Universität Bielefeld

Faculty of Technology

Interactive sonification for monitoring and skill learning

Diplomarbeit

In the degree program 'Naturwissenschaftliche Informatik'

written by

Jessica Hummel

May 14, 2009

Supervisors: Dr. Thomas Herrmann
Prof. Dr. Helge Ritter
Dr. Christopher Frauenberger (Queen Mary, University of London)
Dr. Tony Stockman (Queen Mary, University of London)

Abstract

Sonification is the representation of data through non-speech sound.
This thesis presents the design, implementation and evaluation of a sonification system, which gives real-time auditory feedback to a performer who is carrying out moves on sports equipment called a German wheel. The system, which is implemented in the programming language SuperCollider, uses a magnetometer to collect data about the motion of the wheel. Parameters of this motion are then transformed into sound to give real-time feedback to the performer. The aim of the project is to examine whether such additional convergent audio feedback can lead to an improved performance of wheel moves. The design and implementation of four different sonification approaches is discussed, two of which were chosen to conduct an exempla study. The study was carried out with a group of seven novices and four experts and shows a significant positive influence of one of the sonifications on the performance of the given task by the experts.

Preface and Acknowledgements

The implementation of this research project would not have been possible without the help of many friendly people whom I wish to thank sincerely for the encouragement and support they gave me.

First and foremost, I wish to thank my supervisors Dr Thomas Hermann, Dr. Christopher Frauenberger and Dr Tony Stockman for the large amount of knowledge they passed on to me. I also want to thank them for the great guidance and support they gave me despite the spatial distances, which sometimes made communication more difficult.

Special thanks also goes to Simon Schulz and Jan Anlauf for the assembly of the magnetometer and for sharing their specialised knowledge concerning this hardware with me. For endless advice regarding scientific writing I would like to thank Jonathan Howard. Furthermore I would like to thank Nhung Nguyen, Jan Block and their flatmates for accommodating me and making me feel welcome during my stays in Bielefeld.

I would also like to thank all people who supported me or gave me advise on various other aspects of this projects. Besides people that have already been mentioned they include Rene Tuennermann, Eckhard Riedenklau, Tobias Grosshauser, Till Bovermann, Jennifer Sheridan, Nick Collins, Ben Meghreblian, Louise Valgerdur Nickerson, Dan Seal, Nick Thomas, Andrea Steinmetz, Marc Kammer, Alex Craig, Ulf Großekathöfer and all members of the Ambient intelligence group, Bielefeld Universtiy, who I have not mentioned so far.

I also owe thanks to all participant of the experiments, including some people who have already been mentioned and Rebecca Förster, Janina Brandes, Anna Kunze, Svea Gabbei, Daniel Dornbusch, Silke Fischer, Kirsten Bergmann, Maha Salem, Christof Elbrechter, Hannah Rufus, Lisa Marie Schwarz and Claudia Muhl.

Finally I would like to thank my family and my friends, who always believe in me and support me in everything I do.

Contents

List of Figures

1 Introduction

This thesis describes the design of a closed-loop audio feedback system, which was developed during the past months in a collaborative project between the Ambient Intelligence group at the Center of Excellence in Cognitive Interaction Technology, Bielefeld University and the Interaction, Media and Communication group, Queen Mary, University of London. Sonification is the representation of data by non-speech sound. In the course of this project the motion of a piece of sports equipment called the German wheel[1] was sonified. The resulting sonification serves as real-time feedback for a performer, who carries out acrobatic moves on the rolling wheel. Our aim is to examine whether our hypothesis, that such additional auditory information can contribute significantly to skill improvement of the performer, can be statistically supported.
To scrutinise the correctness of this hypothesis, we implemented the real-time sonification system for German wheel motion. This system obtains its input data from a magnetometer that is attached to the wheel and supplies several different sonifications to give feedback to the performer. For a selection of these sonifications an exempla study was carried out for a group of novices and a group of experts to evaluate and compare the impact of the sonifications on the performance of the participants. While the results for the novices did not show a significant difference between the performance with and without sonification, the results for the experts showed a statistically significant improvement of performance with one of the sonifications. Further observation suggests that the sonification may also prove to be an effective means to support the monitoring task of a trainer.

1.1 Motivation

Human perception has always been multimodal. Besides improving overall coherence, information on the acoustic channel reinforces the allocation of redundant information on the visual channel. In nature these properties of sonic feedback are used to the advantage of living organisms when retrieving information related to physical actions. A good example, in which auditory information, as part of the multi-modal perception, contributes to motor control and task learning is the natural sound in sports. (This is discussed for tennis for instance in [Tak93]).

These considerations suggest that the use of additional auditory feedback may also

[1] see `http://www.rhoenrad.com/`

contribute to an improved perception of physical actions that do not naturally generate sound. Besides, the human sense of hearing is designed to process time-variant information, which makes sound a very suitable means for the expression of time-critical data. Therefore the use of additional auditory feedback may be of particular interest for those physical actions, which restrict the use of vision and for which the perceived information is time-critical.

Sonification is the representation of data by non-speech sound and provides different techniques to give such auditory feedback [Kra94], [Her09]. Where visualisations, such as an image or graph, represent data visually, sonification does the same acoustically. While during recent decades visualisation has been used broadly in many fields to submit information, sonification is a relatively young but promising field of research.

To appreciate how the above statements about real-time auditory feedback apply to the use of sonification as feedback for a German wheel performer, some fundamental knowledge about German wheel gymnastics is essential. Without going into different wheel move details at this point, we discuss a few properties common to the majority of moves here, in order to justify our motivation for the use of sonic feedback for this application.
Many German wheel moves are very dynamic thus include very fast changes of perspective and can only be performed in a short time window, when the wheel momentarily stands still or slows down. This means that they are highly time-critical and that the visual channel can barely be used for real-time feedback, because of the constant and quick changes of perspective the performer experiences. Thus training on the German wheel represents an adequate example of use, in which auditory real-time feedback may contribute to an improved perception of the performed action, and thereby to the acquisition of skills. This assumption is supported by the fact that research has shown that such additional convergent auditory information can enhance perception accuracy for complex sports movements [Eff05].

Besides the use of sonification as real-time feedback for the performer, the auditory feedback can also deliver additional information to the trainer of the performer to facilitate the appraisal of the quality and the deficits of a move that is carried out.

In summary this project is motivated by the following facts and assumptions:

- In nature acoustic feedback plays an important role in the perception of physical actions.
- Additional acoustic real-time feedback can contribute significantly to skill learning of physical actions.
- German wheel gymnastics is a sports discipline, which is an adequate example of use because it includes highly time-critical moves, that also restrict the use of the visual channel.
- Sonification can be the means to provide additional auditory real-time feedback to a wheel performer.

- The use of sonification may also be a valuable instrument to support a trainer in the monitoring process.

1.2 Aims of the project

Based on the assumptions that motivate this project (see section 1.1), we set up the hypothesis that additional auditory feedback, supplied by a real-time sonification system, can contribute significantly to an improved performance of wheel moves. Our aims are therefore:

- To supply the performer with an additional intuitive and effective audio real-time feedback, that supports him or her in the acquisition of new skills.
- To statistically circumstantiate our hypothesis that such additional auditory feedback can improve skill learning
- To get an indication concerning the potential usefulness of sonification as an instrument to support a trainer in the monitoring process.

To achieve these aims we had to:

- implement a closed-loop audio feedback system, that provides one or several sonifications to give real-time feedback to a performer, who practices in the wheel and/or to an outside audience.
- carry out a study to test the effect of the resulting sonification on different performers' performance and to get the opinion of observers about the usefulness of sonification to assist monitoring.

An additional aim was to make the system usable outside of a laboratory environment; thus we set the constraint that all components have to be low cost, cable free and easy to set up.

1.3 Structure of this work

The structure of this work is fundamentally comprised of four sub-tasks, which the implementation of our project is divided into. Three of these sub-tasks, namely data acquisition, feature extraction and sonification, concern the implementation of the real-time sonification system. The fourth sub-task consists of the evaluation of the exempla study we conducted. The main aim of the data acquisition was to find a sensor that when attached to the wheel, delivers adequate information about the wheel's motion. In the next step, features that are appropriate to represent this motion had to be extracted from the collected data. An accurate choice of features was crucial in this context, as their sonic representation has to adequately inform the performer about the tasks to be fulfilled in the different wheel moves. Based on these

features several different sonification approaches were then implemented, tested and finally evaluated regarding their potential to support performance improvement of beginners and advanced wheel performers.

Chapter 2 provides background information about the German wheel and different categories of wheel moves.

Chapter 3 gives an insight into existing sonification techniques and discusses the current state of the art.

Chapter 4 introduces all aspects of the approaches used to achieve the first three sub-tasks namely data acquisition, feature extraction and sonification.

Chapter 5 presents the hardware and software that was used for the implementation of the real-time sonification system and explains how to start the real-time sonification system. Furthermore it examines how the approaches described in chapter 4 were implemented - for the most part in the audio programming language SuperCollider.

Chapter 6 presents the study that was carried out to appraise the influence of the sonification on the performance of a move in the wheel. It also contains a statistical and explorative analyses of the results and some further remarks on the experiments.

Chapter 7 discusses our solutions for the different sub-tasks and provides an outlook into possible future research.

2 Background on the German wheel

In the following some basic concepts important to an understanding of this work are explored. A closer look is taken at the wheel, its properties, different wheel move categories and the task a German wheel performer has to carry out. Consecutively the 'rotational angle' of the wheel is defined.

2.1 The German wheel

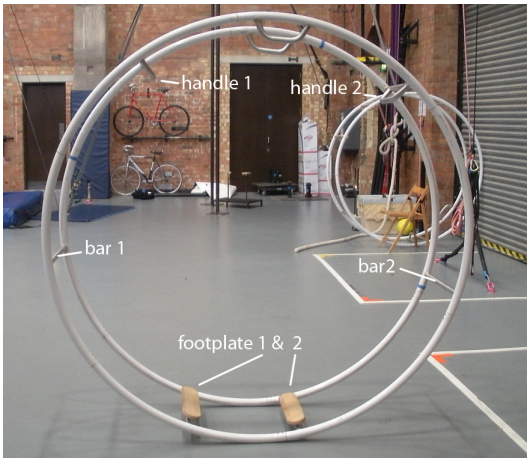


Figure 2.1: The German wheel

A German wheel is a sports apparatus, which consists of two rings whose size depend on the height of the performer (approximately height of performer + 40 cm, see Figure 2.1). The rings are connected by 6 bars of about 40 cm length.
There are three different categories of German wheel moves, two of which are performed while the wheel is rolling in a straight line and one while the wheel is rolling in a circle on one of its rims:

- In *centralised moves* the wheel is rolling in a straight line and the performer is in the centre of the wheel. Imagine someone doing a cart-wheel with a wheel around them, as depicted in Leonardo da Vinci's 'Vitruvian Man'.
- *Decentralised moves* are performed on one of the bars while the wheel is rolling in a straight line. Imagine someone sitting down on one of the bars on the sides

of the wheel. The weight of the performer causes a rolling motion, because it pushes one side of the wheel down. In the following we call the vertical axis through the wheel's centre of mass the *zero axis* in dependency on the zero point in simple harmonic motion (see Figure 4.3). If the centre of the performer's weight is on the zero axis, it does not cause a rolling motion. Later in this chapter we investigate more advanced decentralised moves such as spins around one of the bars, which are performed when the bar reaches its highest point after coming up on one side of the wheel.

- In a *spiral* the wheel is tilted and rolls in a circle on one of its rims. The performer is in a position similar to those required for centralised straight line moves. The wheel is not rolling in a straight line however. In a spiral move the movement of the wheel is similar to a coin that is standing on its edge, but then tips over and gyrates down until it is lying flat on one of its faces. By applying force and weight shifts, the performer prevents the wheel going all the way down to the floor, but brings it back up to standing instead.

For this project we mainly focus on the two straight line categories. In centralised as well as decentralised moves the motion of the wheel is induced by a shift of weight away from the zero axis. A rolling motion can be generated in different ways, all of which have in common the fact that the effective mass is transferred ahead of the general rolling motion.

Centralised moves are usually more controlled than decentralised moves. Force is applied continuously, and accordingly the direction and strength of the applied force constantly affect the rolling motion of the wheel. Therefore while performing a centralised move the aim is to permanently use the correct amount of force, direction of force and weight shift to generate the desired speed and direction of the wheel's motion.

Decentralised moves are more dynamic and therefore more time-critical, which suggests that additional acoustic feedback may be particularly useful to support a good execution of decentralised moves. Furthermore, they include very fast changes of perspective, which restrict the use of visual, but not acoustic feedback. Many decentralised moves can only be performed in a short time window, when the wheel momentarily stands still. We divide decentralised moves into two categories, according to two different events in which the wheel momentarily stands still.

1. In an *onward move* the bar comes up on one side and travels all the way over the top of the wheel: The wheel is rolling very fast and the bar with the performer on it comes up on one side of the wheel. It passes over the wheel's centre of weight force and experiences a minimum of velocity when it reaches its *highest point*. If the wheel is not too fast, it (almost) stands still in this moment. Thereafter it keeps rolling in the same direction and the bar comes down on the opposite side of the wheel.

2. In a *returning move* the bar moves up and comes back down on the same side of the wheel: The wheel is rolling slightly slower. The bar with the performer

comes up on one side of the wheel. It does not pass the highest point, which is over the wheel's centre of mass. The performer's weight force counteracts the movement. The wheel experiences a *change of rolling direction* and the bar comes down on the same side of the wheel.

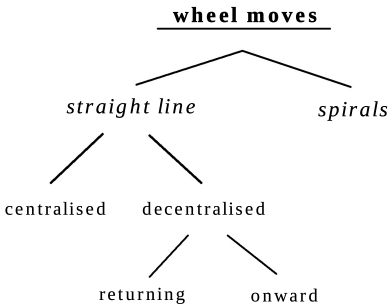


Figure 2.2: Categorisation of German wheel moves

Besides these main categories the *basic rock* (see Figure 2.3) is an important move, which can be used in all categories to generate the momentum for the performance of a move. During a basic rock the performer stands with one foot on each of the two foot-plates (see Figure 2.1). The performer straightens and bends the left and right leg alternately, thereby causing the wheel to swing. The rock can be performed facing the rolling direction of the wheel or facing out towards one of the sides of the wheel.

Decentralised straight line moves have to be performed when the wheel reaches an almost still highest point or changes its rolling direction. The performer controls the speed of the wheel by shifting weight and applying force to the rims using the principle of leverage. Two of the main tasks of the performer are therefore:

- to use the correct amount of *force and weight shift* to bring the used bar into the desired position (for instance a certain height when the wheel changes its rolling direction).
- to carry out the move with the exactly right *timing* when the wheel momentarily stands (almost) still.

Factors involved for these tasks are: the position the wheel is in, how fast it is rolling, the height of the bar when the move is performed, the time point when the wheel changes its rolling direction or the bar passes the highest point, whether the bar moves up and returns down on the same side (change of rolling direction), or whether the bar travels all the way over to come down on the opposite side (rolling the same direction).

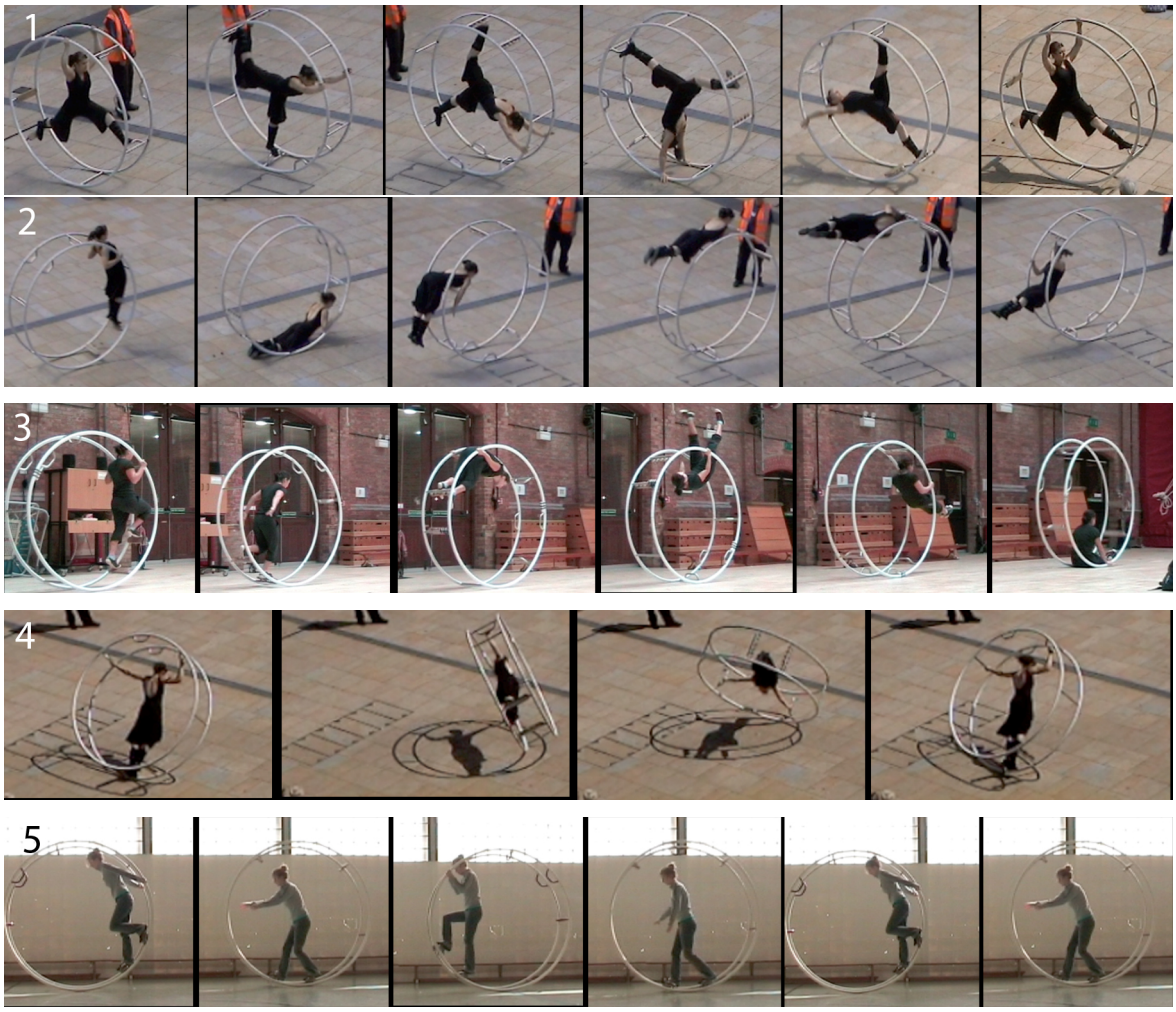


Figure 2.3: Examples of the wheel move categories: 1) centralised straight line 2) returning decentralised straight line 3) onward decentralised straight line 4) spiral 5) basic rock

2.2 Rotational angle of the wheel

In Figure 2.2 the *initial position* of the wheel is shown. The wheel is in the initial position, when the reference point between the foot-plates on the circumference of the wheel (marked in red) is on the floor. In subsection 4.3.1 our approach for the calculation of the rotational angle of the wheel and therefore the wheel's position is presented. Due to the nature of these calculations, the angle that is calculated when the wheel is in the initial position (we call this the *initial angle*) is not always the same. To provide consistency this initial angle is treated as an offset and subtracted from every calculated angle so that the resulting rotational angle of the wheel in the initial position is always $\varphi = 0$. In the following all rotational angles of the wheel are angles in relation to this initial position.

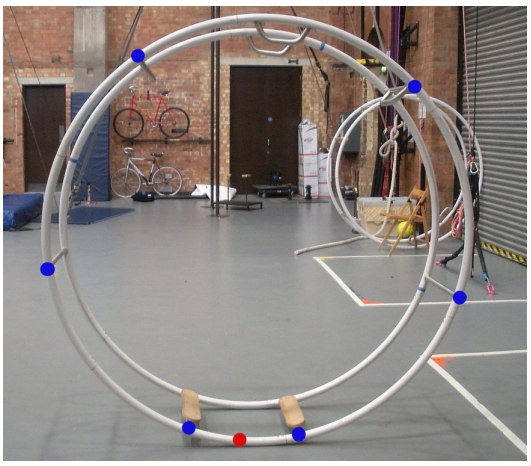


Figure 2.4: The initial position of the wheel: the reference point (red) is on the floor and the wheel's rotational angle is $\varphi = 0$.

3 Related work on motion sonification

In this chapter some existing sonification techniques are discussed. Furthermore past and current research in the field of sonification, and movement sonification in particular, is surveyed.

3.1 Existing sonification techniques

Sonification is a relatively young research field, and unlike visualisation, it is not very common yet as a means of representing data. However, sound and the human perception of it have some advantages, such as the high temporal resolution of sound and the ability to distinguish different rhythms and multiple sound-streams (e.g. different instruments in an orchestra). These characteristics have led to the emergence of sonification as a new research field (see [Kra94]). In recent years various applications for sonification have been explored, some of which are presented in section 3.2. To put this work into a greater context, we introduce the main sonification techniques which have been established:

Audification directly maps data values to time-varying sound pressure. If the input data varies, the resulting oscillation in sound pressure produces an audible sound. Without a variation in data, no sound can be produced. [Her09]

Earcons were defined by Blattner as non-verbal audio messages that are used in the computer/user interface to provide information to the user about some computer object, operation, or interaction [BSG89]. Earcons are usually abstract synthetic sounds, which are composed from smaller units/melodies that are called motives. Examples for Earcons are ring-tones, marker sounds, GUI add-ons or the sound an email inbox produces when a message arrives. In contrast to Auditory Icons, Earcons are not intuitive and therefore have to be learned.

Auditory Icons are acoustic event markers which, unlike Earcons, use everyday sounds to support an intuitive association process. One example is to express a file being deleted by the sound of throwing it into a wastebasket. Parameterised Auditory Icons are Auditory Icons which are modifiable in one or several parameters (e.g. the pitch or the volume). The above mentioned deleting sound could for example sound different depending on the type of file that is deleted. (See [Gav92].)

Parameter Mapping sonification directly maps a data attribute to a sound attribute, for example, the velocity of the rolling wheel to the pitch of a note. Parameter

Mapping sonification can be further sub-classified into the following two classes: direct Parameter Mapping sonification generates one sound event for each data item in an incoming data set (e.g. every incoming velocity data item produces one short sound, the pitch of which depends on the data value). Continuous Parameter Mapping sonification plays a continuous sound stream and alters it according to the incoming data (e.g. a continuously playing sound whose pitch varies depending on the velocity). The two approaches can be used concurrently leading to hybrid Parameter Mapping sonification. When a high data density is present Event-based Parameter Mapping sonification is often a good alternative: Single sound events are generated not for each data item, but when a certain event occurs. One example would be the generation of specific sound events to indicate when the velocity exceeds or goes below predetermined thresholds. [Her09]

Model-Based sonification uses attributes of the data to determine the properties of a sound producing system. That is, the data defines how a virtual instrument sounds. Similar to a real instrument the user can interact with a dynamic model configured from the data and excitations of the model cause the generation of sound.

More information about sonification, the specific techniques and their applications can be found in [Kra94] and [Her09]. Instruments for systematic reasoning concerning the relevant data to be displayed and the application of different sonification techniques are supplied by the 'Task Data Analysis' and the 'Sonification Design Space Map' (see [Bar98] and [dC07]). These are used and presented in more detail in subsections 4.2.2 and 4.2.3.

3.2 State of the art discussion

Alongside other applications sonification allows to generate acoustic feedback for physical actions that do not naturally produce sound. Various different applications for such audio feedback have been investigated during recent years and this sections provides a few examples. One such application can be found in the field of *sports games*. Two examples are: Blindminton, a game similar to Badminton that can be played using auditory feedback only [HHR05] and Digiwall a hybrid between a climbing wall and a computer game [LL06] .

Blindminton is one application of the hard-/software system AcouMotion, which combines tangible interfaces and sonification to a closed-loop human computer interface that allows non-visual motor control by using non speech auditory displays as major feedback channel. The system, designed by Hermann, Höner and Ritter, consists of three components namely a tangible sensor device, a dynamic model implemented in a computer simulation and a sonification engine. In its application for Blindminton the input device is a small handheld device, which can be used in a 'racket like' manner. Its location is determined by tracking the 3-D position of markers with a Lokotronic motion capture system. The dynamic model contains a virtual representation of a playing field. The player has to hit a virtual ball with the racket. If she

or he places a ball into 'out' or fails to hit it, the opponent gets a point. There is also a one-player version for Blindminton. The sonification engine uses three different types of information-carrying variables for the production of the auditory real-time feedback. Continuous variables contain information such as the ball position, ball relative position, distance to racket, ball velocity, racket position and racket orientation. Discrete variables represent events such as ball/racket contacts and ball/floor contacts. The virtual space is divided into different zones and pseudo-discrete events occur when the ball crosses a zone plane. A flying ball can, for instance, contribute a level-modulated sound pattern the pulse rate of which increases when the ball approaches the racket. Hermann, Höner and Ritter also experimented with application of the AccouMotion system in other sports such as Goalball.

Lilljedahl and Lindberg designed the DigiWall, which is a hybrid between a climbing wall and a computer game. The climbing grips are equipped with touch sensors and lights. Verbal information is used for instructions on how to play a game, scores etc. and non-verbal information is given about speed, position, direction and occurred events. Horizontal motion, for example, is indicated by changing volume balance between pairs of adjacent loudspeakers and vertical position is generally signalled through sound pitch. Lilljedahl and Lindberg implemented and tested a range of different climbing games for the DigiWall gaming environment.

Rehabilitation is another field, where acoustic feedback derived from movement data has proven useful.

Chen et al. describe their design of a real-time multimodal biofeedback system for stroke patient rehabilitation in [CHX+06]. The environment provides a purposeful engaging visual and auditory scene in which patients can practice functional therapeutic reaching tasks, while receiving different types of simultaneous feedback indicating measures of both performance and results. 12 labelled three dimensional markers are used to track the position of the patient's arm and torso with a motion capture system, produced by Motion Analysis Corporation. Visual feedback is given in the form of virtual reality and several different auditory feedback types are provided to map parameters of the motion to sound. One implementation for instance plays a sequence of notes, which follow a traditional, forward-moving musical progression that requires it be completed for the subject to hear a resolution, thus for the music to sound as if it has reached a resting point. This point is reached, when the grasping task is fulfilled. The velocity of the hand is mapped to density of notes of the musical cloud being played. For the definition of the feedback sounds the Musical Instrument Digital Interface protocol (MIDI) is used. Chen et al. carried out experiments with nine subjects (non-impaired) to test if the multimodal environments communicate the semantics of action. The trials showed that the visual-audio feedback design can guide normal subjects to do the reaching as accurately as they would in real world, while at the same time being very engaging.

Kapur et al. present a framework for sonification of Vicon motion capture data in [KTVB+05]. Their main goal was to build the necessary infrastructure in order to be able to map motion parameters of the human body to sound. The system is used to

collect and sonify different types of motion data. These include data from individuals having impairments in sensor motor co-ordination, traditional performances on musical instruments and the acting of emotions. For the sonification the following three software frameworks are used: Marsyas, traditionally used for music information retrieval with audio analysis and synthesis, CHUCK, an on-the-fly real-time synthesis language and Synthesis Toolkit (STK), a toolkit for sound synthesis that includes many physical models of instruments and sounds. Depending on the data that the system collects, the sonifications it produces have different purposes. For its application in the field of rehabilitation the system gives real-time audio feedback to the user to support the learning of moves. When data about the motion of a musician or concerning acted emotions is collected the system is used to analyse certain characteristics of the performed moves. It is for instance, used to find out which aspects of the motion have to be tracked to make a musical performance reproducible and to find out which motion data makes different emotions clearly distinguishable.

