided'01, Melbourne, Australia
16. Washington, USA

Appendix B

Team Sweden 2001

The following people were actively involved in the Team Sweden project:

Name	Role	Location
Alessandro Saffiotti	Team captain	OrU
Zbigniew Wasik	Graduate student	OrU
Robert Johansson	Master student	OrU
Stefan Johansson	Site coordinator	BTH
David Erman	Master student	BTH
Jan Kronqvist	Master student	BTH
Kalle Prorok	Site coordinator	UmU
Andreas Björklund	Master student	UmU
Peter Larsson	Master student	UmU
Johan Carstensen	Master student	UmU

Table B.1: TEAM SWEDEN members for 2001

Appendix C

Internet Resources

The table below gives some of the internet sites relevant to the project. They are current as of November 4th, 2001.

`http://www.aass.oru.se/Agora/RoboCup/`	The official TEAM SWEDEN site.
`http://www.aass.oru.se/~asaffio/Software/TC/`	The 'Thinking Cap' site.
`http://www.aibo.com`	The official AIBO site.
`http://www.robocup.org`	The official ROBOCUP site.
`http://www-2.cs.cmu.edu/~robocup2001/`	The official ROBOCUP 2001 site.
`http://www.bth.se`	Blekinge Institute of Technology
`http://www.oru.se`	Örebro University
`http://www.umu.se`	Umeå University

Table C.1: Selected Internet resources and university sites

Appendix D

Acknowledgements

The author would like to thank the following for their support and help during the project.

- The Department of Software Engineering and Computer Science at BTH for sponsoring the project.
- My family for supporting me no matter what. Most notably my partner for bearing with my many mood swings during the project.
- TEAM SWEDEN – For making all those long nights of coding bearable; in particular during the weeks in Seattle.
- Michael Numminen for volunteering to proofread this paper.
- All the people who helped me salvage the remnants of this thesis after a massive hardware failure.

Appendix E

Flowcharts

The flowcharts in this section have intentionally been kept overly simple, and are depicted here to be viewed in conjuction with their descriptions in the text. Most notably, basic functionality such as normalisation and scaling is most often not included in the charts.

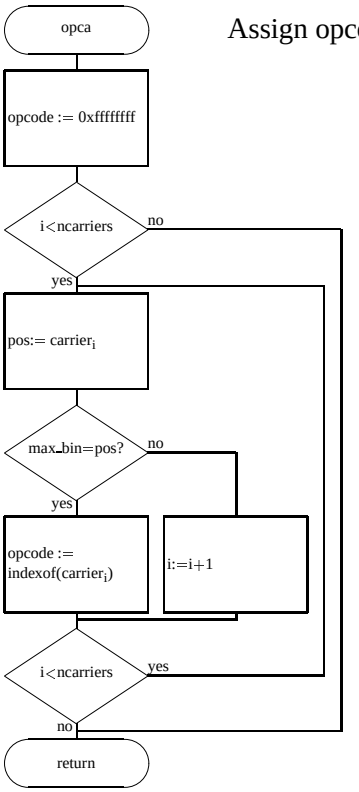
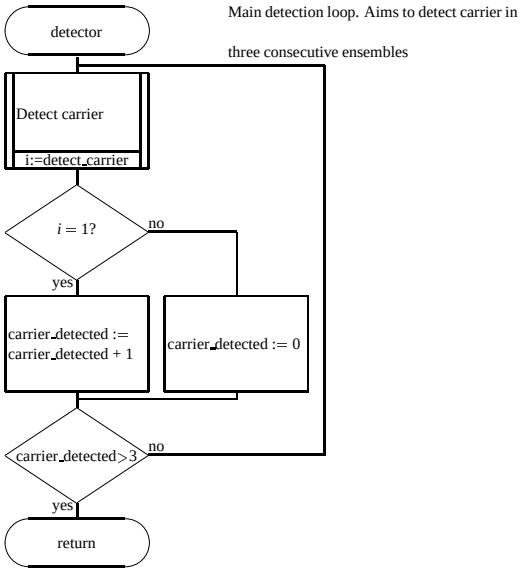
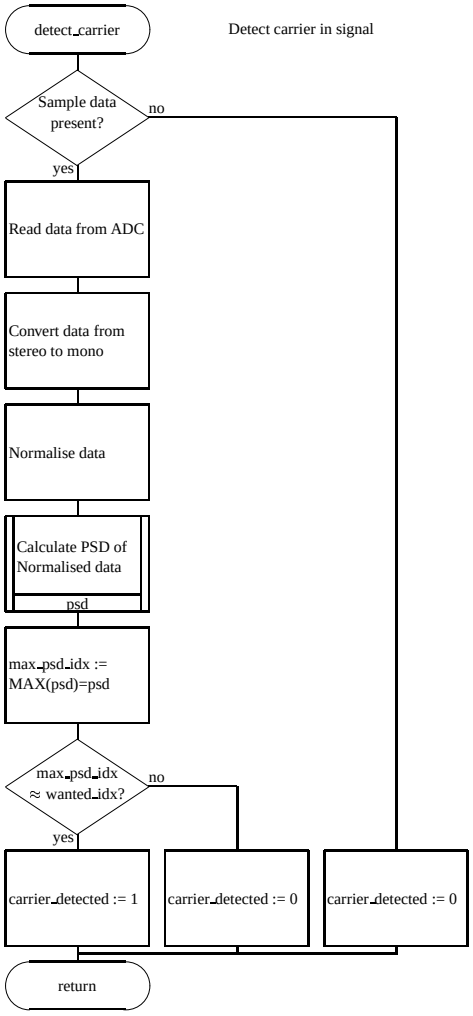


Figure E.1: Safe mode opcode assignment



(a) Main detector loop



(b) Carrier detector

Figure E.2: Detector flowcharts

Appendix F

Acronyms

ADC Analog-to-Digital Converter. Device for converting analog voltage levels to digital numbers. Implies sampling.