Finally, and this is the most relevant category for this project, sonification of movement data has been used for *monitoring and skill learning* of several different categories of movement

Effenberg examined the effects of movement sonification on perception and action in [Eff05]. In his research the subjects had to estimate and reproduce countermovement jumps. The jumps were performed on a Kistler force plate, which contains quartz sensors that record every change of force. The vertical component of the ground reaction force was mapped in real-time to the amplitude and frequency of sound as an electronically sampled vocal 'a'. Effenberg conducted two different studies.
For the first experiment the participants had to carry out a task under three different conditions namely with visual feedback in form of a recorded video, with acoustic feedback, which consisted of the movement sonification and with visual and acoustic feedback. In the first study the participants had to estimate the height of jumps by looking at/listening to the recording of a jump. The absolute error with visual and auditory feedback revealed no significant difference, although the perceptual accuracy was lower under the exclusive use of auditory feedback. Perception of audiovisual convergent stimuli enhanced the precision of height assessment with a significant difference to the use of both single modal conditions.
In the second experiment the participants had to reproduce jumps which they were shown on a video. This task had to be fulfilled under two different conditions namely with visual feedback and with visual and audio feedback. Again the absolute accuracy of reproduction was significantly better under convergent audio-visual conditions.

In [HHG04] Hermann, Höner and Grunow present their approach for the use of sound to assist the analysis of tactics and tactical training in sports games. For a pilot study a system for the video recording and processing (i.e. tracking of the players and the ball) of tactical training cycles in handball was set up. Sound examples for tactically correct and deviating group behaviour in game situations are provided and discussed for the example of the 6:0-defense. The system is used (at the time of the publication off-line) to give feedback to a trainer. The main task of the sonification

is to provide the coach with two sorts of information: which player shows a deviation behaviour from his nominal trajectory and in what degree. Different timbres are used to distinguish the players. A perfect tactic according to the model is silent, deviations are audible by the superposition of percussive instrument sounds.

Bovermann et al. designed 'juggling sounds', a system for real-time auditory monitoring of juggling patterns [BGdC+07]. The juggling clubs are tracked with a Vicon motion capture system. Bovermann et al. suggest the use of the system for the exploration of juggling moves, monitoring of the same and to heighten the awareness for details of movements and motions in arts performances. As an approach they use several different direct mappings, which map properties of the data to sound parameters. Thus the pattern-recognition abilities of the human listener are used for the analysis of the displayed data. The motion features that are mapped to sound parameters include streamed features such as the rotational velocity around a flipping axis of the club or the distance between the club and the head of the juggler as well as events such as the moment when a club crosses a horizontal virtual plane.

As a specific example for the sonification of complex human movement Kleinman-Weiner and Berger examine the sonification of a golf swing [KW06]. The velocity of the club head and the relative rotation of shoulders with respect to the hips (the X-factor) are considered two critical factors for the execution of a golf swing. Kleinman-Weiner and Berger introduce several different mappings for the representation of these features. Their fist method is direct mapping of the velocity to frequency and the X-factor to the spatial location of the sound (implemented with simple stereo pan). In their second approach they compare the swing with a previously built database of ideal golf swings from professional golfers. The sonification is silent during a correct swing. When the club head leaves the 'correct' plane it provides feedback at a volume and intensity proportional to its distance from the plane. The sound becomes more chaotic the more the move deviates from the optimum. Their third method uses vowel sounds that are provided by a formant synthesis, the formant filters of which are controlled by the features.

Sonification of fine motor skills was presented by Fox and Carlile in [FCB05]. Their system, called SoniMime, sonifies hand motion and works towards the goal of assisting a user to learn a particular motion or gesture with minimal deviation. As an input device SoniMime uses (wired) 3-D accelerometers to track hand gestures while an atmel microprocessor formats the serial accelerometer output into Open Sound Control (OSC) messages. To generate the output sonification, the programming language Pure Data is used. SoniMime translates acceleration data into a data stream that reflects three specific kinds of hand movement: tilt, jerk and impact. The sensor data is mapped directly to various synthesis parameters such as pitch, amplitude, playback rate or virtual location in a timbre space. Different applications of the system were suggested by Fox and Carlile including the physical implementation of the tristimulus timbre model. In this application the user can shape the sound of a synthesised vowel sound by moving her/his hands.

4 Methodology

This chapter explains the concepts behind the implementation of the closed loop audio feedback system. The system was realised as three sub-tasks, namely: data acquisition, feature extraction and sonification, which are presented in the following sections. For each of these sub-tasks different approaches were considered and are discussed here, including the method and rationale leading to the final resolution of each sub-task.
It is important to note that the three sub-tasks are interdependent, as the range of applicable sonification methods depend upon the features that can be extracted from the data, which in turn rely upon the procedures and technology used for data collection.
Section 4.1 presents a variety of sensors that were taken into consideration for the data collection. Section 4.2 introduces reasons for the choice we made concerning our sonification approaches and the features they are based on. As most other features are based on knowledge concerning the rotational angle of the wheel, it is the most important feature and the entire section 4.3 is dedicated to its computation. Section 4.4 introduces a range of further features, the derivation of which is based on the findings of section 4.3. Finally, section 4.5 explains how these features were used to produce acoustic feedback.

4.1 Data acquisition

The first step in the generation of meaningful auditory feedback for a performer on the wheel, is the collection of adequate data about the wheel's motion. The available information is linked closely to the used sensors. In this section the set of sensors that were considered for the data collection are therefore presented, and based on their properties, the choice of sensors for this project is explained.

4.1.1 Sensor options

The sensors are categorised into three groups as follows: inertial sensors, optical and acoustic sensors, and other (or combined) sensors.

<u>Inertial sensors:</u> are attached to the rolling wheel to measure its acceleration, velocity or orientation:

- *Accelerometers* measure the linear acceleration (see [Wil05]).
- *Gyroscopes* are devices that consist of a spinning mass, typically a small disc. Due to the rotation, the mass resists a change of its rotational plane. If mounted on a base so that its axis can turn freely in one or more directions, the mass maintains its orientation regardless of any movements of the base. Gyroscopes can therefore provide a reference vector, which (in world-coordinates) is fixed in two dimensions. The orientation of this vector in relation to the sensor carries information about the orientation of the sensor and thus of the wheel it is attached to. (See [Sim61].)
- *Angular rate sensors* measure the angular velocity.

Optical and acoustic sensors: are usually positioned some distance from the wheel

- *Ultrasonic range finders* transmit a short pulse of sound at an inaudible frequency (see [Loe92]). The time from transmission to echo reception gives information about the distance to an object. If located facing towards the rolling direction of the wheel in a straight line move (see section 2.1), an ultrasonic range finder can potentially find the distance to the wheel and therefore its location.
- *Infrared cameras* can be used to track infrared light. By marking the point we wish to track with reflectors and by illuminating the wheel with infrared light, the position of the point on the wheel can be tracked by the camera. [Lee07]
- *AR-Toolkit marker*[1] in combination with the ARToolKit software are commonly used for object-tracking in augmented reality applications. By attaching an enlarged version of a toolkit marker to the wheel, the wheel can be tracked in a similar way, using a single visible-light camera.
- *Motion capture systems* such as the Vicon system[2], find the 3-D position of a marker by merging the input of several cameras that are located around the object to be tracked.

Others

- *Inclinometers* measure inclination and are usually based on the measurement of the direction of gravity, which is essentially an acceleration vector[3].
- *Pressure sensors* consist of a set of interrupters that are activated when pressure is applied to them. By attaching them to the entire external perimeter of the wheel, the side of the wheel which is on the floor can be found, which in turn allows the rotational angle and the location of the wheel to be determined.
- *Magnetometers* measure magnetic fields. If not distorted by other magnetic field sources, a magnetometer attached to the wheel acts in a similar manner to a compass. It returns a vector that points towards the earth's magnetic south

[1] see `http://www.hitl.washington.edu/artoolkit/`
[2] see `www.vicon.com`
[3] see `http://en.wikipedia.org/wiki/Inclinometer`

pole, which is located close to the geographic North Pole, and is therefore parallel to an even floor. (The exact location of the earth's magnetic pole changes slowly but constantly.) The orientation of this vector in relation to the sensor provides information about the orientation of the magnetometer and of the wheel that it is attached to (for more information about magnetometers see [R+01]).

- *Artificial horizons* usually include several of the above mentioned sensors and are used in airplanes to detect the orientation of the aircraft in the space. An MT9 from xsense technologies, which combines accelerometers, angular rate sensors and magnetometers in three dimensions, is a good example of an artificial horizon (also see [BDU+99]). In combination with the appropriate software it returns the rotational angles of the sensor in three dimensions. Attached to the wheel this allows us to make a statement about the orientation of the wheel. A variety of artificial horizons that use different sensor types are commercially available.

4.1.2 Chosen sensors

As stated in section 1.2 our aim is to build a real-time auditory feedback system which, while providing a basis for our research, is also sufficiently practical and robust to support training and monitoring outside a laboratory environment. Therefore the system has to be transportable (minimum hardware, simple setup), low cost and scalable for the use with several wheels. Knowing the location and orientation of the wheel would supply us with a high amount of information about the wheel's motion. Finding this location is comparatively simple using a motion capture system, but this restrict the experiments and application to a laboratory environment. These systems are neither low cost nor easily transportable and few motion capture systems are installed in a space large enough to perform German wheel in (a minimum length of about 9m is required for advanced moves).
Most optical solutions are not adequate for our purposes, as to capture the entire length of a wheel movement, a very wide angle has to be captured, which means that the camera would have to stand relatively far away, increasing the required space and thereby significantly restricting the venues in which the system can be used.
Affordability is the main issue with combined sensors such as the MT9 and other high quality artificial horizons. Thus, even though experiments showed that an MT9 with its corresponding software can supply us with the rotational angle and hence the wheel's location, we chose not to use it for our final implementation.
Most low cost combined sensors suffer from some disadvantages or restrictions (usability in outside daylight only, for example). For our sonification system pressure sensors are not adequate because covering the entire wheel with sensors does not comply with our requirements for a simple setup. Inclinometers usually measure the earth's gravity to find the orientation of the sensor in space, however, since gravity is an acceleration vector the results are significantly distorted if the sensor itself is accelerated, which is the case if we attach it to a rolling wheel.

Inertial sensors and magnetometers are therefore the sensors that were chosen for our initial experiments. Aside from *Wii Remotes*, which include accelerometers in three dimensions, we experimented with a 3-D magnetometer. For our final implementation we use the *MicroMag 3-axis magnetometer* from *SparkFun Electronics*.

4.1.3 Sensor data and calibration

The sensors that we use (either a magnetometer, or an accelerometer as part of a Wii Remote), are attached to the wheel and thus move with it while measuring its motion. The measured magnetic field or acceleration vector (see Figure 4.1) consists of three components:

- the x-component that is tangential to the wheel
- the z-component, which is centrifugal towards the wheel centre
- and the y-component, which is orthogonal to the wheel's rolling direction and the other two components.

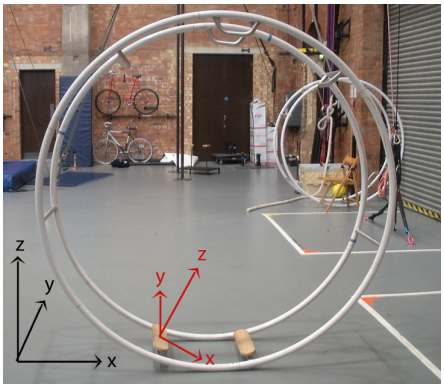


Figure 4.1: World coordinate system and sensor coordinate system at a rotational angle of $\varphi = 0$

Thus the inputs of all calculations described in the following sections are 3-D vectors of either magnetic field or acceleration data. The *Wii Remote* and the *MicroMag3* magnetometer are the sensors that were used to collect this data. They are presented in more detail in section 5.2.

For both sensor types experiments showed that the measured values contain different offsets for the three axes. To avoid an adverse effect on the results, this 3-D offset vector has to be subtracted from the input data before other calculations are done. Taking the x-axis as an example we explain how to determine the offset for each axis. A 'calibration line' and its direction are determined for example by drawing an arrow onto the surface which the experiment is conducted on. To obtain two calibration values x_+ and x_- the value of the x-axis is measured twice: once with the axis aligned

to the calibration line to measure x_+ and once with the axis antipodal to the line to measure x_-. The mean value of x_+ and x_- provides the offset of the x-axis. The same procedure is repeated for the y and z-axis.

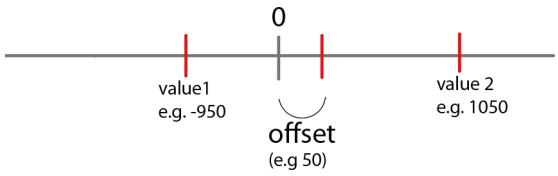


Figure 4.2: Offset calculation

The concept behind this calibration is comprehensive if we consider an example (see Figure 4.2). If the axis has no offset, the first and second value differ only in their sign, because the same magnetic field or acceleration is measured in opposite directions. The values measured could for example be $x_+ = 1000$ and $x_- = -1000$. In this case the average of the values results in zero. If the axis has an offset its entire scale, and therefore both values, are shifted by the offset, which leads to the values $x_+ = 1050$ and $x_- = -950$ for an x- offset of 50. Their average $\frac{x_+ + x_-}{2} = \frac{1050+(-950)}{2} = 50$ results in the offset.

4.2 Reasoning about sonification approaches and adequate features

The aim of our sonification system is to provide auditory real-time feedback to support the execution of German wheel moves. For the generation of an effective feedback the design of this acoustic feedback and the choice of features it is based on are essential. This section motivates why the four different sonification approaches that we implemented were chosen. (These are discussed in more detail in section 4.5). Three main questions form the basis for our reasoning:

- What level of information do we want the sonification to contain?
- Which features do we need to represent this information?
- How do we implement the acoustic representation of this information?

The following three subsections answer these three questions.

4.2.1 Level of adaptation

After a close observation of the purpose, which the sonification is supposed to support, we found that the question about the information to be represented mainly

concerns the level of adaptation of the sonification to the task a wheel performer has to fulfil. On one end of the spectrum lies a sonification directly from the data, which renders the data acoustically without interpreting its meaning. On the other end of the spectrum is a sonification, that adapts highly to the use of sonification for a wheel move by expressing exactly the elements that are critical for this specific move. (Section 2.1 contains a discussion of different wheel moves and the aspects that are important for their performance). The optimal approach depends on the application of the sonification system. We therefore chose to implement the following approaches from different parts of the spectrum.

The *Direct-data approach* has the lowest level of adaptation. It directly sonifies the magnetometer data and is the most flexible approach. It is not restricted to the use for German wheel motion sonification, but can be used for the acoustic representation of any 3-D data, including the data received from sensors other than magnetometers. The analysis of the data is left completely to the user of the system. For the German wheel application the low level of adaptation to a specific task may lead to a less intuitive auditory feedback than other semantically more meaningful approaches.

The next level of adaptation is implemented in the *Cartoonification approach*. The basic idea behind it is to implement an approach that imitates the sound that could naturally be expected from a rolling wheel. This design is restricted to the use for German wheel applications. It does, however, implement a general feedback about the wheel's motion without adapting much to one specific wheel move.

Even more adapted to certain characteristics of wheel moves are the remaining two approaches that we implemented, namely the *Vowel synthesis sonification* and the *Event-based sonification*. Their representation of the data adapts far more to the characteristics of the straight line German wheel moves, which are listed in section 2.1. Neither the Vowel synthesis sonification nor the Event-based sonification is located at the end of the continuum however, because they are adapted to categories of German wheel moves rather than to one specific move. (Otherwise the use of the system would be very restricted.) The Vowel synthesis approach and the Event-based approach differ in their way of representing certain features acoustically. More information about this is given in subsections 4.5.3 and 4.5.4.

4.2.2 Features and their categorisation

This subsection presents our reasoning about the features that are necessary to allow the different levels of adaptation described in the previous subsection. Depending on the level of adaptation different features are required for the acoustic representation. The Direct-data approach requires the original input data as its only feature. The parameter which influences the natural sound of a wheel most is the wheel's (angular) velocity, which therefore is the main feature of the Cartoonification approach. Further features arose from the considerations about the specific tasks of the performer when the wheel move is carried out (see section 2.1).

Following is a list of all features which were used for the different sonification approaches. The features are divided into two groups according to their properties: Certain features supply continuous information about the wheel's motion, whilst other features occur at discrete time-points. We suggest the following features:

continuous features:

- The *input data* itself can be used, leaving the interpretation to the user rather than implementing a semantically meaningful approach.
- The *rotational angle* φ also gives information concerning the location of the wheel and is the most important feature as all following features are based on it.
- The *angular velocity* ω, gives intuitive feedback about the motion of the wheel.
- The *direction of the velocity* gives additional information.

discrete features:

- The moment when the bar passes the zero axis (see Figure 4.3) on top of the wheel and reaches a *highest point* indicates the correct timing for many onward moves (see section 2.1).
- The moment when one of the bars passes the zero axis on the bottom, reaches a *lowest point* and touches the floor: Besides an additional guide to orientation this feature implies when force has to be applied in some moves (some straight line decentralised moves in particular).
- A *change of rolling direction and its timing* provide information about the timing when a returning move should be performed.
- The *rotational angle of the wheel when it changes its rolling direction* gives information about the location of the bars at the time. This again shows whether the bar was brought to an adequate height for the respective move.
- The *time difference* (t_{diff}) between two changes of rolling direction: In all the moves listed in section 2.1 the wheel exhibits a certain inertia. Two meaningful changes of rolling direction don't usually occur consecutively in a short time span (t_{diff}). A small time difference therefore indicates that a change of rolling direction is caused by a small jerk applied to the wheel or an inaccuracies in the data rather than by an intentionally performed move. When t_{diff} is very small a change of rolling direction is therefore seen as less significant.
- The *average (angular) velocity* $|\omega_{av}|$ *between two changes of rolling direction* contributes to the detection of the same significance. The timing of two changes of rolling direction can show a large difference. However, if the average velocity between points is very small, the wheel has moved very little and it is likely that the change of rolling direction is of no significance to the performer.

For each of our sonification approaches we used a different selection of the above features. To support our reasoning concerning relevant data to be displayed and the

application of appropriate sonification techniques, we have carried out a Task Data Analysis (TaDa). A TaDa-analysis summarises the task, data and information to be extracted from the data in a methodological way. It informs the specification of information that is to appear in the display [Bar98]. The complete TaDa-analysis is presented in appendix B.

4.2.3 Acoustic representation

The last aspect we want to reflect about is the acoustic representation of the features listed in the previous subsection. As described in section 3.1 there is a range of existing sonification techniques. To choose a technique that is adequate for our purpose, we used the Sonification Design Space Map as introduced by Alberto de Campo before we implemented the system (see [dC07]). The Sonification Design Space Map allows systematic reasoning about the application of different sonification techniques such as Earcons, Auditory Icons, Parameter Mapping or Model-based Sonification (see section: 3.1). The Sonification Design Space Map suggests different sonification techniques for sonification tasks, based on the following three features:

- the data dimensionality
- the number of sonification streams
- the number of data points needed for perception of an auditory gestalt. In psychoacoustics an auditory gestalt is a part of the sound that is perceived as an entity (e.g. [CW92]). Normally this is roughly the number of data points represented acoustically in 1-3 seconds (see: [dC07], [Sny00]).

The time scale determines how many data points form an auditory gestalt and thus is one of the most fundamental design decisions. In our implementation the time scale is set by the need for real-time feedback. Our data stream has a sample rate of about 20 Hz (50-100Hz for the approaches that use acceleration data) and we predicted a dimensionality of 3 or 4 (e.g. location, velocity, and acceleration). This places our task in an area of the Sonification Design Space map, which suggests a continuous feedback production, for example by parameter mapping.

Using this indication as a starting point for our sonification, we started designing and implementing the audio feedback system. Further analysis introduced the two categories of features that were mentioned in subsection 4.2.2, namely continuous features and discrete features. Features such as the angular velocity are continuous data, and its change over time is highly relevant for the performer. Therefore its variation is usually represented as the temporal change of acoustic attributes of a continuous sound stream. Other features such as the changes of rolling direction are discrete events and can therefore be adequately represented by discrete sound events. For the three domain specific sonifications Cartoonification, Vowel synthesis sonification and Event-based sonification this led to an auditory feedback that is composed of different layers, namely a *continuous rolling sound* layer and one or more sound

event layers. All sound layers were implemented using the different parameter mapping techniques that are presented in section 3.1. The mappings of the data features to sound features are discussed for each approach in section 4.5.

4.3 Finding the wheel's rotational angle

Based on the tasks a wheel performer has to fulfil subsection 4.2.2 introduced a list of features that appear a good choice for the sonification of straight line German wheel moves (see section 2.1). The extraction of the input data as a features is trivial. All other listed features can be calculated, if the rotational angle of the wheel is known. (Due to the properties of a rolling wheel, which are described in appendix A.1 the rotational angle also gives information about the location of the wheel.) This makes the rotational angle of the wheel the most important of the listed features. Therefore this section focuses on the different approaches we have considered for the computation of the wheel's rotational angle. In subsection 4.3.1 our final approach to the calculation of the rotational angle of the wheel, for which the output of a 3-D magnetometer is used as input, is presented. In the consecutive subsections we discuss several other approaches which were tried before achieving to calculate the rotational angle of the wheel from the magnetometer data. For these approaches one or two accelerometers were used as input devices. Appendix C presents two additional approaches, which are based on acceleration data, but which were not fully implemented.

4.3.1 Magnetometer approach

The magnetometer approach uses data collected by a magnetometer that is attached to the wheel. The definition of the sensor-coordinate system is described in subsection 4.1.2 and can be seen in Figure 4.1. If not distorted by other magnetic sources, the magnetometer senses the earth's own magnetic field and acts similar to a compass: It returns a vector $\vec{v}$ pointing towards the earth's magnetic south pole, which is close to the geographic North Pole. In world coordinates this vector is constant, while the sensor's coordinate system rotates with the wheel.

In a straight line move the wheel rotates around the y-axis. In other words the x- and z-axis of the sensor rotate within the world coordinate x/z-plane. The angle by which the x- or the z-axis has rotated in this plane is also the angle by which the wheel has rotated. To assign a value to this rotational angle of the wheel, we need a constant vector in the x/z-plane of the world coordinate system as a reference to which we calculate the angle. As $\vec{v}$ is constant, its projection $\vec{v}_{xz}$ onto the x/z-plane is also constant and we can use it as a reference. The angle between $\vec{v}_{xz}$ and the x-axis of the sensor in world-coordinates provides the rotational angle of the wheel. Due to our definition of the world- and sensor coordinate systems (see subsection 4.1.3), the x/z-plane is 'the same' in sensor and world coordinates. Therefore the angle between

$\vec{v}_{xz}$ and the sensor's x-axis in world-coordinates equals the angle between $\vec{v}_{xz}$ and the sensor's x-axis in sensor coordinates. This angle can easily calculated by applying the $\arctan 2$, which returns the angle between a point (here $\vec{v}_{xz} = \begin{pmatrix} v_x \\ v_z \end{pmatrix}$) and the x-axis of the coordinate system:

$$\varphi^* = -\arctan 2(v_x, v_z) \tag{4.1}$$

This calculation returns values between $-\pi$ and π (see Figure 4.3).

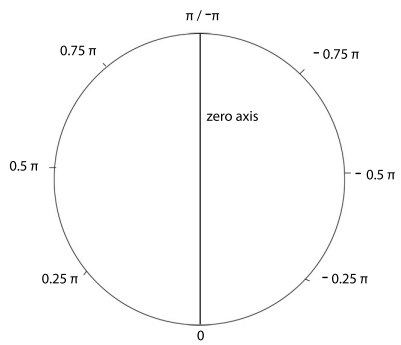


Figure 4.3: Range of the rotational angles of a bar: there is a jump from π to $-\pi$ when the bar reaches the top.

Here the sign of the rotational angle complies with the right hand rule: if the thumb of the right hand points in the direction of the rotation axis (here the y-axis, which points rearwards) the curved fingers indicate a positive direction of rotation.

Due to factors such as an inclined floor and the position the sensor is attached in, this calculation does not necessarily return a rotational angle of $\varphi = 0$ for the initial position described in section 2.2. As indicated in that section the initial angle is therefore calculated by applying formula 4.1 when the wheel is in the initial position, in which the rotational angle should be $\varphi = 0$. The result $\varphi_{initialPosition}$ is treated as an offset, which is subtracted from all calculated angles:

$$\varphi = \varphi^* - \varphi_{initialPosition} = -\arctan 2(v_x, v_z) - \varphi_{initialPosition} \tag{4.2}$$

When this calibration is used, it is important to notice that the subtraction can lead to values larger than π or smaller than $-\pi$. To transform them back into the desired range the following algorithm has to be applied:

CASE :$\lvert\varphi\rvert <= \pi$	angle is in range anyway
$\varphi_{new} = \varphi$	
CASE: $\varphi > \pi$	
$\varphi_{excess} = \varphi - \pi$	φ_{excess} is the value by which φ exceeds π
$\varphi_{new} = -\pi + \varphi_{excess}$	

CASE: $\varphi < -\pi$

$\varphi_{excess} = \varphi + \pi$ $\qquad$ φ_{excess} is negative here

$\varphi_{new} = \pi + \varphi_{excess}$

In future when we refer to an angle (e.g. φ), we presume that it has been formatted in this way.

Potentially the magnetometer approach may fail in case the magnetic field vector $\vec{v}$ of the earth is exactly parallel to the sensor's y-axis. As the wheel rolls in a line orthogonal to this axis, the projection of $\vec{v}$ onto the x/z-plane would always be $\vec{0}$ in that case. In practice we have never experienced this problem and it can easily be solved by changing the wheel's rolling direction. A general advantage of the magnetometer approach is its extensibility. Even though we did not implement it for this project, the principle theoretically allows the wheel's orientation in three dimensions to be found.

4.3.2 Simple integration

Our first experiments with inertial sensors were carried out using an accelerometer. A double integration seems to be the obvious way to find the location of the accelerometer and thus the wheel, since the acceleration is the second derivative of the location. Unfortunately there are several reasons why this simple approach is not applicable. In a real situation, where the time step between the reception of two data items can not be infinitely small, an error is generated and accumulates over time, causing significant drift. Even advanced integration techniques, which use several time-steps to minimise this error (such as the Runge Kutta method [PV92]) cannot keep the drift sufficiently small to make this approach reliable. Another reason for the failure of this approach is that gravity causes an acceleration as well and always contributes to the overall measured acceleration (see appendix A.3). As the accelerometer moves with the wheel, it is not possible to identify the change of velocity component of the measured acceleration that is generated other than by gravity. To address this problem, several approaches using two accelerometers instead of one were tested and are discussed in the following subsections.

4.3.3 Differential equation approach

The physical properties of a rolling motion as presented in appendix A.1 are the starting point for this bottom-up approach of calculating the wheel's location. Equations for the location of two accelerometers that are attached to two points on the wheel's circumference are set up. The two accelerometers are attached at an angle of β between each other. The formulas for the locations of accelerometer 1 and accelerometer 2 are derived twice to obtain formulas for the acceleration the sensors

should experience. As discussed in detail below, the measured accelerations are inserted into these formulas to compute relevant variables such as the rotational angle of the wheel.

Considerations are made according to the following world-coordinate system (see Figure 4.4): The x-axis of the right-hand coordinate system points into the rolling

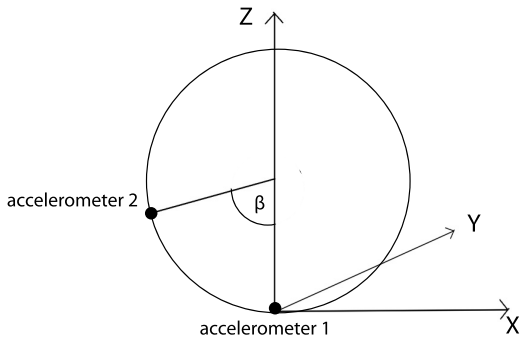


Figure 4.4: The initial position of the wheel and the sensors in the differential equation approach in reference to the world coordinate system: accelerometer 1 and 2 start at a rotational angle of $\varphi = 0$ and $\varphi + \beta$

direction of the wheel, the y-axis points rearwards and the z-axis upwards. The x/y-plane represents the floor and the z-axis represents the distance from the floor. The location of the wheel is represented by the location of its centre $\vec{m}$, whose starting position is $\vec{m}(0) = \begin{pmatrix} 0 \\ 0 \\ r \end{pmatrix}$, with r being the radius of the wheel.
As the wheel centre only moves in x-direction, the y and z-value of $\vec{m}$ are constant. The formula for an accelerated motion, known from Newtonian mechanics, provides the following equation for the location of the wheel centre depending on the time:

$$\vec{m}(t) = \begin{pmatrix} m_x(t) \\ m_y(t) \\ m_z(t) \end{pmatrix} = \begin{pmatrix} m_x(0) + V \cdot t + \frac{1}{2} \cdot a \cdot t^2 \\ 0 \\ r \end{pmatrix} \tag{4.3}$$

V is the velocity of the wheel centre, a its acceleration and $m_x(0) = 0$. We can not measure the value of a, as it is not possible to attach an accelerometer to the wheel's centre.

To include the measured acceleration into the equation, a formula for the location of the sensors is required. For this and the following approach we presume that the rotational angle of the wheel φ is zero when accelerometer 1 is on the floor. The trigonometric functions lead to the location $\vec{w}_1$ of a accelerometer 1 (see Figure 4.5):

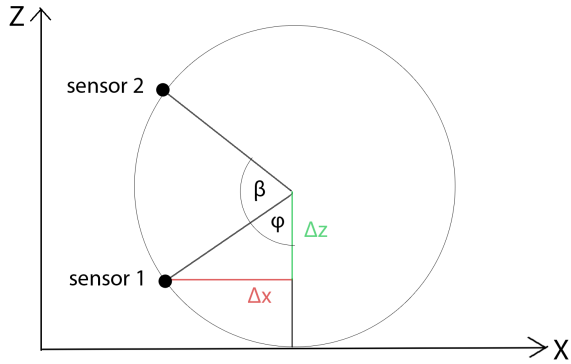


Figure 4.5: Determination of the sensors' location in reference to the wheel's centre; the trigonometric functions provide: $\sin(\varphi) = \frac{\Delta x}{r} \rightarrow \Delta x = r \cdot \sin(\varphi)$, $\cos(\varphi) = \frac{\Delta z}{r} \rightarrow \Delta z = r \cdot \cos(\varphi)$.

$$\vec{w}_1(t) = \vec{m}(t) + \begin{pmatrix} -r \cdot \sin(\varphi) \\ 0 \\ -r \cdot \cos(\varphi) \end{pmatrix} \tag{4.4}$$

Due to the trajectory the accelerometers travel on, the rotational angle φ and the x-coordinate of the wheel centre $m_x(t)$ have the relation $\varphi = \frac{m_x(t)}{r}$ (see appendix A.1).
If we use formula 4.3, we therefore get the following formulas for φ and its first derivative:

$$\varphi = \varphi_0 + \frac{m_x(t)}{r} = \varphi_0 + \frac{m_x(0) + V \cdot t + \frac{1}{2} \cdot a \cdot t^2}{r}$$

$$\varphi' = \frac{V + a \cdot t}{r}$$

To retrieve a formula, which contains the acceleration measured by accelerometer 1, we derive the location of the accelerometer in formula 4.4 twice: First we insert $\vec{m}(t)$ and derive the resulting formula to retrieve the velocity of accelerometer 1:

$$\vec{w}_1(t) = \begin{pmatrix} m_x(0) + V \cdot t + \frac{1}{2} \cdot a \cdot t^2 - r \cdot \sin(\varphi) \\ 0 \\ r - r \cdot \cos(\varphi) \end{pmatrix}$$

$$\vec{V}_1 = \vec{w}_1'(t) = \begin{pmatrix} V + a \cdot t - \frac{V + a \cdot t}{r} \cdot r \cdot \cos(\varphi) \\ 0 \\ \frac{V + a \cdot t}{r} \cdot r \cdot \sin(\varphi) \end{pmatrix}$$

$$= \begin{pmatrix} V + a \cdot t - V \cdot \cos(\varphi) - a \cdot t \cdot \cos(\varphi) \\ 0 \\ V \cdot \sin(\varphi) + a \cdot t \cdot \sin(\varphi) \end{pmatrix}$$

a further derivation leads to the acceleration of accelerometer 1:

$$\vec{a}_1 = \vec{V}_1' = \vec{w}_1''(t) = \begin{pmatrix} a + V \cdot \sin(\varphi) \cdot \frac{V+a\cdot t}{r} - (a \cdot \cos(\varphi) + a \cdot t \cdot \frac{V+a\cdot t}{r} \cdot (-\sin(\varphi)) \\ 0 \\ \frac{V+a\cdot t}{r} \cdot V \cdot \cos(\varphi) + a \cdot \sin(\varphi) + a \cdot t \cdot \cos(\varphi) \cdot \frac{V+a\cdot t}{r} \end{pmatrix}$$

$$= \begin{pmatrix} a + \frac{V^2}{r}\sin(\varphi) + \frac{a\cdot t\cdot V}{r} \cdot \sin(\varphi) - a \cdot \cos(\varphi) + \frac{a\cdot t\cdot V}{r} \cdot \sin(\varphi) + \frac{a^2\cdot t^2\cdot}{r} \cdot \sin(\varphi) \\ 0 \\ \frac{V^2}{r}\cos(\varphi) + \frac{a\cdot t\cdot V}{r} \cdot \cos(\varphi) + a \cdot \sin(\varphi) + \frac{a\cdot t\cdot V}{r} \cdot \cos(\varphi) + \frac{a^2\cdot t^2}{r} \cdot \cos(\varphi) \end{pmatrix}$$

The aim is to insert the measured values. We are therefore only interested in the instantaneous values and can set $t = 0$, which leads to the following formula:

$$\vec{a}_1 = \begin{pmatrix} a + \frac{V^2}{r}\sin(\varphi) - a \cdot \cos(\varphi) \\ 0 \\ \frac{V^2}{r}\cos(\varphi) + a \cdot \sin(\varphi) \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \tag{4.5}$$

The second summand of formula 4.5 is caused by the influence of the earth's gravity (see appendix A.3). The formula describes the acceleration of sensor 1 in world coordinates.

The data however is measured in relation to the sensor's own coordinate system, which is moving with the wheel. Thus to equate the measured data with the theoretically deduced formula 4.5 we have to transform the measured vector into world-coordinates by rotating it through the rotational angle of the wheel about the y-axis. The additional translation the sensor experiences does not have any influence on the transformation, because it is a constant that drops out in the above described double derivation. If $\vec{a}_1^* = \begin{pmatrix} a_{1x}^* \\ a_{1y}^* \\ a_{1z}^* \end{pmatrix}$

is the measured acceleration of accelerometer 1, this is transformed into world-coordinates as follows:

$$\vec{a}_1 = \begin{pmatrix} \cos(\varphi) & 0 & \sin(\varphi) \\ 0 & 1 & 0 \\ -\sin(\varphi) & 0 & \cos(\varphi) \end{pmatrix} \cdot \begin{pmatrix} a_{1x}^* \\ a_{1y}^* \\ a_{1z}^* \end{pmatrix} \tag{4.6}$$

The next step is to equate the acceleration that should theoretically be measured (formula 4.5) and the acceleration that is measured transformed into world-coordinates (formula 4.6). This leads to:

$$\begin{pmatrix} a + \frac{V^2}{r}\sin(\varphi) - a \cdot \cos(\varphi) \\ 0 \\ \frac{V^2}{r}\cos(\varphi) + a \cdot \sin(\varphi) + g \end{pmatrix} = \begin{pmatrix} \cos(\varphi) \cdot a_{1x}^* + \sin(\varphi) \cdot a_{1z}^* \\ a_{1y}^* \\ -\sin(\varphi) \cdot a_{1x}^* + \cos(\varphi) \cdot a_{1z}^* \end{pmatrix} \tag{4.7}$$

The radius of the wheel r and the gravity constant g are known, so row 1 and 3 of this formula provide us with 2 equations that include three unknown variables: the

acceleration a and velocity V of the wheel centre and the rotational angle φ of the wheel. For the second accelerometer, which is attached at an angle β in relation to accelerometer 1 (see Figure 4.5) we also obtain 2 equations by replacing φ with $\beta + \varphi$ in formula 4.7. Setting the angle between the two accelerometers to $\beta = \pi$ (sensors are attached at opposite sides of the wheel) and using the properties of the trigonometric functions: $\cos(\varphi + \pi) = -\cos(\varphi)$ and $\sin(\varphi + \pi) = -\sin(\varphi)$ results in the following 4 formulas (in which $\sin = \sin(\varphi)$ and $\cos = \cos(\varphi)$):

<u>accelerometer 1:</u>

$$1) \qquad a + \frac{V^2}{r}\sin - a \cdot \cos = \cos \cdot a^*_{1x} + \sin \cdot a^*_{1z}$$

$$2) \qquad \frac{V^2}{r}\cos + a \cdot \sin + g = -\sin \cdot a^*_{1x} + \cos \cdot a^*_{1z}$$

<u>accelerometer 2</u>:

$$3) \qquad a - \frac{V^2}{r}\sin + a \cdot \cos = -\cos \cdot a^*_{2x} - \sin \cdot a^*_{2z}$$

$$4) \qquad -\frac{V^2}{r}\cos - a \cdot \sin + g = \sin \cdot a^*_{2x} - \cos \cdot a^*_{2z}$$

out of these equations follows:

$$2) + 4) \Longrightarrow 2g = \underbrace{(a^*_{2x} - a^*_{1x})}_{d_1} \cdot \sin + \underbrace{(a^*_{1z} - a^*_{2z})}_{d_2} \cdot \cos$$

To solve this equation and find φ the substitution $\sin(\varphi) = x$ is used as well as the correlation between sinus and cosine: $\sin^2 + \cos^2 = 1 \rightarrow \cos = \sqrt{1 - \sin^2}$, out of which follows:

$$2g = d_1 \cdot x + d_2 \cdot \sqrt{1 - x^2}$$

Transposition results in

$$0 = x^2 \underbrace{- \frac{4gd_1}{d_1^2 + d_2^2}}_{p} \cdot x + \underbrace{\frac{4g^2 - d_2^2}{d_1^2 + d_2^2}}_{q}$$

which is solved by:

$$x_{1/2} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$$

However, when we applied this formula to real data to calculate $x = \sin(\varphi)$, negative values were produced under the square root, making the equation insolvable. We searched our method of resolution for possible sources of error and found the following: for our experiments we worked on data that was collected with the accelerometers in two Wii Remotes, which do not send at a fixed rate. The missing synchrony of the two data streams has to be effectuated. Moreover the data measured by the Wii Remotes has a slight offset, which can differ for each device and

data axis. A calibration as described in section 4.1.3 has to be done.
We did not implement these improvements in our sonification system because at that state of the project the experiments with a magnetometer were proving to be successful and we were able to detect the rotational angle of the wheel using a magnetometer as described in subsection 4.3.1. The above suggestions can be considered as possible avenues for improvement.

4.3.4 Vector approach

This approach is based on the following considerations for two accelerometers attached to the wheel on opposite sides ($\beta = \pi$): The acceleration each sensor experiences consists of different components, which are caused by the rotational part of the motion, the translational part of it and gravity (see Figure 4.6):

- The tangential acceleration a_t is the part of the acceleration that is caused by the pure rotation, which is tangential to the wheel's circumference.
- The centripetal acceleration a_z is also caused by the rotation and points towards the centre of the wheel.
- The translational acceleration a_{tr} is caused by the displacement of the wheel in space and thus is parallel to the floor.
- Gravity contributes to the overall acceleration as described in appendix A.3. This component points away from the floor and is orthogonal to a_{tr} and the floor.

If the sensors are attached to the wheel as described in subsection 4.1.2, a_t contributes to the acceleration measured on the x-axis of the sensor and a_z to the value measured on the z-axis. As g remains orthogonal and a_{tr} parallel to the floor, but the sensor rotates with the wheel, they contribute to the measured x and z value depending on the rotational angle of the wheel. The rotational components a_t and a_z have the same value on any point of the wheel's circumference.

If the motion of the wheel was a pure rotation, gravity would be the only difference between the data measured by accelerometer 1 and accelerometer 2. Due to the angle π between the coordinate systems of accelerometer 1 and 2, gravity would point in opposite directions in the two systems. Therefore the accelerations of sensor one and two would be

$$\vec{a}_1^* = \vec{a}_t + \vec{a}_z + g$$

$$\vec{a}_2^* = \vec{a}_t + \vec{a}_z - g$$

Subtraction leads to $\vec{a}_1^* - \vec{a}_2^* = 2g$. Knowing g gives information about the direction of the floor in relation to the coordinate systems of the sensors and thereby about the rotational angle of the wheel.

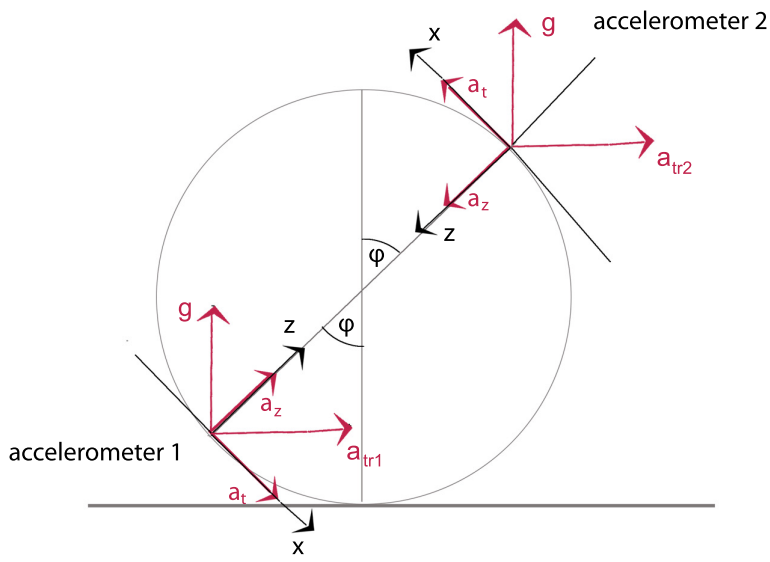


Figure 4.6: Components of data measured by the accelerometer: tangential acceleration a_t, centripetal acceleration a_z, translational acceleration a_{tr} and contribution of gravity g

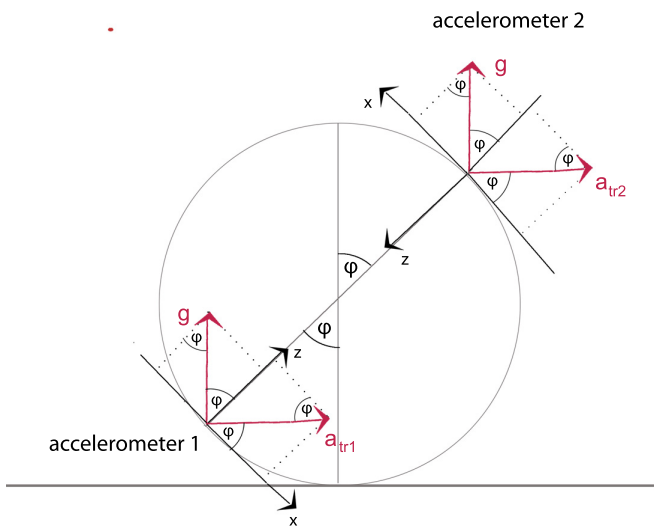


Figure 4.7: The projections of g and a_{tr} onto the x- and z- axes contribute to the acceleration measured in these directions.

In a normal rolling motion, however, the additional component a_{tr} contributes to the data a_1^* and a_2^* which are measured by accelerometer 1 and 2. For the measured x- and z- constituents of the two accelerometers we get (see Figure: 4.7): <u>accelerometer 1:</u>

$$1) \qquad a_{1x}^* = a_t - \sin(\varphi) \cdot g + \cos(\varphi) \cdot a_{tr1}$$

$$2) \qquad a_{1z}^* = a_z + \cos(\varphi) \cdot g + \sin(\varphi) \cdot a_{tr1}$$

<u>accelerometer 2</u>:

$$3) \qquad a_{2x}^* = a_t + \sin(\varphi) \cdot g - \cos(\varphi) \cdot a_{tr2}$$

$$4) \qquad a_{2z}^* = a_z - \cos(\varphi) \cdot g - \sin(\varphi) \cdot a_{tr2}$$

The centre of the wheel experiences only translation, which is the average of the translations on the rim:

$$5) \qquad a = \frac{a_{tr1} + a_{tr2}}{2}$$

Even though this provides only 5 formulas with 6 unknown variables $(\varphi, a, a_{tr1}, a_{tr2}, a_z, a_t)$, we can solve the equation owing to the correlation between some of the variables as follows:

$$1) - 3) \qquad \overbrace{a_{1x}^* - a_{2x}^*}^{-d_1} = -2\sin(\varphi) \cdot g + \cos(\varphi) \cdot (a_{tr1} + a_{tr2})$$

$$2) - 4) \qquad \underbrace{a_{1z}^* - a_{2z}^*}_{d_2} = 2g\cos(\varphi) + \sin(\varphi) \cdot (a_{tr1} + a_{tr2})$$

We insert $a_{tr1} + a_{tr2} = 2a$, which formula 5 leads to and achieve:

$$6) \qquad -d_1 = -2\sin(\varphi) \cdot g + 2a \cdot \cos(\varphi)$$

$$7) \qquad d_2 = 2\cos(\varphi) \cdot g + 2a \cdot \sin(\varphi)$$

We transpose 6) to get a:

$$a = \frac{-d_1 + 2g \cdot \sin(\varphi)}{2 \cdot \cos(\varphi)}$$

and insert it into 7):

$$d_2 = 2\cos(\varphi) \cdot g + \frac{-d_1 + 2g \cdot \sin(\varphi)}{\cos(\varphi)} \cdot \sin(\varphi)$$

The substitution of $\sin(\varphi) = x$ and $\cos(\varphi) = \sqrt{1 - x^2}$ under usage of the correlation between sinus and cosine: $1 = \cos(\varphi)^2 + \sin(\varphi)^2$ and subsequent transpositions lead to:

$$d_2 = 2g \cdot \sqrt{1 - x^2} + \frac{-d_1 x + 2gx^2}{\sqrt{1 - x^2}}$$

$$\rightarrow d_2 \cdot \sqrt{1 - x^2} = 2g \cdot (1 - x^2) - d_1 x + 2gx^2 = 2g - d_1 x$$

$$\rightarrow d_2^2(1 - x^2) = 4g^2 - 4gd_1 x + d_1^2 x^2$$

$$\rightarrow 0 = (d_1^2 + d_2^2)x^2 - 4gd_1x + 4g^2 - d_2^2$$

$$\rightarrow 0 = x^2 - \frac{4gd_1}{d_1^2 + d_2^2}x + \frac{4g^2 - d_2^2}{d_1^2 + d_2^2}$$

This is the same formula as found in subsection 4.3.3 and therefore leads to the same results.

4.4 Features derived from the rotational angle

As indicated in the introduction of this chapter all features listed in section 4.2.2 can be computed if the input data and the rotational angle of the wheel are known. This section gives some complementary information concerning the rotational angle and describes in detail how the other features are derived from it.

4.4.1 Rotational angle $\varphi(t)$

Except for the direct-data values, all features are derived from the *rotational angle* of the wheel, the calculation of which is discussed in chapter 4.3.1. In some cases we are interested in the rotational angle of one of the bars rather than the rotational angle of the reference points between the foot-plates (e.g. when we want to find the highest or lowest points of a bar, see section 2.1). The angle of each bar can be calculated by adding or subtracting the fixed angle α_{bar} between the reference point and the bar to each value that is calculated for the rotational angle of the wheel (see Figure 2.2). To ensure that its values lie within the range of $-\pi$ to π, the resulting angle has to be brought into the right format as described in subsection 4.3.1.

$$\varphi_{bar1}(t) = \varphi(t) + \alpha_{bar}$$

$$\varphi_{bar2}(t) = \varphi(t) - \alpha_{bar}$$

$$\varphi_{footplate1}(t) = \varphi(t) + \alpha_{footplate} \cdots$$

4.4.2 Angular velocity

The angular velocity ω is the first derivative of the rotational angle of the wheel. To numerically differentiate the angle, we use the following formula, which is explained in more detail in appendix A.2:

$$\omega = \frac{\Delta\varphi}{\Delta t} = \frac{\varphi(t_n) - \varphi(t_{n-k})}{t_n - t_{n-k}} \tag{4.8}$$

For a better comprehensibility the following explanations presume that two successive values of the angle are used for the calculation of the derivation (k=1).

Our implementation for this numerical differentiation can be used for linear as well as for angular arguments even though the detailed underlying algorithm is different for the two cases. In case of angular input the values range from $-\pi$ to $+\pi$. A case differentiation has to be made to avoid errors caused by the jump of the angle between π and $-\pi$ or vice versa (see Figure 4.3). This jump causes a misinterpretation in the computation of $\Delta\varphi$ in formula 4.8. $\Delta\varphi$, which we want to calculate, is the angle between the previous angle $\varphi(t_{n-1})$ and the current angle $\varphi(t_n)$. In other words it is the angle we have to rotate by to get from $\varphi(t_{n-1})$ to $\varphi(t_n)$. A rotation from $\varphi(t_{n-1})$ to $\varphi(t_n)$ is ambiguous however, as we can either rotate by $\Delta\varphi$ or in opposite direction by its opposite angle $\varphi_{opposite} = \pi - |\Delta\varphi|$ (see Figure 4.8). In our application the

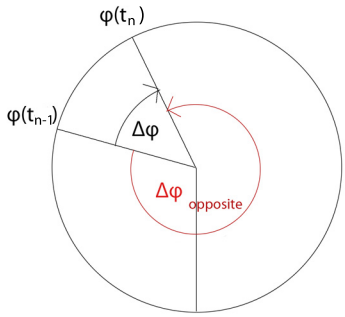


Figure 4.8: The ambiguity of a rotation from $\varphi(t_{n-1})$ to $\varphi(t_n)$. A rotation by $\Delta\varphi$ or by $\Delta\varphi_{opposite}$ lead to the same result.

time steps and thus the change of the angle per time step are relatively small. Hence we are generally interested in the smaller of the two angles. If the 'jumping point' from π to $-\pi$ or vice versa is not passed in the time between the two measurements of the angle at t_{n-1} and t_n, $\Delta\varphi = \varphi(t_n) - \varphi(t_{n-1})$ results in this smaller angle. If the 'jumping point' is passed between t_{n-1} and t_n however, $\Delta\varphi$ returns the larger of the two angles $|\Delta\varphi| = |\varphi(t_n)| + |\Delta\varphi(t_{n-1})|$ and the angle of interest is $\Delta\varphi_{opposite}$ (see Figure 4.9).

$|\Delta\varphi|$ and $|\varphi_{opposite}|$ add up to 2π, leading to the conclusion that the absolute value of the smaller angle has to be smaller than π. Thus we can find the smaller angle by requesting that $|\Delta\varphi| < \pi$. If this is not the case, $\varphi_{opposite}$ is used instead. If $\Delta\varphi_{opposite}$ is chosen, we rotate the opposite way to get from $\varphi(t_{n-1})$ to $\varphi(t_n)$ and therefore $\varphi_{opposite}$ is given the opponent of $\Delta\varphi$'s sign.
This leads to the following case differentiation over the size and sign of $\Delta\varphi$:

$$\begin{aligned}
&\underline{\text{CASE:}}\ -\pi < \Delta\varphi < \pi \\
&\qquad \Delta\varphi_{new} = \Delta\varphi = \varphi(t_n) - \varphi(t_{n-1}) \\
&\underline{\text{CASE:}}\ \Delta\varphi > \pi \qquad\qquad \Delta\varphi \text{ is positive, its sign is: } + \\
&\qquad \Delta\varphi_{new} = \overbrace{-}^{opposite\ sign} (2\pi - |\Delta\varphi|) \qquad \Delta\varphi \text{ is positive} \rightarrow |\Delta\varphi| = \Delta\varphi
\end{aligned}$$

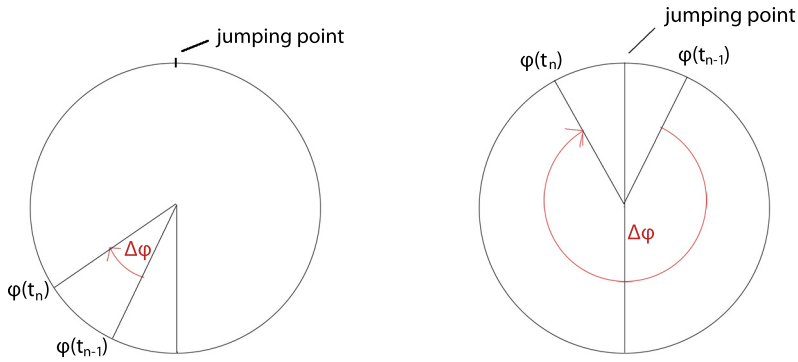


Figure 4.9: The value of $|\Delta\varphi|$ returns the smaller angle between $\varphi(t_{n-1})$ and $\varphi(t_n)$, if the jumping point between π and $-\pi$ does not lie within this angle; otherwise $|\Delta\varphi|$ returns the larger of the two angles.

$$
\begin{aligned}
& & &= -(2\pi - (\varphi(t_n) - \varphi(t_{n-1})) & & \\
& & &= \varphi(t_n) - \varphi(t_{n-1}) - 2\pi & & \\
& \underline{\text{CASE:}}\ \Delta\varphi < -\pi & & & & \Delta\varphi \text{ is negative, its sign is: } - \\
& \Delta\varphi_{new} & &= \overbrace{+}^{\textit{opposite sign}} (2\pi - |\Delta\varphi|) & & \Delta\varphi \text{ is negative} \rightarrow |\Delta\varphi| = -\Delta\varphi \\
& & &= 2\pi - (-(\varphi(t_n) - \varphi(t_{n-1})) & & \\
& & &= \varphi(t_n) - \varphi(t_{n-1}) + 2\pi & &
\end{aligned}
$$

If the input is an angle the <u>*angular velocity*</u> is calculated using $\Delta\varphi_{new}$ instead of $\Delta\varphi$:

$$\omega = \frac{\Delta\varphi_{new}}{\Delta t} \tag{4.9}$$

4.4.3 Changes of rolling direction, lowest and highest points

Every time a bar of the wheel reaches a *lowest* or *highest point* (see section 2.1), it crosses the zero axis of the wheel (see Figure 4.3) and the sign of the angle changes. We therefore call lowest and highest points 'changes of sign'. Every time the wheel *changes its rolling direction*, the rotational angle of the wheel reaches either a minimum or a maximum, depending on the original rolling direction.