AI Artificial Intelligence.

AMI Alternate Mark Inversion. A channel encoding scheme using alternating positive and negative representations of a binary 1.

API Application Programming Interface. A set of functions and/or structures for accessing a specific piece of hard– or software.

ASK Amplitude Shift Keying. PCM method using varying amplitude for denoting values. Also known as amplitude modulation.

AWGN Additive White Gaussian Noise. Noise that is spread on $[0,1]$ with a Gaussian distribution. The term 'additive' refers to the addition of this noise to some signal.

BTH Blekinge Institue of Technology.

bps bits per second. Number of bits transferred per second.

codec coder-decoder. Collective name for a coder/decoder pair implementing some protocol or algorithm.

CMD Commander. The hardware abstraction layer of the TEAM SWEDEN architecture.

CPU Central Processing Unit.

CRC Cyclic Redundancy Check. A method for veryifying data integrity using a cyclic XOR calculation of the data. The data is uniquely identified by the result of the operation, called a *checksum*.

DAC Digital-to-Analog Converter. Device for converting digital numbers to analog voltage levels.

DFT Discrete Fourier Transform. Discrete form of the Fourier transform.

DOF Degrees Of Freedom. The number of movable joints on a robot.

DSP Digital Signal Processing.

EFA Electric Field Approach.

FFT Fast Fourier Transform. Fast version of the DFT for sample vectors whose lengths are a power of 2.

FHT Fast Hartley Transform. A real-valued transform similar to the FFT but using less computations.

FIR Finite Impulse Response. A non-recursive filter.

FSK Frequency Shift Keying. PCM method using varying frequencies for denoting values. Also known as frequency modulation.

GDP Graphical Development Platform.

GM Global Map. *Maintains* a perceptual image of the immediate surroundings of the AIBO robot.

HAL Hardware Abstraction Layer. Exports a consistent API for controlling similar, but different types of hardware.

HBM Hierarchical Behaviour Module. The part of the TEAM SWEDEN architecture responsible for implementing basic bahaviours.

HDB3 High Density Bi-Polar 3. Channel encoding scheme using zero-substitution.

KTH Royal Institute of Technology.

LPS Local Perceptual Space. The data structure representing the immediate surroundings of the AIBO robot.

LTI Linear Time-Invariant. A system that is both linear and does not vary over time.

OrU Örebro University.

PAM Perceptual Anchoring Module. *Creates* a perceptual image of the immediate surroundings of the AIBO robot.

PCM Pulse Code Modulation. Methods for creating discrete levels from binary data.

PSD Power Spectral Density. Measurement of the energy content of a given signal, frequency by frequency.

PSK Phase Shift Keying. PCM method using varying phase for denoting values.

RISC Reduced Instruction Set Computer.

RP Reactive Planner. The high-level AI part of the TEAM SWEDEN architecture.

SLRL Sony Legged Robot League.

SND Sound Module.

SNR Signal-to-Noise Ratio. A measure of the energy of the desired signal in relation to the amount of noise in a given signal.

TRP Transmission Request Packet. The prinicipal data packet transferred from the RP to the SND module.

UmU Umeå University.

Bibliography

[1] Aibo website.
http://www.aibo.com.

[2] Robocup website.
http://www.robocup.org.

[3] Team sweden team description website.
http://www.aass.oru.se/Agora/RoboCup/.

[4] Robocup 2001.
In Press, 2002.

[5] R. N Bracewell.
The Hartley Transform.
Oxford University Press, New York, 1986.

[6] Anders Brandt.
Ljud– och vibrationsanalys I.
Saven EduTech, Blekinge Tekniska Högskola, 2000.

[7] Fred Halsall.
Data Communications, Computer Networks and Open Systems.
Addison–Wesley, 1995.

[8] Simon Haykin.
Digital Communications.
John Wiley & Sons, 1998.

[9] John Johansson.
An electric field approach.
Master's thesis, Blekinge Technical Institute, 2001.

[10] Robert Johansson.
A graphical development platform for aibo robots.
Master's thesis, Örebro University, 2001.

[11] S. Johansson and A. Saffiotti.

Using the electric field approach in the robocup domain.
In *Proc. of the Int. RoboCup Symposium*, Seattle, WA, 2001.
To appear. Online at http://www.aass.oru.se/~asaffio/.

[12] K.K Masahiro Fujita.
An open architecture for robot entertainment.
Technical report, D21 Laboratory, Sony Corporation, 1998.
Online at http://www.aiboworld.tv/_download/common/OpenR.pdf.

[13] A. Saffiotti P. Buschka and Z. Wasik.
Fuzzy landmark-based localization for a legged robot.
In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1205–1210, Takamatsu, Japan, 2000.
Online at http://www.aass.oru.se/~asaffio/.

[14] John G. Proakis and Dimitris G. Manolakis.
Digital Signal Processing. Principles, Algorithms and Applications.
Prentice Hall, 1996.

[15] A. Saffiotti and K. LeBlanc.
Active perceptual anchoring of robot behavior in a dynamic environment.
In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3796–3802, San Francisco, CA, 2000.
Online at http://www.aass.oru.se/~asaffio/.

[16] Sony.
Sony Four Legged Robot Football League Rule Book.
Confidential.

[17] Sony.
Sony develops open-r architecture for entertainment robots.
Press Release, 1998.
Online at http://www.sony.co.jp/en/SonyInfo/News/Press/199806/98-052/index.html.