The detection of changes of sign, minima and maxima of the rotational angle is implemented in one algorithm, which makes a case differentiation over these three cases:

Case 1: change of sign:
For the detection of changes of sign we look at two successive values of the angle. Figure 4.3 shows how the value of the angle passes 0 when the bar reaches a lowest

point and jumps between $-\pi$ and π when it reaches a highest point. In both cases the sign of the previous and the current value differ. Such a change of sign can be found by calculating the sign of the product of the previous and the current value:

IF : $\varphi(t_{n-1}) \cdot \varphi(t_n)$ is negative $\rightarrow$ report change of sign

To determine whether the change of sign indicates a highest point or a lowest point, we check if the absolute value of $\varphi(t_n)$ is close to 0 (lowest point) or close to π (highest point).

Case 2 : minimum:
An extremum, which indicates a change of rolling direction, can be found by comparing 3 successive values $\varphi(t_{n-2})$, $\varphi(t_{n-1})$ and $\varphi(t_n)$. For a minimum the following condition has to be fulfilled:

IF : $\varphi(t_{n-2}) > \varphi(t_{n-1}) < \varphi(t_n) \rightarrow$ report minimum at $\varphi(t_{n-1})$

Case 3: maximum
Accordingly the condition for a maximum is:

IF : $\varphi(t_{n-2}) < \varphi(t_{n-1}) > \varphi(t_n) \rightarrow$ report maximum at $\varphi(t_{n-1})$

Some special cases have to be taken into account due to the potential source of error when the angle jumps from $-\pi$ to π or contrariwise. When this is the case a change of sign is detected. If no change of sign is found normal conditions for extrema apply. To also detect extrema that occur at the same time as a change of sign, the following is done:
Instead of saving the last two values $\varphi(t-2)$ and $\varphi(t-1)$, to compare them for the detection of extrema, the algorithm remembers if the last step was upwards or downwards. (Under normal conditions a step upwards and then a step downwards indicate a maximum for instance.) When a change of sign occurs, the current step has to be into the same direction as the previous one, unless the previous step was a change of sign as well. Therefore unless the previous step was a change of sign as well, the information whether the value is rising or falling is retained from the previous step and the error caused by the 'jumping point' is avoided.
In case of two successive changes of sign, an extremum has to be reported. If the change of sign is a highest point, the 'jumping point' changes the rules for minima and maxima; if the change of sign is a lowest point, normal minimum/ maximum conditions apply. We differentiate between the occurrence of a highest and a lowest point by checking if the absolute value of the angle is closer to π or to 0. This leads to the following pseudo code, which is explained in more detail consecutively:

IF : two successive changes of sign:

CASE A: $\lvert\varphi(t_n)\rvert < \frac{\pi}{2}$	closer to 0 than to π therefore
IF: $\varphi(t_n)$ is positive	lowest point $\rightarrow$ normal min/max conditions
$\rightarrow$ report minimum	
ELSE	
$\rightarrow$ report maximum	
CASE B: $\varphi(t_n) > \frac{\pi}{2}$	closer to π, therefore
$\rightarrow$ report maximum	highest point $\rightarrow$ changed min/max conditions
CASE C: $\varphi(t_n) < -\frac{\pi}{2}$	
$\rightarrow$ report minimum	

In case A $|\varphi(t_n)|$ is closer to 0 than to π, which implies a lowest point. The angle does not jump at a lowest point and a minimum fulfils the same condition as it normally would. Because of the two consecutive changes of sign we already know that a change of rolling direction was found. If the current value of the angle $\varphi(t_n)$ is positive, $\varphi(t_{n-1})$ is negative and $\varphi(t_{n-2})$ is positive and a minimum at $\varphi(t_{n-1})$ has to be reported. Contrariwise the extremum is a maximum if $\varphi(t_n)$ is negative. In case B and C $|\varphi(t_n)|$ is closer to π, which implies a highest point. Due to the jump of the angle, the condition for a minimum and maximum are inverted. In case B $\varphi(t_n)$ is positive leading to a maximum, in case C $\varphi(t_n)$ is negative, leading to a minimum.

When the wheel stands still, small tremors or inaccuracies in the data measurement can cause the detection of many successive changes of rolling direction or changes of sign. Additional to the calculations which were described above, a minimum distance d between two successive extrema or changes of sign can therefore be determined (measured in input data-items). For $d = 3$ a change of sign or extremum would for example only be reported if a minimum of three data-items has arrived since the last change of sign or extremum. Every time a minimum or maximum is found, it is reported together with the current angle $\varphi(t_n)$. This gives access to the information about the *rotational angle of the wheel when it changes rolling direction.*

4.4.4 Average angular velocity and time difference

The *time difference* t_{diff} between two changes of rolling direction is calculated by subtracting the timing of the current extremum from the timing of the previous one. Finally, the *average (angular) velocity* ω_{av} between two changes of rolling directions is computed by adding up all velocities reached between the two extrema and dividing the result by the number of values added.

4.5 Sonification

In the following the concepts behind the four different sonification approaches we chose for our implementation are presented. Subsection 4.5.1 presents the relatively simple Direct-data sonification. Subsection 4.5.2 presents the *Cartoonification* approach, which aims to imitate a real-life rolling sound, subsection 4.5.3 introduces *Vowel synthesis sonification*, which uses synthesised vowel sounds to represent the rotational angle of the wheel and subsection 4.5.4 shows our implementation of an *Event-based sonification*. For all of these sonification techniques, different versions arose during the implementation process. After testing the different versions, we chose one main implementation for each of the four sonification approaches. Unless specified otherwise, the following descriptions refer to these main implementations.

4.5.1 Direct-data sonification

Table 4.1: Mapping of the Direct-data sonification

data feature	sound parameter
data value	rate of a pulsing sound
change of data values	volume

The Direct-data approach produces a sonification directly from the input data (see Table 4.1 for an overview of the mapping). It does not use knowledge about the rotational angle of the wheel or any of the other features derived from it (see subsection 4.2.2). Our main motivation for this approach is to provide a sonification, which can be used for various types of three dimensional data and for versatile applications, including those which are not German wheel-related. As this approach does not use previous knowledge about the wheel and its rolling characteristics, it leaves the interpretation of the resulting feedback up to the performer on the wheel.

The Direct-data sonification generates three sound streams, each of which is controlled by one of the components of the three dimensional input data vectors. Each sound stream is a pulsing sound, whose rate of the pulse is controlled by the values of the incoming data on the represented axis. To distinguish the three streams, each data axis is assigned a fixed pitch for its pulse.
The representation of the values of the incoming data items through the pulse rate was chosen having the functionality of a Geiger-counter in mind because a Geiger-counter is likely to be the most widely known representation of varying data values through sound. To distinguish the three axes acoustically, we chose the pitch as it allows to differentiate the three streams easily even when they are played simultaneously.
The pulsing sound is implemented by using an impulse, the rate of which is controlled by the input data, to stimulate a resonant filter, whose resonant frequency differs for the sound stream of each axis. Intuitively there should be no acoustic

feedback if nothing is happening, thus if no changes occur. (In our application this is the case, for instance, when the wheel stands still.) For this reason the amplitude of each sound stream is set to be dependent on the changes in the respective axis. If incoming data values on the corresponding axis remain the same, the amplitude is set to 0. The larger the changes of the values are, the more the amplitude rises.
The implementation of the Direct-data approach in SuperCollider is discussed in subsection 5.4.1.

4.5.2 Cartoonification

Table 4.2: Mapping of the Cartoonification

data feature	sound parameter
angular velocity	volume and playback velocity of friction sound file
lowest points of all bars	clicking sounds

In the field of sonification, and with auditory icons in particular, the term Cartoonification is used for acoustic feedback, that synthesises, amplifies and exaggerates real-life sounds. This allows the user to use real-life experience for the interpretation of the sound. A cartoonified auditory icon is for example the file deletion sound, which imitates the sound of a sheet of paper being rumpled. Our Cartoonification approach picks up the same idea. We can all associate some kind of sound with a rolling object and, even though it is not very loud, the German wheel *does* produce a sound, which depending on the properties of the floor can sound different and be more or less perceivable.
In spiral moves this natural sound is far more audible then in straight line ones and, depending on its volume, can give feedback about the execution of the move even without artificial amplification. Extending this idea, we come to the conclusion that the Cartoonification of a naturally expected rolling sound can produce a very intuitive auditory feedback for the performer.
With the aim of producing an artificial rolling sound in mind we reconsider the formation of a natural rolling sound: a natural rolling sound is produced by the friction between the floor and the rolling object. Bumps in the floor or on the object can cause additional sound events. To imitate these natural properties we take the following approach (see Table 4.2 for an overview of the mapping of data features to sound parameters). A continuous rolling sound, that mimics the friction between floor and wheel, is produced by playing a recorded friction sound. The volume and the velocity of the playback are controlled by the angular velocity of the rolling wheel, which is computed as described in subsection 4.4.3. This produces a continuous sound that is similar to an amplified natural rolling sound. To cartoonify the rolling sound further and supply more audible information about the wheel's motion, we add the assumption that each of the six bars of the wheel produces a clicking sound when it reaches the floor. The resulting sound events are the synthesised equivalent of the natural sounds caused by bumps on the floor or the wheel. The timing when a bar reaches the

floor is found by continuously computing the current angle of each bar as described in 4.3.1 and detecting all lowest points by applying the algorithm presented in 4.4.3. It is desirable that the friction sound and the clicking sound merge to a consistent rolling sound and that (almost) no sound is produced if the wheel is standing (almost) still. For this reason the volume of the clicking sound is also controlled by the velocity of the wheel.

One of the strengths of human auditory perception is its ability to distinguish and recognise different rhythms and patterns in sound streams. When the above mentioned sound events are generated for each bar that touches the floor, the different fixed distances between the bars (see Figure 2.1), produce such recognisable rhythms, which vary depending on the execution of the respective move. The natural potential for pattern recognition in sound streams allows the performer to use these rhythms for an intuitive comparison and evaluation of performed moves.

During the implementation of the Cartoonification approach we tried a variety of different sounds for the generation of the continuous rolling sound. We used various sound files and also experimented with some fixed frequency resonators, which can be used to simulate the resonant modes of an object. Some of these additional sonifications are still available and can be chosen in the graphical user interface. (More information about calling the sonification from source code and about the graphical user interface is given in sections 5.1 and 5.1.5).

4.5.3 Vowel synthesis sonification

Table 4.3: Mapping of the Vowel synthesis sonification

data feature	sound parameter
rotational angle φ	vowels: 0 to $\pm\pi$ mapped to 'a, e, i, o, u'
angular velocity	volume and pitch of vowels (100-166Hz)
changes of rolling direction	sound events reminiscent to xylophone
φ at change of rolling direction	pitch of the same event
t_{diff} and ω_{av} (see subsection 4.4.4)	volume of the same event
highest points of bar 1 and bar 2	clicking sound

The main idea of the Vowel synthesis sonification approach is to map the rotational angle of the wheel to a corresponding vowel sound or an interpolated sound between two vowels. (Table 4.3 shows an overview of the complete mapping). The idea is motivated by the fact that the listener is already highly adapted to the task of distinguishing between vowels, as they form an essential part of speech. This technique therefore lends itself to quick recognition of changing formants by human users leading to a short feedback loop. The resulting vowel transitions over time can also be imitated with the normal voice for descriptive purposes. Besides others, vowel-based sonifications have been used for the sonification of hyperspectral colon tissue images,

EEG data and hand motion due to these advantages. (Those and further applications of Vowel synthesis for sonification can be found in [CBL+04], [HBSR08], [FCB05], [Coo02] and [KW06].)

Each sound we perceive in every-day life has a characteristic frequency spectrum. For the distinction of different sounds the human auditory system is particularly sensitive to peaks in this frequency spectrum. The most significant peaks, which are usually determined by the acoustic resonance of the sound source, are called formants. Formant filters, which alter a frequency spectrum by amplifying the frequencies within a certain bandwidth around a given formant frequency, can be used in sound synthesis to imitate this effect. Human vowels have different formant frequencies depending on the speaker's type of voice and the vowel type (see [PB52]). Vowel-like sounds can be generated by applying several formant filters according to the formant frequencies of the voice to a complex sound source. The resulting sounds are superimposed to form the vowel (also see: [Kla80] and [Coo02]).

A natural vowel sound consists of a voiced and an unvoiced part. The voiced part is predominant for example when an opera singer is singing and the unvoiced part stands out when someone is whispering. The voiced part can be produced by superimposing the output of several formant filters. The frequency of the sound source that is filtered is the fundamental frequency of the resulting vowel, thus determines the pitch of the vowel. For a more natural result this fundamental frequency can be modulated with another frequency, so that it varies slightly just as a natural voice would when it produces a vowel.
The unvoiced part can be synthesised by superimposing the output of several band-pass filters that filter wide-band noise (which does not have a perceivable pitch) to ensure that the output is unvoiced. The band-pass filters use the same formant frequencies and bandwidths as the formant filters do for the voiced part.
For our Vowel synthesis sonification we use the vowel synthesiser described in [HBSR08], which uses five different formant frequencies for each vowel. Each of these frequencies has a corresponding bandwidth and amplitude level, that specify which part of the spectrum has to be amplified, and by how much. The unvoiced part and the modulation of the fundamental frequency are implemented in our vowel synthesiser to give the option of a more realistic vowel synthesis. After experiments with different ratios between the voiced and the unvoiced part, we found that a more accurate perception of the data is possible when only the voiced part is used and the fundamental frequency is not modulated. Our implementation does still include the option to change the ratio and add a modulation frequency in case a more natural vowel sound is desired for other applications.

We use the vowel synthesiser to map the current angle of the wheel to a vowel sound in the spectrum 'a, e, i, o, u'. At a rotational angle of $\varphi = 0$ a clear 'a' is produced at $\varphi = \pm\frac{\pi}{4}$ an 'e' is audible etc. (see Figure 4.10). A few other mappings which use only a selection of the vowels and/or arrange the vowels differently, were tried. One example includes the mapping of a full rotation instead of half a rotation to the five vowels. This resulted in an asymmetric sound for the very symmetric basic rock move

(see section 2.1). We chose the representation of each half circle by the spectrum 'a, e, i, o, u' because a mapping that preserves the symmetry seemed perceptually more coherent.
For all angles, that lie between the values assigned to two vowels, the frequencies, bandwidths and amplitudes of the formant filters adapt linearly interpolated values. The fundamental frequency, thus the pitch of the voice, is controlled by the angular velocity and ranges from 100 to 166 Hz, which is within the range of spoken language. In some experiments the full range of spoken language (100 to 350) Hz was used, but led to acoustic results which sounded overloaded. In order to avoid the wheel producing sound feedback when it is standing still, the velocity also controls the volume of the vowel sound.

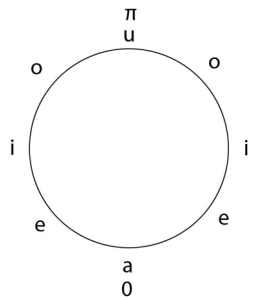


Figure 4.10: Vowel synthesis sonification: the absolute value of the wheel's rotational angle is mapped to vowel sounds in the spectrum 'a, e, i, o, u'.

Another data feature that was chosen for this sonification is the changes of rolling direction of the wheel. They are represented by a sound grain, which is reminiscent of the sound of a rebounding stick when a xylophone or gong is struck. The pitch of this sound is determined by the rotational angle of the wheel when the change of rolling direction occurs. To avoid many irrelevant changes of rolling direction being found due to noise or small oscillations in the input data, the temporal distance (t_{diff}) to the last change of rolling direction and the average angular velocity (ω_{av}) between them dictate which value is assigned to the volume of these sound events. This also causes the changes of rolling direction, which are more significant due to a faster wheel motion, to produce louder sound events than less important changes. Additionally the highest points of bar 1 and bar 2 (see Figure 2.1) are audible as a clicking sound.
The implementation of the Vowel synthesis sonification in SuperCollider is presented in subsection 5.4.3.

4.5.4 Event-based sonification

Table 4.4: Mapping of the Event-based sonification

data feature	sound parameter
rotational angle φ	stream of sound events (one event every time thresholds are trespassed), pitch of those events
angular velocity (indirectly)	density of sound events
lowest points of all bars	same sound events but louder
changes of rolling direction	triangle sound
φ at change of rolling direction	pitch of triangle sound
t_{diff} and ω_{av} (see subsection 4.4.4)	volume of triangle sound
highest points of bar 1 and bar 2	clicking sound

The Event-based sonification is motivated by the suitability of sound to represent time dependent patterns and the ability of human auditory perception to recognise the resulting rhythms. Event-based parameter mapping generates a sound event every time certain conditions are fulfilled (see section 3.1). Strictly speaking we have already come across Event-based sonification, when we discussed the generation of sound events for every change of rolling direction, highest point or lowest point. However, in both Cartoonification and Vowel synthesis sonification, the continuous rolling sound is generated by mapping some of the features, such as the velocity, to acoustic attributes of a continuous sound stream. In this subsection we discuss an approach for which the generation of the 'continuous' rolling sound is implemented Event-based (for an overview of the used mapping see Table 4.4).

The main input feature of the Event-based approach is the rotational angle of the wheel. The values between 0 and $\pm\pi$, which it adapts, are divided into 30 equal steps (steps=30), which are numbered consecutively from 1 to 30 for each half circle. The values between 0 and $\frac{1}{30}\pi$ for instance are assigned to the step $s = 0$, values between $\frac{1}{30}\pi$ and $\frac{2}{30}\pi$ are assigned to $s = 1$ etc. Every time the rotational angle traverses a value, such as $\frac{1}{30}\pi$, which is a threshold between two steps, a sound event is generated. The frequency of the sound events depends on the step that is being traversed and thereby on the rotational angle of the wheel:

$$f = 100 + f_{step} \cdot s \tag{4.10}$$

Here f_{step} is the difference in frequency between the auditory representation of two consecutive steps (see Figure 4.11).

The range of angles that are associated with one step has the size $'stepSize' = \frac{\pi}{steps}$. For each incoming angle φ we find the step $s(\varphi)$, that the current angle is assigned to, by calculating how many steps of the size $stepSize$ fit into it and rounding the value down.

$$s^*(\varphi(t)) = \frac{\varphi(t)}{stepSize} = \frac{\varphi(t)}{\frac{\pi}{steps}} = \frac{\varphi(t) \cdot steps}{\pi}$$

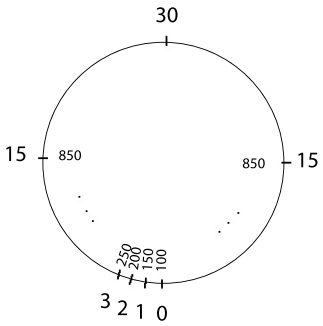


Figure 4.11: Event-based sonification: Every half circle is divided into 30 steps (s, outside number). Every time the reference point passes one of these thresholds, a sound event with the frequency (inside number) $f = 100 + 50 \cdot s$ (in Hz) is generated.

$$s(\varphi(t)) = \lfloor s^*(\varphi(t)) \rfloor \tag{4.11}$$

The resulting value is compared with the step of the previous angle $s(\varphi(t-1))$. If they are not equal ($s(\varphi(t)) \neq s(\varphi(t-1))$) a new step is reached and a sound event is generated.

During the implementation of the Event-based sonification we experimented with different amounts of steps and finally decided to use 30. This is motivated by the fact that for higher amounts of steps, the sound events merge acoustically and are not clearly distinguishable and for lower amounts of steps the resolution of the transmitted information is not as high.
Even though the velocity is not the input feature, its absolute value is clearly audible in form of the density of sound events. At the same time the frequency of the event gives information about the absolute position of the wheel.

As mentioned above the Event-based approach is based on the human ability to recognise and analyse rhythms. When the Cartoonification approach was discussed in subsection 4.5.2 we saw that additional characteristic rhythms can be produced by generating a sound event every time one of the bars reaches the floor. We therefore adopt this approach for the Event-based sonification in a modified version. As in the Cartoonification approach the timing of the events is found by calculating all lowest points of the bars as described in subsection 4.4.3 and a sound event is generated for each of them. In the Event-based approach however, we integrate these sounds into the sequence of events which form the continuous rolling sound. The event that is produced when one of the bars touches the floor is generated in the same way as the 'continuous' rolling sound events. The fixed angles in which the bars touch the floor lie between two steps and likewise the frequency of the effected sounds lie between the two corresponding frequencies. The only difference in the calculation of the frequency is that when the step $s = s_{lowestPoint}$ is calculated and inserted into formula 4.10, the step $s_{lowestPoint}$ is not rounded down and therefore is no integer

but a rational number.

$$s_{lowestPoint} = s^* = \frac{\varphi(t) \cdot steps}{\pi} \tag{4.12}$$

This approach has the advantage that the resulting sound also carry information about the rotational angle of the wheel and thus about which bar is on the floor. Additionally this information is congruent with the information provided by the 'continuous' rolling sound events and marks when some of the most relevant positions of the wheel are passed. We do have to ensure however that the resulting rhythm is clearly audible and does not dissolve into a mere part of the 'continuous' rolling sound. For this reason the amplitude of the sound events, that are generated for each lowest point, are significantly higher, which acoustically lifts the sound into a second sound layer.

A third layer of sound is formed by sound events which are generated for every change in the wheel's rolling direction. These are calculated as described in subsection 4.4.3. For the acoustic representation of these changes of direction we chose the sound of a triangle instrument, the pitch of which adapts to the rotational angle of the wheel, measured at the time of the change of direction. Due to the momentary stillness when the wheel changes its rolling direction, a short silence is audible before and after the stroke of the triangle. The resulting sounds is particularly pleasant if the absence of movement, and therefore the silence, is relatively long. This acoustic representation encourages the user to maintain the stillness for as long as possible. This effect is of special interest for the execution of a range of returning and onward wheel moves, as for most of these moves a longer stillness means more time for the execution of the (usually time-critical) move. For this reason we preferred this acoustic representation over others.
Again as we only want relevant changes of rolling direction to be audible (c.f. Vowel synthesis sonification subsection 4.5.3) the volume of the third sound layer depends on the average angular velocity and the time that has elapsed since the last change of rolling direction.

Beside this main implementation of the Event-based sonification approach we experimented with other amounts of steps and acoustic representations of the different events. Amongst others we implemented a version, that mainly uses piano sounds. The amount of steps for this particular implementation is set to 12, representing the twelve halve steps of an octave.
The implementation of the Event-based sonification in SuperCollider is discussed in subsection 5.4.4.

5 Implementation

In this Chapter the implementation of the concepts described in the previous chapter are presented in detail. In section 5.1 we introduce the programming language SuperCollider and explain how to start the real-time sonification system from a source file or using a graphical user interface. (Appendix D.2 contains information about the installation of SuperCollider and the sonification system.) Section 5.2 covers the implementation of the data acquisition including the hardware, the calibration of the sensor, our approach for live data acquisition and the playback of recorded data. Section 5.3 presents how the feature extraction described in 4.3 and 4.4 is implemented in SuperCollider. Finally section 5.4 covers the implementation of the different sonification approaches discussed in 4.5. Some of the source code examples are simplified. The complete source code of each SuperCollider class mentioned in this chapter can be found on the enclosed CD. An overview of the functionality of all relevant classes can be found in appendix E.1.

5.1 System components and setup

This section gives information concerning hardware, the SuperCollider programming language, and the installation and initialisation of the real-time sonification system. In subsection 5.1.1 and 5.1.2 we present the sensors that were used for our experiments and explain how they need to be set up for use in the sonification system. Subsection 5.1.3 introduces the programming language SuperCollider, which was our main tool for the implementation of the sonification system. Subsection 5.1.4 shows how the sonification system can be started from a SuperCollider source code file (using either recorded data or live data) and subsection 5.1.5 introduces graphical user interfaces, which can be used alternatively.

5.1.1 Magnetometer

In subsection 4.1.2 we stated that our final implementation is based on the data collected with a *MicroMag 3-axis*[1] magnetometer from *Sparkfun Electronics*[2] (see Figure 5.1). The magnetometer (which was assembled by Simon Schulz and Jan Anlauf) is attached to a self-soldered circuit board and accessed by an Atmel At-

[1] data sheet: `http://www.sparkfun.com/datasheets/Sensors/MicroMag3%20Data%20Sheet.pdf`
[2] `www.sparkfun.com`

mega168[3] processor with a clock of 16MHz. Power is supplied by four AA batteries, whose voltage is reduced to 3.3 V by a voltage converter. The data is transmitted wirelessly using a *RN-41 Class 1 Bluetooth Module*[4] as a transmitter and a LevelOne Bluetooth *USB-Adapter MDU-0005USB, Class 1* as a receiver. The USB-Adapter was used because the range of the built in Bluetooth device of the Mac book, which the sonification system was implemented on, was to short (10m compared to 100m with the adapter). Before we upgraded the data transmission hardware to the RN-41

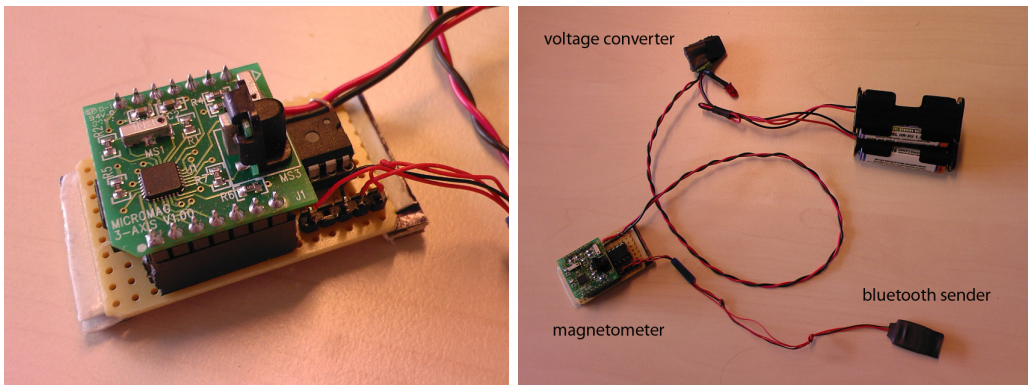


Figure 5.1: The magnetometer

module, we experimented with an *XBee 802.15.4 radio module*[5][6] (see Figure 5.2), which transmits at a frequency of 2.4 GHz. We ultimately opted to use a Bluetooth

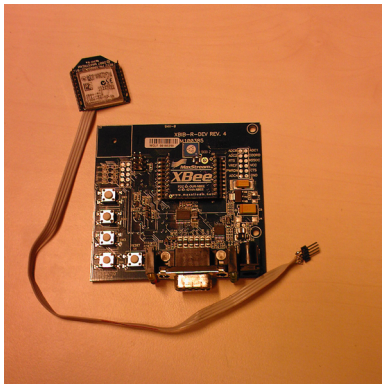


Figure 5.2: The XBee 802.15.4 radio module

transceiver, as the XBee radio module did not have a sufficiently high transmission range, in particular when its line of sight was obscured. In addition to this limitation,

[3]data sheet: `http://atmel.com/dyn/resources/prod_documents/doc2545.pdf`
[4]data sheet at: `www.rovingnetworks.com/documents/RN-41.pdf`
[5] retailer website: `http://www.digi.com/technology/wireless/products.jsp`
[6]data sheet: `http://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf`

the range of the XBee 802.15.4 varied significantly depending on the environment in which we conducted our experiments.

Using the Rn-41 Bluetooth transmitter, we receive the magnetometer data with no visible or audible latency within a range of over 30m, even without direct line of sight between transmitter and receiver.
To use the system in real-time, the Rn-41 Bluetooth transmitter has to be registered with the Bluetooth input device of the used computer. For a successful pairing of the device a passkey in form of a four digit integer is needed (default: 1234). We experienced that sometimes when the magnetometer was used several times, the Mac OS X Bluetooth client failed to recognise the Bluetooth module after the first time. This can be solved by reregistering the Bluetooth transmitter before starting the system.

The MicroMag 3-axis measures magnetic fields and therefore interference can occur if magnetic fields other than the earth's own field are present. Such magnetic fields can for example be caused by electric power lines, because their electrical current flow induces a magnetic field, or by electronic devices that contain magnets or electromagnets, loudspeakers for instance.
For the collection of appropriate data the sensor own power supply and connecting cable should remain in the same spatial relation to the magnetometer over the entire duration of each experiment. As previously mentioned, a calibration has to be undertaken in order to inhibit a negative influence of offsets on the three axes (see subsection 4.1.3). The implementation of this calibration routine is described in subsection 5.2.2.

For the calibration routine each axis of the magnetometer has to be aligned to a 'calibration line' (see section 4.1.3) and for the calculation of the wheel's rotational angle the y-axis has to be aligned to one of the wheel's bars (see subsection 4.3.1). To facilitate these alignments the magnetometer is fixed in a case, the sides and edges of which are each parallel to one of the sensor axes, so that when the edge of the case is aligned to a line, the corresponding sensor axis is as well (see Figure 5.3).

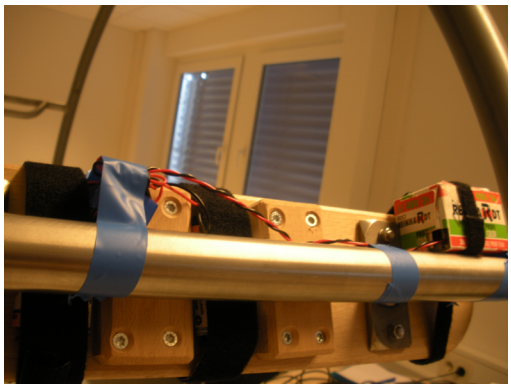


Figure 5.3: The attachment of the sensor to the wheel

5.1.2 Wii Remote

As mentioned in subsection 4.1.1 we used a Wii Remote control for some of our experiments. Even though most of our sonification approaches are based on magnetic field data we include the Wii Remote into our discussion. It can be used as an alternative input device for the Direct-data approach and our experience with this device provides useful knowledge about its characteristics for future research.
The Wii Remote is a controller designed for Nintendo's Wii game console. Besides others, it contains accelerometers in three dimensions, which we used to collect the acceleration data for the calculations described in subsections 4.3.2 - 4.3.4. The programming language SuperCollider, which was used for the implementation of the sonification system (see subsection 5.1.3) contains a class called `WiiMote`, which offers a direct way to access the data transmitted by a Wii Remote. Unfortunately this class is not compatible with the operating system which we implemented our sonification system on (Mac OS X 10.5). For that reason we used the software Darwiin remote and OSCulator. Both programs can act as a gateway between the Wii Remote control and SuperCollider by forwarding the data items received from the Wii Remote as OSC messages. OSCulator can be used concurrently for several Wii Remotes and offers a free Shareware version[7]. Darwiin remote is freely available on the internet[8], but can only process the input data of one Wii Remote. When we worked with Darwiin remote we obviated this drawback by installing several instances of the program.
There are a few properties of the Wii Remote that are important to mention, as ignoring them can lead to incorrect results. The Wii Remote communicates with other devices using the Bluetooth wireless communication. There are certain events that cause the Wii Remote to send a packet of updated state information. Those events include, for example, changes in the data measured by the accelerometer. This means that the Wii Remote does not send a fixed number of data items per time unit. Thus when two or more Wii Remotes are used, it is likely that they send a different amount of data items within the same time frame. Furthermore as discussed for the magnetometer in subsection 4.1.3 the data sent by the Wii Remote can contain an offset and this can necessitate a calibration. The coordinate system of the Wii Remote is shown in Figure 5.4[9]. Depending on the alignment of the sensor on the wheel, this coordinate-system may not always comply with our definition of the sensor-coordinate system in subsection 4.1.2 (see Figure 4.1.2). Nevertheless we always refer to the coordinate system in the way described in subsection 4.1.2 and presume that a transformation into this system has been done internally.

5.1.3 SuperCollider

For the implementation of the real-time sonification system the programming language SuperCollider version 3.2 was used. SuperCollider is a modern real-time lan-

[7] `http://www.osculator.net/wiki/`
[8] `http://sourceforge.net/projects/darwiin-remote/`
[9] source : `http://yg.typepad.com`

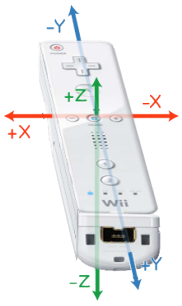


Figure 5.4: The cooordinate system of a Wii Remote

guage for audio programming and sound synthesis, which is available for OSX, Linux and Windows. Our sonification system was implemented on Mac OS X 10.5. We can distinguish two different components of SuperCollider: The audio server 'scsynth' and the programming language 'sclang'. The server calculates the audio signal by using graphs consisting of simple audio elements called 'UGens'. The language 'sclang' is an object oriented programming language whose structure resembles the one of SmallTalk (see [Lou06]) and whose syntax is similar to C (see [KRE88]). The entire communication between 'sclang' and 'scsynth' is realised using the protocol Open Sound Control (OSC)[10]. In our implementation we use OSC for the transmission of data between 'sclang' and 'scsynth' as well as between different parts of our application. SuperCollider is freely available on the internet and was released under the GNU General Public License in 2002. Information about the installation of SuperCollider and the sonification system is given in appendix D.2. Appendix D.1 contains some additional directions for new users. For more information about SuperCollider, its system requirements, its installation on your platform and SuperCollider tutorials please refer to SuperCollider website[11].

5.1.4 Starting the sonification from source code

In this subsection we describe how to start the sonification from source code. This also gives a first overview of some of the most important SuperCollider classes that are part of our implementation. In the course of the following subsections all these classes are presented in more detail. During the discussion of each of these classes it is important to know where the class is placed in the overall context. We therefore advise the reader to come back to this passage regularly to be reminded of the broader context. For guidance to the installation and setup of the real-time sonification system please refer to appendix D.2.

Consecutively an example shows how to start the sonification of live data received from a magnetometer via Bluetooth. Thereafter the recording of data with the sys-

[10] see `http://opensoundcontrol.org`
[11] `http://supercollider.sourceforge.net/`

tem is explained followed by a second example, which starts the sonification of data that was recorded in a file previously. These data files usually consists of a list, the first and second element of which contain an array of calibration data and the initial angle of the wheel (see sections 4.1.3 and 2.2). All further elements contain the magnetic field vectors. To make the source code more comprehensive, we have given all variables descriptive names. If you wish to use this code to run the application, you have to replace all names (e.g. sender, calibrationData...) by one letter variables to make them global. Alternatively you can open the file 'main' on the enclosed CD, which contains the same source code altered in this manner. To start the real-time sonification system, execute the code block-wise:

Live data sonification

```
0    s.boot;                                          // boots the sound server
1   (
2       sender=MagnetometerOscSender.new('/magnetometer/data');
3       sender.start;
4       calibrator=Calibrator.new('/magnetometer/daten');
5   )
6   (
7       calibrationData=calibrator.getCalibrationData;
8       wheelModel=MagnetometerWheelModel.new('/magnetometer/data', '/angle');
9       wheelModel.calibrate(calibrationData);
10      wheelModel.setZeroAngle;
11      wheelModel.start;
12   )
13   (
14      sonifier=SonCartoonification.new('/angle');
15      //sonifier=SonVowel.new('/angle');
16      //sonifier=SonEventBased.new('/angle');
17      //sonifier=SonDirectData.new('/magnetometer/data');
18      sonifier.loadSynthDef(0);
19   )
20   (
21      sonifier.start;
22   )
23   (
24      sonifier.close;
25      wheelModel.close;
26      sender.stop;
27   )
```

Line 2 creates an object of the class `MagnetometerOscSender`, which splits the magnetic field data into 3-D magnetometer data items and sends them in OSC-messages named `/magnetometer/data`. Line 3 starts this sender. Line 4 opens a graphical user interface, which leads the user through a calibration process for the sensor (see subsection 5.2.2). Line 7 retrieves the resulting calibration array. Line 8 creates a `MagnetometerWheelModel`, which uses the magnetic field data to calculate the wheel's rotational angle. Its first argument is the OSC name pattern of the input messages which contain the magnetic field vectors. As the OSC name patterns of the messages are concordant, the object `wheelModel` receives the data sent by the object `sender`. After calculating the angle the `wheelModel` sends it in a new OSC

message called `/angle`. Line 9 passes the calibration data to the `wheelModel`, so that the calibration values can be taken into account in the calculation of the rotational angle. Line 10 sets the initial angle (see section 2.2) to the rotational angle that is currently calculated for the wheel. Therefore the wheel should be brought into the according position with the reference point on the floor before this line is called (see Figure 2.2). Line 11 starts the wheel model and thereby the calculation and visualisation of the rotational angle of the wheel. Line 14 generates the `Sonification` object which uses the angle to calculate all further features and then sonifies them (here the sonification is a Cartoonification as described in subsection 4.5.2). Again, as an argument the `Sonification` object has the OSC name pattern of the input messages, which it receives from the `wheelModel`. Line 15 and 16 are commented out as they are alternatives to line 14. They allow the use of the Vowel synthesis sonification or Event-based sonification instead of the Cartoonification (see subsections 4.5.3 and 4.5.4). Line 18 loads the sound settings onto the SuperCollider sound server. Calling this line with the argument 0 loads the default settings. Line 21 starts the sonification. Line 24, 25 and 26 stop the application.
To summarise the flow of data we can say that a `MagnetometerOscSender` sends the incoming data via OSC messages, a `MagnetometerWheelModel` receives the messages, calculates the rotational angle of the wheel and sends it in a new message, which is received by a `Sonification` object and turned into sound.
As for a sonification directly form the input data none of the features has to calculated, the Direct-data sonification can be started by calling lines 2, 3, 17 and 20.

Recording data

The use of the system for experiments and off-line monitoring requires a tool for the recording of data for later analysis. With the recorded data it has to be possible to reproduce the entire experiment or training session. Besides the data measured with the magnetometer, this also requires knowledge concerning the initial angle of the wheel as explained in section 2.2 and the calibration data that is collected as seen in subsections 4.1.3 and 5.2.2. When the class `MagnetometerOscSender` receives the data from the magnetometer this information can not yet be provided. The recording of data is therefore implemented in the class `MagnetometerWheelModel`, which forms the next implementation layer. This class calculates the rotational angle of the wheel and thus requires the initial angle of the wheel as well as the calibration data (see subsection 4.4.1). A list of magnetometer data, which also contains the calibration data and the initial angle as its first and second element can be saved as follows.
To start recording the incoming data the following two lines have to be called on an object of the class `MagnetometerWeelModel`:

```
wheelModel.start;

wheelModel.record('myAbsolutePathname');
//wheelModel.record('myLocalFileName', true); //alternative for saving a file locally
```

The data is saved automatically, when the calculations done by the `MagnetometerWheelModel` are stopped by calling the method `stop` or `close` on the `wheelModel`. The recording of data with its consequent file opening and writing does not have any noticeable performance implication for the responsiveness of the real-time feedback loop of the system. More information about the class `MagnetometerWheelModel` and its functionality is given in subsection 5.3.1.

Sonification of recorded data

To start the sonification of recorded data the following source code can be executed block-wise.

```
1   s.boot;                                                    //boot the audio server
2
3   (                                                          //load all necessary data
4       sender = DataFromFileOscSender.new('/Users/exampleUser/data1.data',
                '/magnetometer/data');
6       calibrationData=sender.getCalibrationData;
7       zeroAngle=sender.getZeroAngle;
8       wheelModel=MagnetometerWheelModel.new('/magnetometer/data', '/angle');
9       wheelModel.calibrate(calibrationData);
10      wheelModel.setZeroAngleManually(zeroAngle);
11      wheelModel.start;
12      //sonifier=SonVocal.new('/angle');
13      //sonifier=SonCartoonification.new('/angle');
14      sonifier=SonEventBased.new('/angle');
15      //sonifier=SonDirectData.new('/magnetometer/data');
16      sonifier.loadSynthDef(0);
17  )
18
19  (                                                          //start application
20      sender.start;
21      sonifier.start;
22  )
23
24  (                                                          //stop application
25      sonifier.stop;
26      wheelModel.close;
27      sender.stop;
28  )
```

In Line 4 an object of the type `DataFromFileSender` is created. As the name suggests, it reads data from the file '/Users/exampleUser/data1.data', and sends each data item as an OSC message named '/magnetometer/data'. The calibration and detection of the initial angle (see subsections 5.2.2 and 2.2) is performed when the data is recorded. Therefore the calibration data and the initial angle are read from the file in Line 6 and 7 and passed on to the `wheelModel` in line 9 and 10. All consecutive lines are equivalent to the ones described in the example for live data sonification. For a Direct-data sonification lines 4, 15, 19, and 20 have to be called.

5.1.5 Graphical user interface (GUI)

Instead of operating the sonification system from a SuperCollider source code file, it can also be controlled from a graphical user interface (GUI). There are two different graphical user interfaces: one for the playback of live data and one for the playback of recorded data. Besides the more intuitive handling of the application, the GUI has the advantage that, for recorded data, the sonification, the visualisation and a corresponding video file can be played simultaneously.

To start the GUI for **live data sonification** the following lines have to be executed:

```
s.boot;
LiveSonificationPlayer.new;
```

This opens the window shown in Figure 5.5. Beside this control window, two additional windows open. One window visualises the current angle of the wheel (see Figure 5.9), the other window shows a visualisation of the magnetometer data (see Figure 5.6).

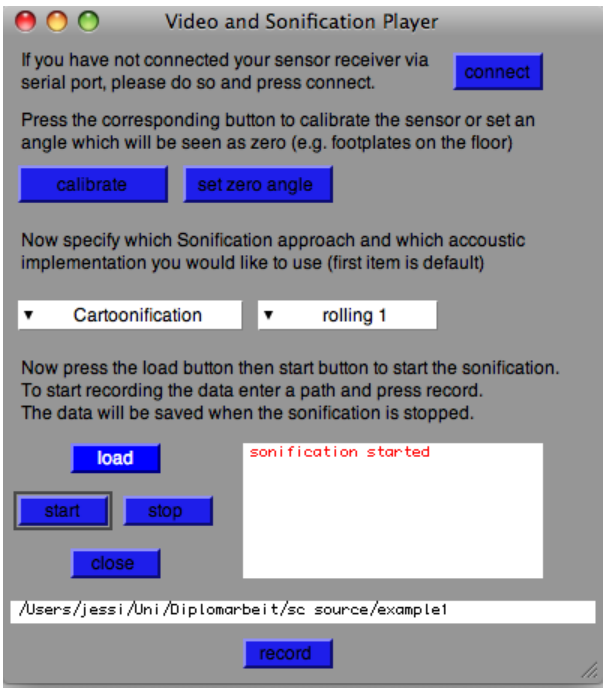


Figure 5.5: The graphical user interface for live data sonification

The GUI allows the user to calibrate the system as seen in subsection 5.2.2, to set the initial angle of the wheel as described in section 2.2 to choose, start and stop one of the sonifications described in section 4.5 and to record the data as presented in 5.1.4. More instructions for the use of the GUI for live data sonification can be found in appendix D.3.1.

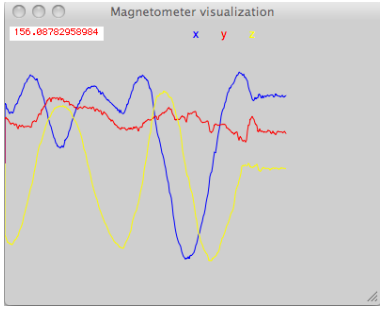


Figure 5.6: The visualisation of input magnetometer data

To start the GUI for **the sonification of recorded data** the following lines have to be executed:

```
s.boot;
VideoAndSonificationPlayer.new('/Users/exampleUser/data1.data',
                               '/Users/exampleUser/data1.m4v');
```

This opens the GUI in Figure 5.7, which provides the same functionality as the above mentioned GUI, but for recorded data. It also allows the user to play back a video simultaneously. Instructions for the use of the GUI for the sonification of recorded data is given in appendix D.3.2.

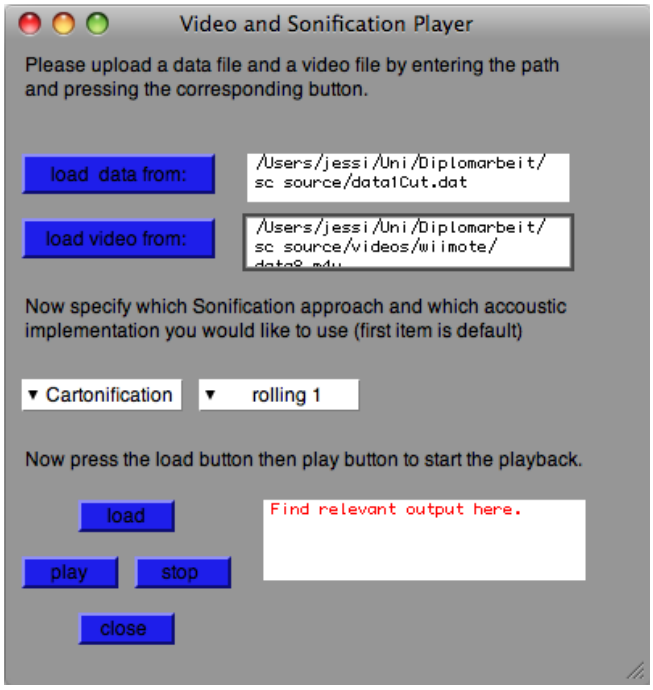


Figure 5.7: The graphical user interface for the sonification of recorded data

Both, the player for live data and the player for recorded data, use OSC messages with identical names. To avoid reciprocal effects only one player should be open at any time.

5.2 Data acquisition

This section introduces the SuperCollider classes, which implement the data acquisition, thus allowing access to the magnetic field data and the calibration data, both of which are either collected live by a magnetometer or read from a recorded file.

5.2.1 Live collection of magnetic field data

As mentioned in subsection 5.1.1, the magnetometer data is received via Bluetooth. The incoming data consists of a continuous stream of ASCII values, which have to be split into data items, each containing one value for the x-, y-, and z-component of the magnetic field vector. The class `MagnetometerOscSender` implements this functionality and sends the resulting data items to the next processing step via OSC messages. An object of the class `MagnetometerOscSender` is created, started and stopped by executing the following lines:

```
sender=MagnetometerOscSender.new('/magnetometer/data');
sender.start;

sender.stop;
```

The string passed to the object is the OSC address pattern of its output message. Additional information about the implementation of the class `MagnetometerOscSender` can be found in appendix E.2.

5.2.2 Magnetometer calibration

The collection of the two calibration values for each axis (see subsection 4.1.3) is implemented in the SuperCollider class `Calibrator`, an object of which can be created by calling:

```
calibrator=Calibrator.new("/magnetometer/data");
```

The passed argument is the OSC address pattern of the input messages that contain the magnetic field data. Creating a `Calibrator` object opens the graphical user interface shown in Figure 5.8. In this interface the user is instructed to consecutively align each axis and its reverse with a fixed 'calibration line' and press a button to collect the corresponding data (see subsection 4.1.3). This data is written into a calibration array of the format $[x_+, x_-, y_+, y_-, z_+, z_-]$. The user has to press the 'done' button to indicate that the calibration is completed. The calibration data is

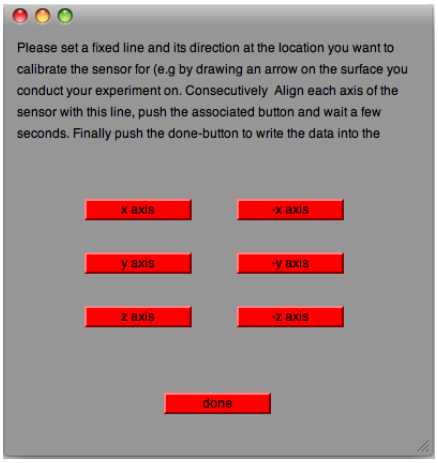


Figure 5.8: The graphical user interface for the calibration

then accessible by calling the method:

```
calibrator.getCallibrationData;
```

To assure that the calculation of the wheel's rotation angle is correct, it is advisable to carry out this calibration before each experiment and to pass the calibration data to the sonification system as described in subsection 5.3.1. However, the offset values appear to be relatively constant for each axis and a default calibration can be undertaken with the `offsetArray= [230, 0, 1000]`. This does not calibrate the y-axis, because the y-values are not used for the calculations of the sonification system.
The calibration can be carried out either when the sensor is attached to the wheel or before attaching it. As the alignment with the calibration line is easier when the magnetometer is not attached, we advice the latter.
An extract from the source code of the class `Calibrator` is provided in appendix E.3. The calculation of the offset values, which the calibration values lead to, is implemented in the class `MagnetometerWheelModel` that is discussed in subsection 5.3.1.

5.2.3 Recorded data

The SuperCollider class `DataFromFileOscSender`, can replace the `MagnetometerOscSender` class as a source of OSC messages that contain the magnetic field data items. Instead of live data arriving via Bluetooth, a `DataFromFileOscSender` object sends data from a file which was recorded previously as described in subsection 5.1.4. A new object of this class can be created and the transmission of data can be started by executing the following lines.

```
sender=dataFromFileOscSender.new('/Users/exampleUser/data1.data');
sender.start;
```

As mentioned in subsection 5.1.4 the first element of the list that is saved in the file contains the calibration data which was collected when the experiment was conducted (see subsection 5.2.2) and the second element contains the initial angle of the wheel (see section 2.2). All following elements of the list consist of magnetic field data, including the x-, y- and z-value and the time when these values were measured. The following methods provide access to the calibration data and the initial angle:

```
calibrationData=sender.getCalibrationData;
initialAngle=sender.getZeroAngle;
```

The time stamp of each data item is used to ensure that the data is sent at the same rate as it was measured originally.

5.3 Feature extraction

This section reviews the calculation of the features discussed in sections 4.3 and 4.4 with a focus on their implementation in SuperCollider. Each SuperCollider class that implements a part of the feature extraction is presented in one of the subsequent subsections.
The calculation of the rotational angle of the wheel is described in subsection 5.3.1. Thereafter the algorithm for the determination of the angular velocity is presented in subsection 5.3.2. The following subsection reviews the computation of all changes of rolling direction, lowest points and highest points. The computation of the average velocity and time difference are not implemented in a separate class, but directly in the sonification classes. Their calculation is discussed in subsection 5.3.4.

5.3.1 Rotational angle $\varphi(t)$

The SuperCollider class `MagnetometerWheelModel` receives an OSC message of the format `[inputOscNamePattern, timeStamp, magnetX, magnetY, magnetZ]` for every incoming magnetic field vector (see subsection 5.2.1). It calculates the current angle of the wheel and sends a new OSC message of the format `[outputOscNamePattern, time, currentAngle]`. Simultaneously it sends a message for the current angle of each of the bars in the same format, but with the output OSC name pattern extended by 'bar1', 'footplate1', 'handle1', 'bar2' etc. Additionally it presents the current position of the wheel in a separate window (see Figure 5.9). To start and stop the calculation and the visualisation, Source code 1 has to be executed.

```
1 wheelModel=MagnetometerWheelModel.new('/magnetometer/data', '/angle');
2 wheelModel.calibrate(calibrationArray);
3
4 wheelModel.setzeroAngle;      //bring wheel into desired initial angle before calling this line
5
6 wheelModel.start;             //start calculation and visualisation
7
8 wheelModel.stop;              //stop (call start to restart)
9 wheelModel.close;             //close
```

Source code 1: Starting a MagnetometerWheelModel.

The first line produces a new `MagnetometerWheelModel` object, which receives the desired OSC name patterns for the input and output OSC message. Two types of calibration data have to be accessible to allow a correct calculation of the wheel's rotational angle: the calibration data of the sensor (see subsection 5.2.2) and the initial angle of the wheel (see section 2.2). The array containing the calibration data of the sensor is passed to the `wheelModel` in line 2, the initial angle of the wheel is set to the currently calculated angle in line 4 and the calculation of the rotational angle is started when line 6 is called.

The array with the calibration data, that is passed to the `wheelModel` in line 2, is used to calculate the x-, y- and z- offset by computing the average of its two calibration values (e.g. x_+ and x_- for the x-axis, see subsection 4.1.3).

```
offsetArray[0]= ((calibrationArray[0]+calibrationArray[1])/2).asInteger;  // x-axis offset
offsetArray[1]= ((calibrationArray[2]+calibrationArray[3])/2).asInteger;  // y-axis offset
offsetArray[2]= ((calibrationArray[4]+calibrationArray[5])/2).asInteger;  // z-axis offset
```

The method calibrate (line 2 in Source code 1) can also be called with an empty array, leading to a default calibration with the `offsetArray= [1000,0,230]` or the offset array can be set manually by calling the calibrate method with an array of only 3 values. For each incoming OSC message of the format `msg= [inputOscNamePattern, timeStamp, magnetX, magnetY, magnetZ]` the `MagnetometerWheelModel` object uses the offsets to assign the following values as effective magnetic field values:

```
magnetx= (msg[2]-offsetArray[0]);  //data= (input - offset)
magnety= (msg[3]-offsetArray[1]);
magnetz= (msg[4]-offsetArray[2]);
```

Using these values, the location of the wheel is calculated with formula 4.2 and the result is brought into the desired range between $-\pi$ and π as described in subsection 4.3.1.

```
angleValue=(atan2(magnetz,magnetx))-zeroAngle;  //calculate value for angle
angle=formatAngle.value(angleValue);            // makes sure angle is between -pi and pi
```

For this calculation the initial angle (`zeroAngle`) of the wheel has to be known. As mentioned, line 4 of Source code 1 sets the desired initial angle to the current angle. Therefore the wheel should be brought into the initial position seen in Figure 2.2, in which we want the rotational angle to be $\varphi = 0$, before calling this line. Alternatively, if the value of the desired initial angle is known, it can be passed to the object

manually (e.g. when the data is read from a recorded file, which also contains the initial angle). In this case the fourth line in the Source code 1 is substituted by:

```
wheelModel.setZeroAngleManually(zeroAngle);
```

The mapping of the angle into the range between $-\pi$ and π is implemented in the method `formatAngle` (see section 4.3.1 for a more detailed explanation of the underlying algorithm):

```
formatAngle={|angle|                                    //input angle might be in incorrect format
        var overshoot,newAngle;                         //overshoot=how much angle exceeds range
        case
                {angle.abs < pi}{newAngle=angle}        //angle already in range
                {angle>pi}{overshoot= angle -pi; newAngle=overshoot-pi}
                {angle<pi.neg}{overshoot=angle+pi; newAngle= pi+overshoot};
        newAngle;                                       //return the formatted angle
};
```

The formatted current angle of the wheel is sent as an OSC message for further processing.

Furthermore the current angle of each bar is found in the same way by adding the fixed angle between the reference point and the bar to the angle before it is formatted (see subsection 4.4.1):

```
angleBar1Value=angleValue+barAngle;     //barAngle=fixed angle betw. bar and reference point
angleBar1=formatAngle.value(angleBar1Value);
```

This information is sent in a separate OSC message for each of the bars. The result of the calculations is visualised in a separate window which can be seen in Figure 5.9. The wheel in its current location is drawn as a circle. A red dot indicates the reference point; blue points indicate the locations of the bars (compare to the photo of a German wheel in Figure 2.2).

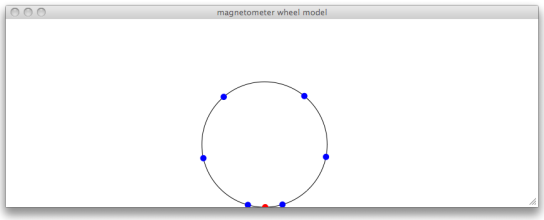


Figure 5.9: The visualisation of the current position/rotational angle of the wheel as generated by a MagnetometerWheelModel

5.3.2 Angular velocity

As its name suggests, the class `Differentiator` numerically calculates the derivation of an incoming data stream. It receives an input OSC message (`msg`) of the format `[inputOscNamePattern, timeStamp, dataValue]`, calculates the derivation as discussed in subsection 4.4.2 and sends a new OSC message of the format `[outputOscNamePattern, time, derivationValue, dataValue]`. The following lines create a new object of the class `Differentiator` and start the computation of the derivation:

```
differentiator=Differentiator.new("/angle",  "/derivation",  [1,3], true);
differentiator.start;
```

The first and second arguments that are passed when the object is created are the OSC name patterns of the input and output messages. The third argument is an array of integers (`kArray`), which specifies which values are used for k when the derivation is computed with formula 4.8. In the example above the derivation is computed for k=1 (two successive values are used to calculate $\Delta\varphi(t)$) and for k=3 ($\varphi(n)$ and its 3rd successor are used). The results are sent in two OSC messages called `/derivation1` and `/derivation3`. As it is necessary to discern angular and linear input, a third argument is passed, which is false for linear input and true for angular input.
The calculation of the derivation is implemented in the method `derive`, which is called for every incoming OSC message. For linear input `derive` implements formula 4.8, which is explained in detail in subsection 4.4.2:

```
1 derive={|t,r,msg|
2       var time=msg[1];                                          //time of measurement
3       phi = msg[2];                                             //current angle phi
4       kArray.do({| item |var k=item;                            //calculate for each k
5               var oldPhi=kAngles[k-1],                          //get phi(n-k)
6               oldTime=kTimes[k-1];                              //get corresponding time
7               deltaPhi=phi-oldPhi
8               derivation=deltaPhi/(time-oldTime);               //derivation
9           net.sendMsg(oscOut ++ k.asString, time, derivation,phi);   //send osc message
10      });
11      kAngles=kAngles.rotate(1);                                //update previous angles
12      kAngles[0]=phi;
13      kTimes=kTimes.rotate(1);                                  //update previous times
14      kTimes[0]=time;
15 };
```

The algorithm has to be expended in case of angular input to anticipate a misinterpretation for jumps between $-\pi$ and π (see subsection 4.4.2). Line 7, in which the difference of angle is calculated, is therefore substituted by the following code, which implements the pseudo code presented in subsection 4.4.2:

```
case
        {(angle-oldAngle) > pi}                         //delta phi > pi
                {deltaAngle=angle-oldAngle-(2*pi);}
        {(oldAngle-angle) > pi}                         //delta phi< -pi
                {deltaAngle=(2*pi) - oldAngle + angle;}
        {true}                                          //-pi<delta phi>pi
                {deltaAngle=angle-oldAngle;};
```

5.3.3 Changes of rolling direction, lowest and highest points

The class `NullMinMaxFinder` receives a stream of messages containing the rotational angle of the wheel or of one of the bars and finds all changes of sign, minima and maxima in its values. As explained in subsection 4.4.3, changes of sign indicate a lowest or highest point and extrema in the angle indicate a change of rolling direction. The data arrives in an OSC message of the format `[inputOscNamePattern, time, angleValue]`. A message of the format `[outputOscNamePattern, time, typeOfPoint, angleValue]` is sent for each detected change of sign (`typeOfPoint=0`), minimum (`typeOfPoint=1`) and maximum (`typeOfPoint=2`). An object of the class `NullMinMaxFinder` is created, started and stopped by the following lines:

```
nullMinMax=NullMinMaxFinder.new('/angle', '/minMax', 3 );
nullMinMax.start;

nullMinMax.stop;
```

As for some of the other classes, which were presented so far, the first two arguments passed to the newly created object are the input and output OSC address patterns. The third Argument determines the minimum distance d between two changes of sign or extrema (see subsection 4.4.3).

The algorithm for the detection of the changes of sign and extrema is explained in detail in subsection 4.4.3. In the following its implementation in SuperCollider is broken down. For a better understanding also refer to the complete algorithm in Appendix E.4.
The basic algorithm differentiates between three different cases: change of sign, minimum and maximum:

```
0 case
1   {(oldValue.sign*value.sign).isPositive.not}                        //change of sign
2             {net.sendMsg(oscOut, msg[1], 0, value);
3              oldValue=value;
4              ...         });}
5
6   {valueIsHigher}                                                    //maximum
7             {if(value<oldValue,{
8                  net.sendMsg(oscOut, msg[1], 2, value);
9                  valueIsHigher=false;
10                  oldValue=value;
11            },{ oldValue=value;})}
12
13  {true}                                                             //minimum
14            {if((value> oldValue),{
15                 net.sendMsg(oscOut, msg[1], 1, value);
16                 valueIsHigher=true;
17                 oldValue=value;
18            },{  oldValue=value; });
19            };
```

Line 1 tests if a change of sign occurred between the last value ($\varphi(t_{n-1})$) and the current value ($\varphi(t_n)$) by checking if the product of their signs is negative. If the product is negative, this implies a change of sign and an OSC message with `typeOfPoint=0` is sent. The variable `valueIsHigher` is a boolean variable which indicates whether

$\varphi(t_{n-2})$ is smaller then $\varphi(t_{n-1})$. If this is the case and line 7 finds that additionally $\varphi(t_{n-1})$ is larger than $\varphi(t_n)$, the condition for a maximum is fulfilled and the maximum is reported by sending an OSC message with `typeOfPoint=2`. In the third case we already know that `valueIsHigher` is `false` and thus the data values were declining in the previous step. In line 14 we test if the values are now rising. If so a minimum is reported through an OSC message with `typeOfPoint=1`. Additionally the variables `oldValue` and `valueIsHigher` are updated after each step for their usage in the next step (line 4, 9, 10, 11...).

This implementation presumes that no minimum or maximum can be found in case of a change of sign. In this way the problems caused by the 'jumping point' between $-\pi$ and π are circumvented. To also detect minima and maxima which occur at the same time as a change of sign, the special cases listed in subsection 4.4.3 have to be considered. This is done by inserting the following lines, which are only executed in case of a change of sign, at the placeholder '...' above:

```
1 if(kNull==0,{             //test if two successive changes of sign
2
3   case
4       { value.abs <piHalf } {                        //value closer to 0 than pi->
5           if(value.isPositive,{                      //lowest point ->
6               net.sendMsg(oscOut, msg[1], 1, value);  //normal min/max conditions
7               valueIsHigher=true;
8               oldValue=value;
9           },{
10          net.sendMsg(oscOut, msg[1], 2, value);
11          valueIsHigher=false;
12          oldValue=value; }); }
13
14      { value>piHalf  }  {                           //value positive close to pi ->
15          net.sendMsg(oscOut, msg[1], 2, value);     //highest point->
16          valueIsHigher=false;                       //reversed min/max conditions
17          oldValue=value; }
18
19      {value<(-piHalf) } {                           //value negative close to -pi->
20          net.sendMsg(oscOut, msg[1], 1, value);     //highest point->
21          valueIsHigher=true;                        // reversed min/max conditions
22          oldValue=value; };
24 });
```

The variable `kNull` counts how many data items have arrived since the last change of sign. Thus the first line checks if the last step was a change of sign as well and therefore if two changes of sign occurred in direct succession. If so, this indicates the occurrence of an extremum.
To find out whether the extremum is a minimum or a maximum, the case differentiation described in subsection 4.4.3 is made. The comparisons in line 4, 14 and 19 aim to determine if the value of the current angle $\varphi(t_n)$ is closer to 0 or to π. Line 4 treats case A in subsection 4.4.3: The value is closer to 0 than π, thus the detected change of sign is a lowest point and normal minimum/ maximum conditions apply because there is no 'jumping point' between the previous and the current angle. Hence if $\varphi(t_n)$ is positive (line 5), $\varphi(t_{n-1})$ is negative and $\varphi(t_{n-2})$ is positive and a minimum is reported for $\varphi(t_{n-1})$ (Line 6). Likewise if $\varphi(t_n)$ is negative, a maximum is reported

(line 10). Case B in subsection 4.4.3 is treated in line 14 ff. and case C in line 19 ff. Here the current angle is closer to $\pi/-\pi$ and the conditions for minima and maxima are inverted due to the influence of the 'jumping point' between $-\pi$ and π.

What has not bee taken into account so far is the minimum distance we require between two changes of sign or extrema. To ensure that this constraint is fulfilled three variables are introduced: `kBigger` counts how many steps the value of the angle has been increasing for, `kSmaller` counts how many steps it has been falling for and `kNull`, which was already used above, counts the steps since the last change of sign. The relevant variables are updated for each step. Before an OSC message is sent to report a change of sign or extremum, it is checked if the distance to the last change of sign or extremum is high enough. For this purpose line 20 to 22 are replaced by the following lines for case C for instance:

```
if(kSmaller>minDistance,{net.sendMsg(oscOut, msg[1], 1, value);});
valueIsHigher=true;
oldValue=value;
kSmaller=0;
kBigger=0;
```

Likewise the variables are updated for case A and B. The full resulting algorithm is presented in Appendix E.4.

5.3.4 Average angular velocity and time difference

The time difference and the average angular velocity between the last and the current change of rolling direction are not implemented in a separate class. They are computed when the calculation of the other features is done in the course of the sonification. There are three different variables which save information that is important for the calculation of the time difference and the average angular velocity: `vAccum`, `vSteps` and `tLastMinMax`. For every incoming data item the current angular velocity is computed and its value is added to `vAccum`, at the same time `vSteps` saves how many of these velocity values have been added up. The variable `tLastMinMax` saves the time of the last change of rolling direction. When a change of rolling direction is found the time difference and average angular velocity are computed and the variables are reset:

```
tDiff= tCurrenMinMax-tLastMinMax;
vAverage=vAccum/vSteps;
vAccum=0;
vSteps=0;
tLastMinMax=tCurrentMinMax;
```

5.4 Sonification

In the following sections we specify how the different sonification approaches, that were presented in section 4.5, are implemented in SuperCollider. Each of those approaches is implemented in a separate class. All these classes in turn are sub-classes of the class `Sonification`, a framework which contains the structure that each sonification class should have for the sake of consistency. The class `Sonification` does for example include the following methods that are inherited by all sub-classes:

`*new` creates a new object.

`initAll` implements initial steps to be performed for *all* sonification objects.

`init` performs initial steps for objects of the specific sub-class.

`loadSynthDef` uploads the sound settings to the SuperCollider sound server. This method takes an integer (`soundNumber`) as its argument. Passing different values for `soundNumber` allows to select from a variety of sound settings. All explanations in the following sections refer to `loadSynthDef` being called with the argument `soundNumber=0`. This uploads the default settings for the respective sonification approach. You can learn more about the alternative sound options, which arose as a by-product of our main implementation, by looking at the source file on the enclosed CD.

`start` starts the sonification.

`record` is called with a file name. It starts recording the magnetic field data as described in subsection 5.1.4. Besides it also writes the velocity values and all changes of rolling direction, minima and maxima into lists as this information is used for later evaluation.

`save` writes the data lists, that are created when `record` is called, into files.

`stop` stops the sonification (can be started again) and calls save in case record has been called.

`close` calls `stop` and cleans up.

Accordingly a new object of any of the sonification classes can be created, started and stopped by the following lines with the corresponding class name:

```
sonifier=Sonification.new('/angle');
sonifier.start;

sonifier.stop;
```

The argument passed to the `Sonification` object in its creation is the OSC name pattern of the input message. This message has to contain the angle of the wheel (format: `[inputOscNamePattern, timeStamp, angle]`) for all `Sonification` classes except the Direct-data sonification. The Direct-data sonification requires the input messages to contain the magnetic field vectors (message format: `[inputOscNamePattern, timeStamp, magX, magY, maxZ]`).

When the method `initAll` is called some variables are initialised and all objects that are necessary for the feature extraction are created (see section 5.3).

```
// computes angular velocity of the wheel
differentiator=Differentiator.new(oscIn, '/derivation', [1],true);
//computes the changes of rolling direction
nullMinMax=NullMinMaxFinder.new(oscIn, '/minMax/null', 3 );

// compute lowest and highest points for the different bars
bar1MinMax=NullMinMaxFinder.new(oscIn++'/bar1', '/minMax/bar1', 3 );
bar2MinMax=NullMinMaxFinder.new(oscIn++'/bar2', '/minMax/bar2', 3 );

footplate1MinMax=NullMinMaxFinder.new(oscIn++'/footplate1', '/minMax/footplate1', 3 );
footplate2MinMax=NullMinMaxFinder.new(oscIn++'/footplate2', '/minMax/footplate2', 3 );

handle1MinMax=NullMinMaxFinder.new(oscIn++'/handle1', '/minMax/handle1', 3 );
handle2MinMax=NullMinMaxFinder.new(oscIn++'/handle2', '/minMax/handle2', 3 );
```

Thus all sonification classes have access to the features listed in subsection 4.2.2 and the calculation of those features that are necessary for the specific approach can be started by starting the corresponding objects.

The NullMinMaxFinder objects report changes of sign (indicating a lowest or highest point) as well as minima and maxima (indicating a change of rolling direction). Thus, when sound events are generated to mark one of these events, a case differentiation over the type of event that is reported by the message has to be done. For the sake of clarity these differentiations are left out in the following sections.

5.4.1 Direct-data sonification

The Direct-data sonification approach, which produces one pulsing sound stream for each data component, is implemented in the class `SonDirectData`. When a `SonDirectData` object is created and the default sound settings are uploaded by calling the `loadSynthDef` method with the argument 0 (see section 5.4), the following definition of the sound is uploaded to the server:

```
1 SynthDef("raw1", { |rateX=5, ampX=0, rateY=5, ampY=0, rateZ=5, ampZ=0,
                      freqX=400, freqY=800, freqZ=1600|
2 var sum;
3
4    sum = (ampX*Ringz.ar(LPF.ar(Impulse.ar(rateX, 0, 0.3), 2000),  freqX, 0.5))
5         + (ampY*Ringz.ar(LPF.ar(Impulse.ar(rateY, 0, 0.3), 2000),  freqY, 0.5))
6         + (ampZ*Ringz.ar(LPF.ar(Impulse.ar(rateZ, 0, 0.3), 2000),  freqZ, 0.5));
7    Out.ar([0,1], sum);
8 }).load(s)
```

Each of the summands of `sum` corresponds to the sound stream of one dimension. To get the pulsing sound the sound generation is triggered with an `Impulse`, whose rate is controlled by the value of the corresponding data component (`rateX`, `rateY` and `rateZ`). After sending the Impulse through a low pass filter to soften it, it is used to trigger the resonant filter `Ringz`. Each resonant filter has a different resonance

frequency (`freqX`, `freqY` and `freqZ`) so that the pitches of the three different pulsing sounds differ. The different frequencies and the range of amplitudes were determined by testing different options in the context of use.

An instance of this `SynthDef` is created when the `start` method of the `SonDirectData` object is called. The method `respData` is then executed for each incoming data item.

```
1    respData={|t,r,msg|
2         var time,x ,rateX, ampX=0, changeOfValueX;
3         time=msg[1];                                        //time when data item was sent
4         x=msg[2]*scaleFactor;                               //x-data value
5         changeOfValueX=(x-oldX).abs/(time-oldTime);         //how much values changed
6         ampX=changeOfValueX.linlin(0,2000, 0,1);            //this change controls amplitude
7         rateX=x.abs.linlin(0,2000,0,15);                    //rate controlled by data component
8
9         threeKlick.set([\rateX, rateX, \ampX, ampX]);
10
11        oldTime=time;                                       //update for next step
12        oldX=x;
13   };
14   dataResponder=OSCresponderNode(nil,oscIn,respData);
```

For the sake of clarity only the x-axis is shown in this example and a few other details are left out. These can be seen in the source file on the enclosed CD.
First the method reads the transmission-time and the x-value from the arriving OSC message in line 3 and 4. It uses them in line 5 to test how much the data in the x component changed since the last step. This calculation is basically the absolute value of a simple numerical differentiation as described in appendix A.2. The amplitude is determined by this value in line 6, which causes the sound stream to be louder, the larger the change on the x-axis is. The rate of the pulse sound is controlled by the value of the input data x-component. We ascertained by experiment, that the data components usually fall between 0 and 2000 and therefore this range is mapped to values between 0 and 15 for the rate. Finally line 9 updates the sound-producing `Synth` by passing the new amplitude and rate to it. Line 11 and 12 do necessary updates for the processing of the next incoming data item.
Due to line 7, x-values above 2000 do not cause changes in the rate. We made this choice because when the earth's magnetic field is the only field present, the values don't usually exceed this threshold. It can therefore be an indication for the presence of an additional magnetic field source, if the rate of the impulse remains at a consistently high level, even when the measured values should be varying.
One last detail that requires explanation before moving on to the next sonification approach is the `scaleFactor` in line 4. This scale factor makes the approach usable for a variety of different input data types. By setting this factor, the data can be brought into the right range for the sonification. The class therefore implements a method which allows the user to set this scaling factor at runtime:

```
sonifier.setScaleFactor(scaleFactor);
```

5.4.2 Cartoonification

In section 4.5.2 we introduced the concept behind the Cartoonification approach that aims to copy a natural rolling sound by generating a continuous rolling sound and additional clicking sounds every time one of the bars touches the floor. This sonification is implemented in the class `SonCartoonification`.
The features that are necessary for this sonification are the angular velocity of the wheel and the lowest points of each of the bars. They are computed by the objects of the classes `Differentiator` and `NullMinMaxfinder` (see subsections 5.3.2 and 5.3.3), which are created in the initiation of each `Sonification` object (see section 5.4). The method `respVelocity` is called for each arriving angular velocity value. This method allows the angular velocity to control the continuous rolling sound by adapting the amplitude and the playback velocity of the played friction sound file according to the values of the angular velocity:

```
1  respVelocity = {|t,r,msg|                    //msg[2] contains the velocity values
2               amplitude=msg[2].abs.linlin(0.007,3,0,0.5);
3               playbackVelocity= msg[2].abs.linlin(0,1.5,minVelocityScale,maxVelocityScale) ;
4               synthRolling.set(\amp, amplitude, \factor, playbackVelocity);
5  };
```

Line 2 and 3 adapt the amplitude and playback velocity according to the angular velocity of the wheel. Line 2 maps velocity values between 0.007 and 3 to amplitudes between 0 and 0.5. Ignoring values below 0.007 avoids the production of sound, due to noise in the input data, when the wheel is standing still. In a similar way the velocity values are mapped to playback velocities between the values of `minVelocityScale` and `maxVelocityScale`. Line 4 passes these new sound settings to the `Synth`, which produces the continuous rolling sound.

The generation of a clicking sound is triggered every time one of the bars touches the floor. This is implemented in the method `respBarMinMax`, which is called every time one of the `NullMinMaxFinder`s reports a lowest point:

```
respBarMinMax = {|t,r,msg|
     Synth.new(\BarOnFloorGrain,[\amp, amplitude.linlin(0,0.5,0.5,10)]) ; });
};
```

The continuous rolling sound is defined in the `SynthDef` '`rollingSound`'.

```
1  SynthDef("rollingSound", { arg i_out = 0, sound, amp=0,srate=44100, start=0.5, end=1, rate=1;
2
3               var duration,source, env;
4
5               minVelocityScale=5;
6               maxVelocityScale=5.2;
7               buffer = Buffer.read(s,"exampleSoundFile.wav", 0, 4*srate)
8               duration = abs(end - start);
9               source=BufRd.ar(1,buffer,LinLin.ar(LFSaw.ar(rate/duration, 1),
10                              -1, 1, start, end)*srate );
11              env = EnvGen.kr(Env.linen(0.01, 0.98, 0.01,rollingVolume), timeScale:
12                              duration, gate: Impulse.kr(1/duration,doneAction: 2));
13              sound=(source*env);
14
15              Out.ar(i_out,Pan2.ar( sound, 0, amp*5) );
```

```
16 }).send(s);
```

As mentioned previously, the variables `MinVelocityScale` and `MaxVelocityScale` determine the minimum and the maximum of the playback velocity. In Line 7 we define which sound file should be used (e.g. a recorded friction sound[12]). A part of the file `exampleSoundFile.wav` is loaded from the current folder onto the sound server `s`. The uploaded sound starts at frame 0 of the original file and ends at frame `4*srate`. The parameter `srate` is the sample rate and defines how many frames are played per second. Hence we read the first 4 seconds of the file into the buffer. Not the whole four seconds are used however. By setting the arguments `start` and `end` (line 1), the part that is to be used is defined. This also determines the duration of the used sound segment (line 8). For the continuous rolling sound we want the friction sound to be played continuously and the sound file therefore has to be looped. Line 9 and 10 implement the loop. The first two arguments of `BufRd` (line 9) define the source sound as a one channel sound which is read from the buffer whose content is uploaded in line 7. The third argument is the phase of the reading, which determines what part of the sound file is currently being played. Its input is slightly more complex, as it essentially implements the loop: `LfSaw` is a sawtooth wave, which adapts values between -1 and 1 before jumping back to -1. `LinLin` maps these values between -1 and 1 to values between `start` and `end` (see Figure 5.10). As the sawtooth wave controls the phase of `BufRd`, it determines which part of the sound segment between `start` and `end` is currently played. For a continuous sawtooth wave as input, the segment is playing repeatedly. As the current phase of the buffer reading is measured in samples

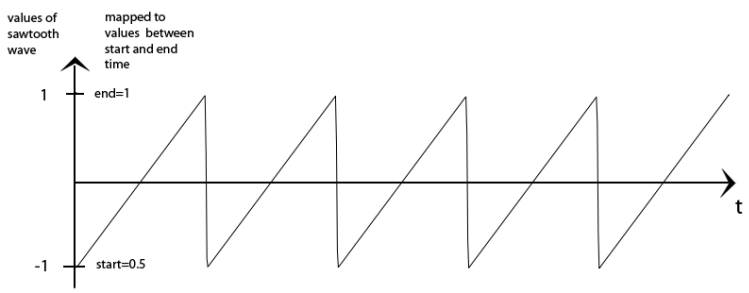


Figure 5.10: The sawtooth wave: values between -1 and 1 are mapped to the times that indicate the segment of the buffer that should be played.

rather then seconds, the output value of `LinLin` is multiplied by the sample rate. The playback velocity can be controlled by setting the frequency of the sawtooth wave, which is the first argument of `LFSaw` in line 9. The file segment is played at a normal speed if one sawtooth takes the time of the original length of the sound segment ($\texttt{frequency} = \frac{1}{\texttt{duration}}$). The variable `rate` is therefore the factor which determines the playback velocity ($\texttt{rate} = 2 \rightarrow$ double speed, $\texttt{rate} = \frac{1}{2} \rightarrow$ half speed etc.). The second argument of `LFSaw` is its phase and ensures that the wave starts

[12]The friction sound that we used was recorded by Tim Kahn.

at its lowest value.
When a part of a sound file is looped there is a potential for clicks caused by sharp transitions between the end and the beginning of the sound segment. To avoid this drawback lines 11 and 12 introduce an envelope, which has the length of the loop, a fast attack and decay, and a gate equal to the frequency of the loop. Multiplying the source sound with the envelope (line 13) before sending it as a 2 channel output (line 15) slightly fades the segment in and out, which avoids that the transitions are audible. Even though there are simpler ways to implement a loop, we chose this implementation for its flexibility. It allows the user to set the speed of the playback as well as the exact sound segment that is to be played at any time, which is a clear advantage over other implementations considered.

The second sound used for the Cartoonification approach is the clicking sound, which is produced every time one of the bars touches the floor:

```
1 SynthDef("BarOnFloorGrain", { arg i_out = 0, pan, freq = 200, dur = 0.01,amp=1;
2               var env;
3
4               env = EnvGen.kr(Env.perc(0.003, dur, 1),doneAction:2);
5               Out.ar(i_out, Pan2.ar(FSinOsc.ar(freq), pan, amp*0.5) * env)   }
6   }).send(s);
```

Line 3 implements the percussive envelope, which determines the length (`dur`) of the sound in seconds. Line 4 generates a fast sine oscillator and writes it to the output bus after multiplying it with the envelope. The acoustic result is a slightly wooden sounding click.

5.4.3 Vowel synthesis sonification

In the Vowel synthesis sonification, which is implemented in the class `SonVowel`, the rotational angle of the wheel is mapped to a vowel sound in the spectrum 'a, e, i, o, u' (see subsection 4.5.3). The pitch and the amplitude of the synthesised voice is controlled by the angular velocity of the wheel. Additionally, a sound event is generated for each change of rolling direction and a different sound event is produced for all highest points of bar 1 and bar 2 (see Figure 2.1).
The input OSC messages contain the rotational angle of the wheel. All objects that are necessary for the computation of the angular velocity, the changes of rolling direction and the highest points of the bars are inherited from the super-class `Sonification` (see section 5.4).

In the following the definition of the continuous rolling sound is introduced and the method `respVelocity`, which adapts the vowel sound based on the incoming data, is presented. Thereafter the definitions and the generation of the additional sound events is discussed.

Each synthesised vowel sound is defined by a set of five formant frequencies, their amplitudes and bandwidths. A vowel sound is produced using the following `SynthDef`, which defines a vowel synthesiser as presented by Hermann and Baier in [HBSR08]:

```
1 SynthDef("vocalSound", { | out=0, f0=135, level=0, vn=1,   // voiced-noise ratio
2               vfreq=0, vmod=0.2, pan=0, amp=0,
3               f1=600,  bw1=80,  g1=(0),
4               f2=1040, bw2=70,  g2=(-7),
5               f3=2250, bw3=110, g3=(-9),
6               f4=2450, bw4=120, g4=(-9),
7               f5=2750, bw5=130, g5=(-20),
8               filtcf=1000, lg=0.05 |
9
10              var ffreq, sum, av0, an0, ain, env;
11
12              env=EnvGen.kr(Env.cutoff(1, 1, 4),doneAction:2) ;
13
14              ffreq = SinOsc.ar(vfreq, 0, mul: vmod, add: f0.lag(5*lg));
15              an0 = LPF.ar( WhiteNoise.ar, 18000);
16
17              an0 =  BPF.ar( an0, f1.lag(lg), (bw1/f1+0.1).lag(lg), g1.lag(lg).dbamp)
18                       + BPF.ar( an0, f2.lag(lg), (bw2/f2+0.1).lag(lg), g2.lag(lg).dbamp)
19                      + BPF.ar( an0, f3.lag(lg), (bw3/f3+0.1).lag(lg), g3.lag(lg).dbamp)
20                      + BPF.ar( an0, f4.lag(lg), (bw4/f4+0.1).lag(lg), g4.lag(lg).dbamp)
21                      + BPF.ar( an0, f5.lag(lg), (bw5/f5+0.1).lag(lg), g5.lag(lg).dbamp);
22              an0 = an0*4;
23              av0 = Formant.ar( ffreq, ffreq.lag(lg), 100, 0.5)
24                      + Formant.ar( ffreq, f1.lag(lg), bw1.lag(lg), g1.lag(lg).dbamp)
25                      + Formant.ar( ffreq, f2.lag(lg), bw2.lag(lg), g2.lag(lg).dbamp)
26                      + Formant.ar( ffreq, f3.lag(lg), bw3.lag(lg), g3.lag(lg).dbamp)
27                      + Formant.ar( ffreq, f4.lag(lg), bw4.lag(lg), g4.lag(lg).dbamp)
28                      + Formant.ar( ffreq, f5.lag(lg), bw5.lag(lg), g5.lag(lg).dbamp);
29
30              ain = (vn.lag(lg)*av0)+((1-vn.lag(lg))*an0);
31              sum = LPF.ar(ain, filtcf);
32              Out.ar(out, Pan2.ar(sum, pan, amp.lag(lg*5),\mul, 0)*env)
33      }).load(s);
```

The fundamental frequency, which defines the pitch of the vowel sound is `f0` (line 1). Line 3 to 7 set the formant frequencies and their bandwidths and amplitudes for the vowel (here a bass a). This information is used to specify the formant filters for the voiced part of the vowel in line 23 to 28 and the band pass filters for the unvoiced part in line 17 to 21.

In subsection 4.5.3 we pointed out that a frequency modulation of the fundamental frequency `f0` can lead to a more naturalistic vowel sound. Line 14 applies this additional vibrato to the fundamental frequency, before its result (`ffreq`) is used as an input for the formant filters. The unvoiced part, which does not have a pitch, uses white noise as an input instead (line 15 and 17 ff.). The variable `av0` comprises the voiced part of the vowel and `an0` contains the unvoiced part. The ratio of these two components is set by a linear interpolation in line 30. As noted in subsection 4.5.3, the use of the unvoiced part and the frequency modulation are not appropriate for our purposes. By setting the voice-noise ratio (`vn`) to 1 and the modulation frequency (`vfreq`) to zero, we turn these additional features off (line 30 and 14).

When the continuous rolling sound is started, a `Synth` is created as an instance of this `SynthDef`. The different vowel sounds can be generated by setting the formant frequencies, bandwidths and amplitudes of this `Synth` to the values of the desired vowel. However, for a continuous mapping of the angle to the spectrum 'a, e, i, o, u' interpolation values are also required. To interpolate between the different vowels,

we define the following envelopes:

```
//formant frequencies
envf1 = Env(#[600,   400 , 250, 400, 350], #[0.25, 0.25, 0.25, 0.25]);
envf2 = Env(#[1040, 1620, 1750, 750, 600], #[0.25, 0.25, 0.25, 0.25]);
envf3 = Env(#[2250, 2400, 2600, 2400, 2400], #[0.25, 0.25, 0.25, 0.25]);
envf4 = Env(#[2450, 2800, 3050, 2600, 2675], #[0.25, 0.25, 0.25, 0.25]);
envf5 = Env(#[2750, 3100, 3340, 2900, 2950], #[0.25, 0.25, 0.25, 0.25]);

//amplitudes
ampEnvf1 = Env(#[0,    0 ,   0,    0,  0], #[0.25, 0.25, 0.25, 0.25]);
ampEnvf2 = Env(#[-7,  -12, -30, -11, -20], #[0.25, 0.25, 0.25, 0.25]);
ampEnvf3 = Env(#[-9,  -9 , -16, -21, -32], #[0.25, 0.25, 0.25, 0.25]);
ampEnvf4 = Env(#[-9,  -12, -22, -20, -28], #[0.25, 0.25, 0.25, 0.25]);
ampEnvf5 = Env(#[-20, -18, -28, -40, -36], #[0.25, 0.25, 0.25, 0.25]);

//bandwidths
bwEnvf1= Env(#[60 ,  40,  60,  40,  40],#[0.25, 0.25, 0.25, 0.25]);
bwEnvf2= Env(#[70 ,  80,  90,  80,  80],#[0.25, 0.25, 0.25, 0.25]);
bwEnvf3= Env(#[110, 100, 100, 100, 100, ],#[0.25, 0.25, 0.25, 0.25]);
bwEnvf4= Env(#[120, 120, 120, 120, 120],#[0.25, 0.25, 0.25, 0.25]);
bwEnvf5= Env(#[130, 120, 120, 120, 120],#[0.25, 0.25, 0.25, 0.25]);
```

The first set of envelopes contains the formant frequencies, the second the amplitudes and the third the bandwidths[13]. All values for one vowel can be found in one column from the first of the two arrays that are passed to each envelope. The vowel 'a' for example has the formant frequencies 600, 1040, 2250, 2450 and 2750, which are amplified by their amplitudes 0, -7, -9, -9 and -20 within the bandwidths 60, 70, 110, 120 and 130. The second column contains the values for 'e' the third column for 'i', etc. As mentioned, when we map the angle to these vowel sounds, we need values for the formant frequencies, amplitudes and bandwidths that produce an interpolated sound which lies between two vowels. The above envelopes adapt all these interpolation values within a duration of one second. As specified in the second arrays it takes the envelopes 0.25 seconds to change from one value to another. The envelope for the interpolated values of the first formant frequency for example (see Figure 5.11) starts with a value of 600, the first formant frequency of an a, measured in pulses per second. Within 0.25 seconds it linearly decreases to 400, the value for an e, after another 0.25 seconds it reaches 250, the value for an i and so on. All interpolation values for the first formant frequency can be accessed by calling the envelope at the corresponding time between 0 and 1.

The method `respVelocity`, which is called for each incoming velocity value, uses this functionality to set the vowel of the sound according to the current angle while at the same time adapting the amplitude and fundamental frequency of the vowel sound according to the current velocity:

```
0 respVelocity = {|t,r,msg|
1               var amp, angularVel, angle;
2               angularVel=msg[2]; angle=msg[3]
3               //      set fundamental frequency according to velocity
4               freq=angularVel.abs.linlin(0, 5, 100, 166);
5               angle=msg[3];
```

[13] the source of this information is `http://ecmc.rochester.edu/onlinedocs/Csound/Appendices/table3.html`.

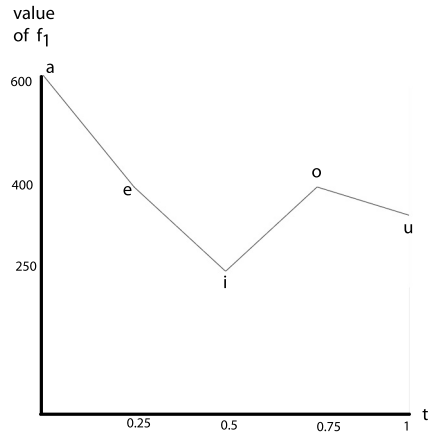


Figure 5.11: The envelope for the interpolation of values for the first formant frequency

```
6               //      set amplitude according to velocity
7               amp= angularVel.abs.linlin(0.007, 3, 0, 0.8);
8               synth.set(\f0, freq, \amp, amp,
9                                           //access interpolated values of formant frequencies
10                              \f1, envf1.at(angle.abs/pi),
11                              \f2, envf2.at(angle.abs/pi),
12                              \f3 ,envf3.at(angle.abs/pi),
13                              \f4, envf4.at(angle.abs/pi),
14                              \f5, envf5.at(angle.abs/pi),
15                                          //access interpolated values of amplitudes
16                              \g1, ampEnvf1.at(angle.abs/pi),
17                              \g2, ampEnvf2.at(angle.abs/pi),
18                              \g3, ampEnvf3.at(angle.abs/pi),
19                              \g4, ampEnvf4.at(angle.abs/pi),
20                              \g15, ampEnvf5.at(angle.abs/pi),
21                                          //access interpolated values for bandwidths
22                              \bw1, bwEnvf1.at(angle.abs/pi),
23                              \bw2, bwEnvf2.at(angle.abs/pi),
24                              \bw3, bwEnvf3.at(angle.abs/pi),
25                              \bw4, bwEnvf4.at(angle.abs/pi),
26                              \bw5, bwEnvf5.at(angle.abs/pi)
27
28              );
29              vAccum=vAccum+angularVel        ;
30              vSteps=vSteps+1;
31      };
```

Line 4 sets the fundamental frequency to values between 100 and 166 for velocity values between 0 and 5. Likewise line 7 sets the amplitude. If the velocity remains lower than 0.007, no sound is produced, which avoids the production of sound due to noise in the input data when the wheel is standing still. The current formant frequencies, amplitudes and bandwidths of the vowel sound are passed to the `Synth` in lines 10 to 26. To determine their values for the current angle, they are read from the envelopes. The absolute values for the angle lie between 0 and π. These values are mapped to times t between 0 and 1 by dividing the current angle by π. The

values for the formant frequencies, amplitudes and bandwidths are found by reading the envelopes at the time t.

The sound events generated for each change in rolling direction and for each highest point of the bars are defined in the `SynthDef` `MinMaxGrain` and `BarOnTopGrain`:

```
//sound event for changes in rolling direction
SynthDef("MinMaxGrain", { arg i_out = 0, pan=0, freq1 = 300, dur = 1.5,
        dur=0.5;
        env = EnvGen.kr(Env.perc(0.003, dur, 1, -10),doneAction:2);
        synth=FSinOsc.ar(f0);
        Out.ar(i_out, Pan2.ar(synth, pan,amp) * env);   })

//sound event for each highest point of the bars
SynthDef("BarOnTopGrain", { arg i_out = 0, pan, freq1 = 500, dur = 0.01;
        var env;
        env = EnvGen.kr(Env.perc(0.003, dur, 1),doneAction:2);
        Out.ar(i_out, Pan2.ar(FSinOsc.ar(freq1), pan) * env);   })
```

These are similar to the definition for the sound event that is produced every time a bar touches the floor in the Cartoonification approach (see subsection 5.4.2). The duration of the `MinMaxGrain`, which is produced for every change in the rolling direction, is longer however and its frequency is adapted to the rotational angle of the wheel in each sound generation. Due to the longer duration the resulting sound is reminiscent of a xylophone. The `BarOnTopGrain` only differs to the sound in the Cartoonification approach in its (fixed) frequency.

The method `repMinMax` produces a sound event of the type `MinMaxGrain` for each change of rolling direction

```
1 //generates sound event for each incoming min. and max. / change in the rolling direction
2 respMinMax = {|t,r,msg|
3    var vAverage, tDiff, currentAngle, amp;
4
5    currentAngle=msg[3];
6    tDiff= msg[1]-tLastMinMax;         \\time since last change of rolling direction
7    vAverage=vAccum/vSteps;            \\average velocity
8    amp=(vAverage*tDiff).abs.linlin(0.005, 2,0,1)**3; \\amplitude controlled by the above
9
10   Synth.new(\MinMaxGrain,[\f0,currentAngle.abs.linlin(0,pi,400,800),\amp, amp);
11   vAccum=0;
12   vSteps=0;
13   tLastMinMax=msg[1];}
14   };
```

The amplitude of the sound event is adapted to the temporal distance and the average velocity between the last and the current change of rolling direction as explained in subsection 5.3.4 (line 8) and the frequency of the sound event is set to a value between 400 and 800 according to the wheel's rotational angle, thus giving information about the position the wheel is in. Lines 11, 12 and 13 update the values that are necessary to compute the average velocity and temporal distance between two changes of rolling direction, for the next step.

The method `respBarMinMax` generates a clicking sound of the type `BarOnTopGrain` for each highest point of one of the bars:

```
//produce sound when bar 1 or bar 2 is on top of the wheel
 respBarMinMax = {|t,r,msg|
    var amp;
    Synth.new(\BarOnTopGrain,[\amp, 0.5]) ;}); }
    };
```

5.4.4 Event-based sonification

This subsection covers the content of the class `SonEventBased`, which implements the Event-based sonification as described in subsection 4.5.4. In this sonification approach the 'continuous' rolling sound is a stream of sound events created by the generation of one such event every time the rotational angle of the wheel (see Figure 2.2) trespasses one of the 30 steps each half circle is divided into. Sound events similar to those of the 'continuous' rolling sound but with higher amplitude are generated every time one of the bars touches the floor and a different sound event, the frequency of which is determined by the angle of the wheel at the time, is produced for every change of rolling direction.

A `SonEventBased` objects receives the current angle of the wheel in its input OSC message. Again the objects for the calculation of the lowest points of the bars, of the changes of rolling direction and of the angular velocity are inherited from the super-class `Sonification` (see 5.4).

The method `respAngle` receives the current angle of the wheel in the OSC message `msg` and generates a grain of the 'continuous' rolling sound whenever one of the 30 steps has been traversed:

```
1 respAngle = {|t,r,msg|
2       var step;
3       currentAngle=msg[2];
4       step= (msg[2].abs *steps/pi).floor, synth;
5       if(step== oldStep,{
6               },{     Synth.new("EventGrain",[\step, step, \amp, 0.5]);
7                       oldStep=step;
8               });
9 };
```

Line 4 assignes the next lower `step` to the incoming rotational angle of the wheel (see Formula 4.11) and line 5 tests if this step is different to the previous step. If they are equal a sound event has already been generated for this step; otherwise a new step has been trespassed and a sound event is generated in line 6.

After experimenting with a range of different sound events, including instrumental sounds, we chose the following relatively simple clicking sound as a grain for the 'continuous' rolling sound.

```
1  SynthDef("EventGrain", { arg i_out = 0, pan, step=0, freq, dur = 0.01, amp=1;
2       var env;
3       //determine amount of steps
4       steps=30;
5       freq = (step*freqStep)+100;
6       env = EnvGen.kr(Env.perc(0.003, dur, 1),doneAction:2);
```

```
7       Out.ar(i_out, Pan2.ar(FSinOsc.ar(freq), pan, amp) *env)
 8 }).send(s);
```

Line 4 determines into how many steps every half circle is divided (See Figure 4.11). Line 5 sets the frequency of the sound event to the frequency associated with the step it is generated for (see formula 4.10). The resulting 'continuous' rolling sound that represents the changing angle is a stream of clicking sounds.

When a bar reaches a lowest point, it touches the floor and a sound event with the same underlying structure as the 'continuous' rolling sound event, but with a higher amplitude is produced. The method `respBarMinMax` implements the generation of these sounds.

```
1 respBarMinMax={ |t,r,msg|
2       var step;
3       step= (currentAngle.abs *steps/pi);
4       Synth.new("BarOnFloorGrain",[\step, step]);
5 };
```

The value saved in `step`, which determines the step that is associated with the current angle, is not rounded down to an integer this time. Thus the frequency of the produced sound can lie between the frequencies associated with two neighbouring steps.

The recorded sound of a triangle, which is played for each change of the rolling direction, is defined in the `SynthDef MinMaxGrain`:

```
1 SynthDef("MinMaxGrain", { arg i_out = 0, pan, , dur = 1, amp=0, currentAngle=0;
2    var env;
3
4    // buffer b14 contains a triangle sound
5    env = EnvGen.kr(Env.linen(0.01, 2, 0.01,1),doneAction: 2);
6    Out.ar( i_out,Pan2.ar(
7              PlayBuf.ar(1, b14, BufRateScale.kr(b14)*(currentAngle.linlin(0,pi,1,2))),
8              pan,amp)*env)
9 }
```

The triangle sound is played from the buffer `b14` (line 7). The third argument of `PlayBuf` determines the playback velocity of the buffer, which in turn determines the pitch of the sound that is played. In this sound definition the playback velocity, and thus the pitch of the sound event, is controlled by the absolute value of the angle (`currentAngle`). Therefore the pitch of the triangle sound is higher the larger the value of the angle.

One of these sound grains is generated by the method `respMinMax` for every change of rolling direction.

```
1 respMinMax = {|t,r,msg|
2       var vAverage, tDiff,amp,a;
3
4       tDiff= msg[1]-tLastMinMax;                      //msg[1] contains time of occurence
5       vAverage=vAccum/vSteps;                         //compute average velocity
6       amp=(vAverage*tDiff).abs.linlin(0.005, 2,0,1)**3;
7
8       Synth.new(\MinMaxGrain,[\amp, amp, \currentAngle, currentAngle.abs]) ;
```

```
9       vAccum=0;
10      vSteps=0;
11      tLastMinMax=msg[1];
12 };
```

The volume of the sound is controlled by the average velocity and the temporal distance between the current and the last change of rolling direction in line 6 (see subsection 5.3.4). The rotational angle of the wheel at the time of the change in rolling direction is passed to the `MinMaxGrain` to allow the adaptation of the pitch as discussed above (line 8). The remaining lines update `vAccum` and `vSteps` for the calculation of the average angular velocity between two changes of rolling direction.

Every incoming angular velocity value is also used to update these variables:

```
1 respVelocity = {|t,r,msg|
2               vAccum=vAccum+msg[2];           //msg[2] contains current velocity value
3               vSteps=vSteps+1;
4       };
```

6 Studies and Evaluation

After finishing the implementation of our four sonification approaches, namely the Direct-data approach, the Cartoonification, the Vowel synthesis sonification and the Event-based sonification, we chose the Vowel synthesis sonification and the Event-based sonification to conduct a comparison between a performance with and without sonification. The study gives an indication about the correctness of our hypothesis, which claims that a performance improvement can be achieved for a wheel performer through the use of sonification. In sections 6.1 we describe the nature and procedure of the conducted experiments. In section 6.2 a statistical analysis of the data that was collected is provided. In this analysis the performance of the participants with and without sonification is compared quantitatively, based on a measure of performance that we introduce. Section 6.2.3 explores whether the data and the statements of the participants yields additional information that can help to create further assumptions regarding the effect of real-time auditory feedback on the performance of German wheel moves. Section 6.3 comments on some further details, such as choices we made concerning the setup and the task the participants of the study had to carry out.

6.1 Procedure

As only a limited amount of time and participants were available for the experiments, we chose the Vowel synthesis sonification (described in subsections 4.5.3 and 5.4.3) and the Event-based sonification (described in subsection 4.5.4 and 5.4.4) for our study. The choice of those two approaches was motivated by the fact that they are the most adapted to the tasks the performers had to fulfil during the experiments (see subsection 4.2.1).

The real-time sonification system was tested with two different target groups namely a group of seven German wheel novices and a group of four experts. For logistical reasons the set up was slightly different for the two groups. In both cases experiments were conducted under three different conditions: without sonification, with the Vowel synthesis sonification and with the Event-based sonification. The experiments with the novices were carried out in a relatively small laboratory and with the use of headphones. (Even though for a more general application of the system the use of cable headphones would not be practical, they did not disturbed the participants in the execution of the particular task they had to carry out.) The experiments with the experts took place in a sports hall and loudspeakers were used to provide the audio feedback. All participants were given the same project description, pre-questionnaire

and post-questionnaire, the content of which can be found in appendix F.
The task of the participants was to perform a basic rock, facing the rolling direction of the wheel (see section 2.1 and Figure 2.3). The aim was to make the height of the swing symmetrical, thus to let the swing come up to the same height in the front and the back.
The task was performed three times, once without sonification, once with Vowel synthesis sonification and once with Event-based sonification. The order of the three conditions was randomised. As the task is quite demanding for novices, they were allowed a time span of ten minutes without sonification, to understand and practice the basic rock. Otherwise the procedure of the experiments was the same for experts and novices: Before each of the runs with or without sonification the participants were given two minutes to practice under the forthcoming condition. For each run the task was then carried out for a time span of approximately two minutes, during which data was recorded for later evaluation.

6.2 Data analysis

To compare the performance of the participants under the different conditions, we need to be able to appraise how well the task was performed. For this purpose we explore some statistical values the data provides, introduce a quantitative measure that is based on these values and conduct a Student's t-test to analyse whether the mean of this measure is significantly different under the three different conditions, namely without sonification, with Event-based sonification and with Vowel synthesis sonification. All calculations that forme part of the analysis were done in the programming language Python.

6.2.1 Measure of performance

As remarked in the previous section, the task of the performer on the wheel was to bring the swing up to equal heights in the front and the back and thus to keep the difference between the absolute angles measured at two consecutive changes of rolling direction as low as possible. Accordingly, the heights measured (in radians) every time the wheel changes its rolling direction, are the data that the appraisal of an executed move is based upon. As changes of the rolling direction are indicated by an extremum of the angle (see subsection 4.4.3), we refer to them as extrema in the following.

Summaries of the swing heights for all experts and novices can be found in appendix F.4 in the form of tables that contain the mean, median and standard deviation for each participant and condition, and in the form of the heights plotted against the index of the changes of rolling direction. The graphics also give information about a set of corrected outliers. The participants needed up to three and a half swings to

Table 6.1: $\Delta\varphi_{eval}$ Experts

$\Delta\varphi_{value}$	mean	median	standard deviation
without	0.194	0.188	0.063
Event-based	0.087	0.071	0.062
Vowel	0.185	0.152	0.130

Table 6.2: $\Delta\varphi_{eval}$ Novices

$\Delta\varphi_{value}$	mean	median	standard deviation
without	0.109	0.105	0.049
Event-based	0.102	0.096	0.039
Vowel	0.134	0.127	0.049

find their preferred height of swing and to stop at the end of the experiment. This is equal to seven changes in rolling direction. The first and last seven detected extrema are therefore not included in the evaluation and illustration of the data.

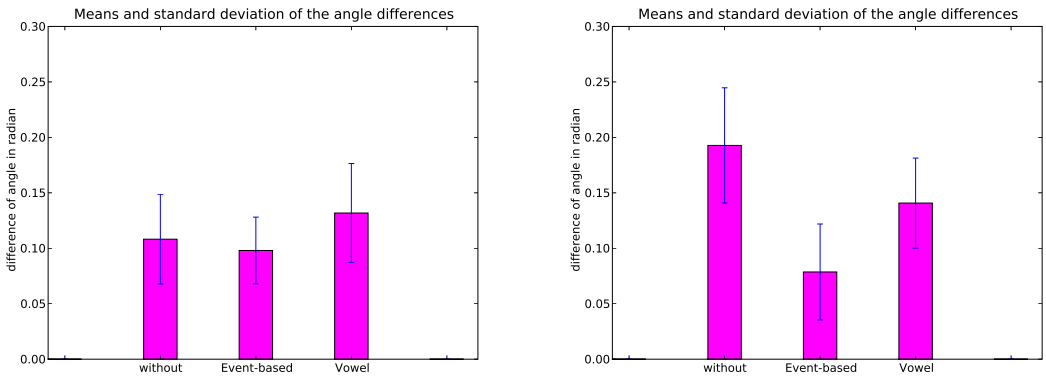
When the swing is brought up equally high in the back and the front, the wheel is rotated about the same angle away from the initial position shown in Figure 2.2. In that case the angles measured for two successive extrema, differ only in their sign. For this reason we chose the difference of the absolute angles measured at two consecutive extrema as our measure of performance:

$$\Delta\varphi_{eval} = ||\varphi(extremum(i))| - |\varphi(extremum(i-1))|| \tag{6.1}$$

Two consecutive extrema always consist of one minimum and one maximum. A smaller value of $\Delta\varphi_{value}$ indicates a better performance of the task.
Appendix F.4 contains lists of the mean, the median and the standard deviation of $\Delta\varphi_{eval}$ for each of the experts and novices under the three conditions, namely without sonification, with Event-based sonification and with Vowel synthesis sonification. For each participant it also contains a graphical comparison of the different conditions, in which $\Delta\varphi_{eval}$ is plotted against the index of the extrema for each condition.

Tables 6.1 and 6.2 give an overview of the performance of the experts and novices under the different conditions by providing the according means, medians and standard deviation of the measure of performance $\Delta\varphi_{eval}$. The means and medians are also compared in the bar charts in Figures 6.2b and 6.2a. The charts do not show large differences under the three conditions for novices. For experts however a far larger difference is observable, in particular between the conditions 'without sonification' and 'with Event-based sonification' (see Figure 6.2b).

Figure 6.1: Means and standard deviations of the measure of performance $\Delta\varphi_{eval}$ without sonification, with Event-based sonification and with Vowel synthesis sonification for novices and experts



(a) novices

(b) experts

6.2.2 Statistical significance

The improved performance, which we observe for the experts when sonification was used, may have occurred due to the influence of the sonification. To statistically support this assumption we have to show that the different means of our measure of performance $\Delta\varphi_{eval}$ for the three conditions are not coincidental. To allow a statement about the significance of this difference, we therefore conducted a Student's t-test. The t-test offers a statistical instrument to judge whether two means are significantly different. When two distributions are thought to have the same variance, but a different mean, the t-test returns the test statistic t from which a probability p arises. This probability states how probable it is that the difference of the means is coincidental. If p is lower than 5 % ($p<0.05$), the difference between the means of the two distributions is seen as significant and the null hypothesis of the t-test, which claims that the difference of the mean is coincidental, can be rejected. There are two different versions for the t-test, one of which is for two independent samples[1] and the other for dependent samples. In our case we used the test for dependent samples[2], because the same group of participants was tested twice. For further information about the t-test please refer to [PFTV92].
As input distributions we used the means that were observed for each expert without sonification and the means that were observed with Event-based sonification. The following values were obtained for t and p:

$$t = 3.596 \qquad\qquad p = 0.036$$

We can therefore assume that the coherence between the use of the Event-based sonification and the lower mean value of the measure of performance was not coincidental

[1] SciPy/Python ttest_ind
[2] SciPy/Python ttest_rel

and that the Event-based sonification influenced the performance positively. For the Vowel synthesis we received the values $t = 0.870$ and $p = 0.448$ and can therefore not reject the null hypothesis that the lower mean is coincidental. The t-test between the condition 'without sonification' and 'with Event-based sonification' for all participants, including experts and novices led to the values $t = 1.851$ and $p = 0.094$.
All conclusions, including those that suggest less effectiveness of sonification for novices and for the Vowel synthesis approach have to be mediated by the fact that the study includes only small numbers of participants.

6.2.3 Explorative data analysis and observations

Due to the results of the statistical analysis in section 6.2 we conclude that the Event-based sonification can influence the performance of German wheel experts positively. This raises the question why we could not find such a significant influence for the novices and for the Vowel synthesis sonification. To answer this question we re-examine the data and some coherence we observed throughout the experiments. This may reveal the reasons for the differences described above, which may inspire changes in the implementation or be of interest for a future series of experiments. The content of this section consists of considerations we made rather then statistically proven facts.

For all novices, who took part in the experiments, this was the first time they used a German wheel. This and the fact that the experiments took place in a relatively small laboratory contributed to the nervousness that most of the novices showed. Several of them stated that they were afraid to damage equipment located in the room and that they were finding it difficult to use the sonification, because even keeping the balance in the wheel was challenging. One interesting example is novice number 5, who performed the experiment in the order: without sonification, Event-based sonification, Vowel synthesis sonification. The impression that the participant was rocking higher in every run of the experiment (apparently because her anxiety was decreasing and she was increasingly enjoying herself) was confirmed by the data (see appendix F.4). As a higher rock is harder to control however, this also led to higher values for $\Delta\varphi_{eval}$ (see Figure F.-2e). Several other novices, in particular the ones who stated that they found the task relatively difficult, as for example novice 2, seemed to be made more insecure by the sonification and the fact that they were being recorded. This was noticeable due to reactions such as oppressed laughter. According to our observations their performance worsened when they had finished their 'training time' with the sonification and knew that the data and a video of their performance were now being recorded.
Generally the impression arose that those novices who were more physically sure of themselves without being overexcited about using a German wheel for the first time benefited most from the sonifications. For future experiments this leads to the conclusion that the participants should get the opportunity to try a German wheel for a longer time before participating in the study. This could for example be implemented by offering a workshop for all participants. The tryout session and the

experiments should then be conducted on different days as symptoms of fatigue may also influence the results significantly.

Another discovery we made and which may partly explain why the results observed for the Event-based sonification are more promising than the ones for the Vowel synthesis sonification, is that some of our sound settings were not optimal. Expert 4 and novice 6 for example did not improve their performance with the Vowel synthesis sonification, even though they responded well to the Event-based sonification (see Figures F.-1d and F.-2f). Both of them mentioned that the sound events generated for each change of rolling direction were not loud enough. This was caused by the linking of the volume to the average velocity and time difference between two changes of rolling direction (see subsection 4.4.4). The swing of both, novice 6 and expert 4 were relatively low and therefore slow and with a small time difference between changes of rolling direction in comparison to other participants of their group. This may also have affected some of the other novices whose swing was rather cautious. This leads us to conclude that for future implementations the sound settings such as the volume of each sound layer should be adaptable to the users' needs at runtime. This request is supported by the fact that, according to their own statement, the participants used different parts of the sound to support the task. Most of them concentrated mainly on the sounds generated for the changes of rolling direction and adapted the height of their next swing according to the error in their last swing. Some of the participants however concentrated more on the continuous feedback, as it allowed an immediate assessment of their current position.
We also asked the participants if they had any kind of musical education, but could find no direct connection between their musical experience and their performance in the experiment. Additionally, the participants gave different opinions about the aesthetics of the sounds. Nine out of eleven participants found the Event-based sonification acoustically more pleasing then the vowel synthesis. One of the participants used the vowels to write down how the Vowel synthesis sonification sounded without being advised to use this advantage of the vowel synthesis sonification.
Besides these remarks concerning the performance of the participants it is interesting that several people who observed the experiments stated of their own accord that they found it easier to judge the quality of a move, when they had the sonifications to support their formation of opinion. Furthermore several of the novices stated that after a while the sonification in combination with the repetitiveness of the swing and the physical action had a soothing almost meditative effect and could be used for therapeutic purposes.

6.3 Further remarks and suggestions

To maintain the height of the rock on the same level over a longer period of time is not a task which is usually performed when training to use a German wheel. However, many exercises do include the task to judge the height of the bar accurately or to compare two heights as for example the heights that a bar reached in two different

executions of the same move. In case of our experiments we were limited in time and space. The exercise we chose offered a simplified way of observing whether the judgement of height can be improved by the sonification. Additionally most wheel performers are specialised on different sets of moves; however the basic rocking motion is a move, which all experts have performed in their training and that each participant was able to perform.

A few other changes and additions to the experiment procedure could improve the conditions for future experiments or lead to further statements about the effect of the sonification on the performance of the participants:

- As we mentioned in section 6.1 logistical constraints led to a slightly different set up for novices and experts. For future experiments it is desirable to provide a setup which is exactly the same for both groups and a more extensive time frame to allow a longer period for the novices to become accustomed to the German wheel.
- The study was conducted with relatively few participants which may be another confounding variable, thus an alternative explanation for some of the findings.
- Longer and more extensive experiments with more adaptable sound settings should be undertaken to assess how the sonification can contribute to a better performance in a wider range of tasks.
- For some of the centralised moves, which are usually more controlled than the decentralised moves, the control of the wheel's velocity could lead to an alternative measure of performance, if the task was to keep the velocity as constant as possible.
- As mentioned in the previous section, the sounds that are produced for events such as a change of rolling direction, are perceived after the event happens, whereas the continuous feedback allows the performer an assessment of the current state. Thus, the sound events allow a slightly time delayed judgement of the performed move, whereas the continuous rolling sound allows to change the move while it is carried out. To observe a possible positive effect of the feedback given through the sound events, a move therefore has to be repeated. To judge the influence of the feedback on more complex moves long-term studies have to be conducted.
- The finding that the sonification was of more use for the experts suggests that sonification may contribute in particular to the refinement of known skill, which is another aspect that should be investigated further.
- A study of how users adjuste sonification parameters during runtime following an initial period of training would make an interesting future study.
- Another direction for future research could be to examine the meaning of the significant real-life differences that were found between moves carried out without sonification as compared with Event-based sonification. (What does the observed difference mean in real life? Would it be worth using sonification

equipment to ascertain this difference?) This could also include a longer and more detailed questionnaire for the participants.

7 Discussion and conclusion

Hitherto we described the implementation and evaluation of different sonification approaches, which are used to sonify the motion of a sports equipment called the German wheel, in order to provide real-time audio feedback to a performer carrying out moves on the wheel. Our main hypothesis is that such acoustic real-time feedback can contribute significantly to the skill learning of the performer. Four different sonification approaches namely the Direct-data approach, the Cartoonification approach, the Vowel synthesis approach and the Event-based approach were presented. In addition an exempla study was conducted with the Vowel synthesis and Event-based sonification. The results of this study suggest that the Event-based sonification can support significant performance improvement. More extensive research should be undertaken to improve the functionality of the real-time feedback system and to confirm our results for a larger sample. For this purpose, section 7.1 summarises our solutions for each of the sub-tasks the project was divided into, namely data acquisition, feature extraction, sonification and evaluation, and provides suggestions for further improvement. Finally section 7.2 gives an outlook into possible future research and related fields of application for which our results may be of interest.

7.1 Discussion of the subtasks

7.1.1 Data acquisition

In section 4.1 we described a range of different sensors that were considered for the calculation of the rotational angle of the wheel, which is a feature that carries a high amount of information about the wheel's motion in a straight line. After failing to detect the wheel's location with several different approaches that use the acceleration of the wheel as input data (see subsections 4.3.2 - 4.3.4), we succeeded in calculating the rotational angle using the data collected with a magnetometer (see subsection 4.3.1). This approach relies on the measurement of the earth's own magnetic field. As long as the sender's range is high enough (see subsection 5.1.1) and the magnetic field is not distorted by other magnetic sources, the approach works well and no visible or audible latency is observable. However, as magnets and magnetic fields induced by electric current flow are present in many places (take the magnets in loudspeakers and the currency flow in high voltage lines for example), this approach is not robust in every environment. More research is therefore required to evaluate alternative sensors or the possibility of combining several different sensors to achieve

more robust results.

7.1.2 Feature extraction

We categorise the features that we use for the sonifications into two groups: Continuous features such as the rotational angle of the wheel and the angular velocity provide continuous information about the wheel's motion, whilst discrete features such as changes in the rolling direction and lowest or highest points of the bars mark the occurrence of time discrete events. Additionally, features such as the average velocity and the time difference between two changes of rolling direction give information about the significance of a completed event.
These features do quite adequately represent a general straight line German wheel move, to which we limited our considerations. However, As mentioned in subsection 4.3.1 the magnetometer approach, that we use to determine the position of the wheel, does have the potential to provide the full 3-D orientation of the wheel. This is relevant for possible sonifications of spiral moves (see section 2.1), for which an approach based on different features is desirable.
Indeed the design of new features or adaptation of known features to the needs of a specific move may lead to a more adequate acoustic feedback for that move. Finally we think that for some applications predictive calculations could be of use to notify the performer of an upcoming event, as opposed to informing the performer that an even has occurred.

7.1.3 Sonification

For our real-time sonification system we designed four different types of sonification. The Direct-data sonification directly sonifies the input data and leaves the interpretation of the resulting sound to the performer. The three other approaches rely on knowledge about the wheel's rolling motion which is supplied by the features that we derive from the magnetometer data. The Cartoonification, Vowel synthesis sonification and Event-based sonification generate a continuous rolling sound from the continuous features and discrete sound events for the discrete features. The Cartoonification approach artificially reproduces and amplifies a natural rolling sound. The Vowel synthesis sonification uses synthesised vowel sounds to indicate the rotational angle and the velocity of the wheel, and sound events to indicate changes in the rolling direction. The Event-based sonification produces a continuous stream of sound events, whose frequencies indicate the rotational angle of the wheel while the density of sound events represents the angular velocity of the rolling wheel. Additionally, changes in the rolling direction are marked by a sound similar to the sound of a triangle instrument. We did not experience problems with any of the sonification approaches concerning an audible latencies of the auditory feedback.
In our study we observed very different results for the Vowel synthesis sonification and the Event-based sonification even though they are based on a similar set of fea-

tures. This shows that unequal sound settings can lead to very different results. A wider range of sonifications approaches should therefore be designed and tested to elicit their advantages and disadvantages. Ideally, the advantageous components from these designs could then be combined to create a considerably more effective acoustic feedback. Our considerations in section 6.2.3 also suggest, that different sound settings may be appropriate for differing wheel moves and performers. For this reason in a user oriented application different sonifications and their sound layers could be made more adaptable to the needs of specific users. One way this could be achieved is by adjusting each sonification to include a wider range of (partly redundant) sound layers, which can be turned on and off depending on the performed move and the aspects of the move that the performer wishes to emphasise.

7.1.4 Evaluation

After the implementation of our real-time sonification system we conducted a study to discover whether the additional acoustic feedback can improve the performance of a move carried out on the wheel. The task the participants (seven novices and four experts) had to carry out was the performance of a basic swing (see section 2.1) that reaches the same height in the front and in the back. We took the difference in height at the front and the back of the swing as a measure of performance. For novices we could observe no significant influence of the sonifications. For the experts however the evaluation parameter showed a significantly lower mean, when the Event-based sonification was used. This confirms our hypothesis that additional acoustic feedback can significantly improve the performance of a German wheel move.
To further circumstantiate this statement more extensive psychophysical experiments, that focus on different aspects of the task and the sonification, should be carried out. A longer adaptation time to the wheel should be granted to the novices. A range of different tasks, that focus on different sub-tasks, which a wheel move includes, should be tested with different sonifications and a long term study should be undertaken to allow the judgement of the skill learning of more complex moves. Furthermore it should be tested if it can be statistically supported that the use of sonification can assist a trainer in the monitoring task.

7.2 Outlook

The study conducted with the Vowel synthesis and Event-based sonification (see section 6.2) confirmed our hypothesis that sonification used as acoustic real-time feedback can significantly improve the performance of a German wheel move.

The next step should be a more detailed study with the current system, to clarify the findings from these relatively small numbers of participants, with more analysis of results. This study could be undertaken with a version of the real-time sonification system, which is adjusted to support the idea of runtime adaptability.

Furthermore the utility of sonification for the support of synchronisation tasks for multiple wheel performers could be investigated and additional analysis may show to what extent different sonification approaches can support a trainer in the monitoring process. Besides the real-time application of the sonification system, recorded sonifications of moves or movement sequences may prove to be of use for skill learning. A performer could, for example, listen to a sonification, that was recorded while a sequence was performed, to support the memorisation of the same sequence. Another way of using recorded sonification could be to simultaneously watch and listen to a move carried out by a more highly skilled second performer. When the first performer then tries to imitate the move, she or he can use the sonification system and hear the difference to the performance of the second performer in real-time, while a visual analysis can only be done consecutively.
Our findings concerning the usefulness of real-time audio feedback for wheel performers also suggest the possibility that the results may be transferable to other sports disciplines. This could be of particular interest for disciplines that share certain characteristics with wheel gymnastics. Many moves in floor gymnastics, trampolining, half pipe related sports disciplines or swinging trapeze for example are also highly time critical, have to be performed when a change of direction is happening and restrict the use of vision due to the fast changes of perspective that the performer experiences.
Besides the use of wheel motion sonification as a tool to improve skill learning, we also anticipate several other possible applications. Using the tactile senses, the wheel can serve as a spatial reference because physical contact with the wheel can be kept constantly. In a well supervised environment a simplified version of wheel gymnastics may therefore prove to be suitable for visually impaired people. As in this case the visual channel is (partially) unavailable, more importance is attached to other types of feedback. Additional acoustic real-time feedback may therefore make this sports discipline more accessible for visually impaired people.
The relaxing effect, which the participants of the study experienced as a result of the combination between the repetitive physical action and the resulting sonification, also suggests the application of the real-time sonification system for therapeutic use.

In summary, the implementation and evaluation of our real-time sonification system for German wheel motion confirmed that real-time audio feedback can provide a means to significantly improve sports movement. Research in the young interdisciplinary field of movement sonification, that combines sports science with computer science, is only at its starting point. Our results are very promising and indicate that further investigation will prove to be very valuable.

Appendix A

Physics of the German wheel

This appendix introduces some mathematical concepts that are related to the motion of the German wheel and which some of the calculations presented in Chapter 4 are based on.

A.1 Cycloid, the trajectory of a point on a rolling circle

A cycloid is the path, that a point $P = (x, y)$ on the circumference of a circle or wheel with radius r describes, when the wheel rolls in a straight line (see Figure A.1). If φ is the angle of rotation (see Figure A.1), the parametric equation for the location of P is:

$$\vec{p}(t) = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r \cdot (\varphi - \sin(\varphi)) \\ r \cdot (1 - \cos(\varphi)) \end{pmatrix} \tag{A.1}$$

Due to this correlation, provided that the starting position is known, the rotational

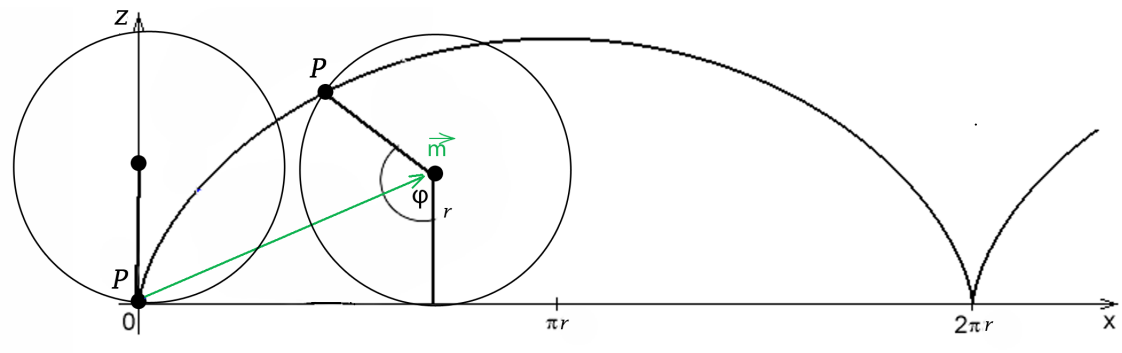


Figure A.1: Cycloid: the curve that a point on the circumference of a wheel draws, when the wheel is rolling in a straight line. The rotational angle φ is positive if the rotation is clockwise.

angle φ of P gives us direct information about the location (x, y) and vice versa. Moreover the x-component m_x of the location of the wheel centre $\vec{m}$(t) and the

rotational angle have the following dependency:

$$m_x = r \cdot \varphi \tag{A.2}$$

This is easily understandable considering the example of a full rotation. The wheel centre travels a distance equivalent to the circumference of the wheel ($\vec{m} = 2 \cdot \pi \cdot r$) and it rotates by $\varphi = 2 \cdot \pi$ (equivalent to 360°).
In line with the common standard, a positive rotational angle is defined by the right-hand-rule as follows: If the thumb of the right hand is pointing into the direction of the axis we are rotating around and the curved fingers indicate the direction of rotation, the angle is positive. The angle in Figure A.1 is positive since we rotate around the y-axis, which points rearwards.

A.2 Numerical differentiation

Using a definition from physics, the derivative is a measure of how a function changes as its input changes. The most relevant examples for us are the first and second derivative of the location with respect to time (linear velocity and acceleration) and the first and second derivative of the rotational angle with respect to time (angular velocity and acceleration). In practice it is often not possible to express the values of the incoming data through a closed formula and therefore the derivation can not be found analytically. In such a case a technique known as numerical differentiation is used to estimate the derivative. We find the numerical derivation of the data values x by calculating the differential quotient of two consecutive values and the corresponding times:

$$\vec{x}'[n] = \frac{\Delta x}{\Delta t} = \frac{x(t_n) - x(t_{n-1})}{t_n - t_{n-1}} \tag{A.3}$$

In some cases it can be more adequate to have a larger distance between the two used data points as, while incresing the error, this also smoothens the curve of the derivation:

$$x' = \frac{\Delta x}{\Delta t} = \frac{x(t_n) - x(t_{n-k})}{t_n - t_{n-k}} \tag{A.4}$$

Here n is the index of the incoming value and, t_n is the time when the nth value was measured (e.g $x(t_0)$ would be the first incoming angle value, which was measured at the time (t_0) etc.). The significance of these equations becomes clear when they are used for the calculation of the wheel's angular velocity in subsection 4.4.2.

A.3 Measurement of acceleration

Acceleration is the first derivative of velocity and therefore its change over time. This definition can lead to a misunderstanding when acceleration is measured in a real-life situation, for the measured acceleration of an object lying still (e.g. on a table)

does not equal zero. To understand this consider Newton's second law of motion: "Change of motion is proportional to the force applied" $F = m \cdot a$. If the object was not restricted by the table, a force caused by gravity ($F_{fall} = m \cdot g$) would be effecting the object and it would be falling towards the geo-centre. Yet, the object is not moving and therefore, due to Newton's law, the over-all force has to equal zero. Thus, there must be a force $F_{table} = m \cdot a$ opposing F_{fall}:

$$F_{total} = F_{fall} + F_{table} = m \cdot g + m \cdot a = m \cdot (g + a) = 0 \qquad \Rightarrow a = -g$$

The mass can not be zero, which implies that the measured acceleration is the inverse of gravity. Whenever acceleration is measured, this contribution of gravity to the acceleration data has to be taken into account.

Appendix B

Task data analysis

Name of Data-set: German wheel data
Date: 04.08.2008
Author: Jessica Hummel

File: not decided yet
Format: no exact information yet, the file will consist of a time-series of sensor data, measured on the rolling wheel (e.g. acceleration data)

B.0.1 Scenario

A German wheel is a sports equipment consisting of two rims of about 2-2,4m diameter connected by 6 bars of about 40 cm length. A performer carries out acrobatic moves in the wheel while it is rolling. In wheel gymnastics moves are divided into different categories. Decentralised moves are carried out on one of the bars. If the performer (on the bar) is on one side of the wheel and not on the vertical axis through the centre of gravity of the wheel (the zero axis), the performer's weight pushes down the side of the wheel and the wheel starts rolling. In more advanced moves the performer uses the speed and momentum of the wheel to perform moves such as spins around the bar. Many of these decentralised moves can only be performed in a short time window, when the wheel momentarily stands still or slows down, because it is either changing the rolling direction or passing the zero axis. The data measured consists of a time series of sensor data about the wheel's motion. This data will be used:

- to support the skill learning of the performer by giving real-time feedback about the wheel's motion.

- to find a way of representing the data, that allows direct comparison of two different executions of the same move.

The Keys:

Question:	How well is the move performed? determined by: How fast is the wheel? How high is the acceleration? Where is the performer(height)? When does the wheel change direction or reach a minimum of speed? What is the maximum velocity during the execution of one move?
Answer:	current velocity(V) in m/s current acceleration(a) in m/s^2 current distance from floor (h) in radian or m point in time (t) in s maximum velocity (V_{max}) in m/s
Subject:	velocity acceleration height point in time of direction change or passing the zero axis max. velocity
Sounds:	pitch rise for velocity increase ?

B.0.2 TaDa

The Task

Generic Question:	How big is it (referring to V ,a ,h ,(V_{max}))? How does it develope (V, a, h)? When is it (t(V_{max}))?
Purpose:	confirm (point of time) judge (execution) compare (execution) alert (time-critical moment) remember (sequence of moves) engage (audience)
Mode:	interactive
Type:	continuous (velocity, acceleration, height) discrete (point of time, max velocity)
Style:	presentation

The Data/Information

Level:	local
Reading:	conventional (direct for velocity?)
Type:	ratio
Range:	velocity range: 0-max possible velocity acceleration range: 0-max possible acceleration height range: 0-diameter of wheel point of time: 0-maximum sequence length max velocity: 0-maximum possible velocity
Organization:	time

The Data

Type:	ratio
Range:	velocity range: 0-max possible velocity acceleration range: 0-max possible acceleration height range: 0-diameter of wheel point of time 0-maximum sequence length max velocity: 0-maximum possible velocity
Organisation:	time

Appendix C

Additional approaches for the calculation of the rotational angle

In this appendix two additional approaches for the calculation of the wheel's rotational angle namely the Cycloid approach and the Neuronal network approach are presented. They were suggested, but not fully implemented as the magnetometer approach proved to be adequate for our purposes.

C.0.3 Cycloid approach

The idea of the cycloid approach is to integrate the acceleration measured by one accelerometer and to use the additional knowledge that the accelerometer has to travel on a cycloid curve to compensate for drift (see A.1 for more information about the rolling curve). A further presumption is that the location, velocity and acceleration do not change drastically in one time-step. The algorithm for the computation of the wheel's location is as follows:

1. Start presuming that the rotational angle $\varphi(t = 0) = 0$ ($\rightarrow$ the location of the sensor in world coordinates $\vec{w}(t = 0) = (0, 0)^T$) and that the velocity of the accelerometer in world-coordinates $\vec{V}_w(t = 0) = (0, 0)^T$.
2. As the accelerometer is rotated with the wheel, rotate the measured data by $\varphi(t)$ to transform it into world coordinates (no rotation in the first step as $\varphi = 0$).
3. Subtract the gravitational component (see A.3), this is possible because the calculations are in world coordinates.
4. To find the new velocity $\vec{V}(t + 1)$ and location $\vec{w}(t + 1)$ of the accelerometer in world coordinates integrate the residual acceleration twice (with initial conditions $\vec{V}_w(t)$ and $\vec{w}(t)$).
5. Find the point P on the cycloid that is the closest to $w(t + 1)$.
6. Find the rotational angle $\varphi(t + 1)$ that the wheel is at if the sensor is on P.
7. Go to step 2.

C.0.4 Neuronal network approach

The aim of this approach is to build an artificial neural network (see [Bis95]) that for given accelerations a^*_{1x}, a^*_{1z}, a^*_{2x}, a^*_{2z} measured by two accelerometers that are attached to the wheel, returns the rotational angle of the wheel. In most cases an artificial neural network is an adaptive system that changes its structure based on information that flows through the network during a learning phase. In each learning step an ideal data-set of input and output data is presented to the network and the internal structure is changed to improve the output function. For this learning phase we require a set of data, consisting of input data (here the measured accelerations a^*_{1x}, a^*_{1z}, a^*_{2x}, a^*_{2z}) and corresponding output data (here acceleration a, velocity V and rotational angle φ of the wheel). To generate training data, the measured data for given a, V and φ is predicted by using the model developed in subsection 4.3.3. Formula 4.5 is the acceleration accelerometer 1 should experience in world coordinates. To predict the measured accelerations, these values have to be transformed into sensor coordinates by rotating them through $-\varphi$ degrees:

$$\begin{pmatrix} a^*_{1x} \\ a^*_{1y} \\ a^*_{1z} \end{pmatrix} = \begin{pmatrix} \cos(-\varphi) & 0 & \sin(-\varphi) \\ 0 & 1 & 0 \\ -\sin(-\varphi) & 0 & \cos(-\varphi) \end{pmatrix} \cdot \begin{pmatrix} a + \frac{V^2}{r}\sin(\varphi) - a \cdot \cos(\varphi) \\ 0 \\ \frac{V^2}{r}\cos(\varphi) + a \cdot \sin(\varphi) + g \end{pmatrix}$$

$$= \begin{pmatrix} \cos(\varphi) & 0 & -\sin(\varphi) \\ 0 & 1 & 0 \\ \sin(\varphi) & 0 & -\cos(\varphi) \end{pmatrix} \cdot \begin{pmatrix} a + \frac{V^2}{r}\sin(\varphi) - a \cdot \cos(\varphi) \\ 0 \\ \frac{V^2}{r}\cos(\varphi) + a \cdot \sin(\varphi) + g \end{pmatrix}$$

$$\begin{pmatrix} a\cos(\varphi) + \frac{V^2}{r}\sin(\varphi)\cos(\varphi) - a\cos^2(\varphi) - \frac{V^2}{r}\cos(\varphi)\sin(\varphi) - a \cdot \sin^2(\varphi) - g\sin(\varphi) \\ 0 \\ a\sin(\varphi) + \frac{V^2}{r}\sin^2(\varphi) - a\cos(\varphi)\sin(\varphi) + \frac{V^2}{r}\cos^2(\varphi) + a\sin(\varphi)\cos(\varphi) + g\cos(\varphi) \end{pmatrix}$$

Values in the maximum range of a, V and φ are inserted into this formula to generate training data-sets. These data sets can then be used to train an artificial neural network such as a multi layer perceptron or a self organising map (see [Gal90] and [Koh98]).

Appendix D

SuperCollider, installation and instructions

D.1 Notes on SuperCollider

For those readers who have not worked with SuperCollider previously we want to mention two basic points about the usage of the SuperCollider interpreter on Mac OS X. You can execute any single line expression by clicking anywhere in that line and pressing the 'Enter' key. A whole block of lines can be executed by marking it before pressing Enter. Note that the 'Enter' key is not the same key as 'Return'. On Macintosh standard laptop keyboards, which do not have an 'Enter' key, this command is replaced by 'function(fn)+return'. Furthermore for sound to be audible the local server has to be booted before running a program. This can be done by pressing the corresponding button in the SuperCollider interface or by executing:

```
s.boot;
```

If you require more detailed information or use a different operating system please refer to the SuperCollider website on `http://supercollider.sourceforge.net`.

D.2 Installation

As mentioned in subsection 5.1.3, the implementation of our real-time sonification system is written in the programming language SuperCollider, which can be downloaded from `http://supercollider.sourceforge.net/`. The sonification system was developed and tested on a *Mac OS X Version 10.5.4* system. To start the application from a source file or to open the graphical user interface (see subsection 5.1.4) you need to install and open *SuperCollider*. The sonification system is written as an extension of the SuperCollider class hierarchy and can be installed by copying the 'classes' folder, which can be found on the enclosed CD, to the SuperCollider's Extensions folder. You can locate the Extension folder by typing and executing one of the following lines in SuperCollider:

```
Platform.userExtensionDir;
Platform.systemExtensionDir;
```

The first line returns the folder which makes the classes available for the current user, the second line returns a folder which makes them available for all users of the operating system. The SuperCollider help file lists the following most commonly used paths:

User-specific
Mac OS X : ~Library/Application Support/SuperCollider/Extensions/
Linux: ~/share/SuperCollider/Extensions/
System-wide (all users)
Mac OS X : /Library/Application Support/SuperCollider/Extensions/
Linux: /usr/local/share/SuperCollider/Extensions/

After you have moved the 'classes' folder to the 'Extensions' folder, copy the file 'main.scd' and the folder 'sourceSounds' to the same location on your computer. The 'main.scd' file contains the source code, which your have to execute to start the sonification manually or with a graphical user interface. The 'sourceSounds' folder contains sound files, that the sonification system. Before you can start the sonification system from the 'main.scd' file as described in subsections 5.1.4 and 5.1.5, you have to open SuperCollider and compile the SuperCollider Library either from the menu *'Lang→Compile Library'* or by using the shortcut *'cmd-k'*. After compiling the SuperCollider library the system is set up (for the playback of recorded data).
For the sonification of live data, received from the magnetometer that is attached to the wheel as described in subsection 4.1.3, the bluetooth transmitter, which transmits the sensor data, has to be paired with the computer's bluetooth client (see subsection 5.1.1). Subsections 5.1.4 and 5.1.5 describe how to start the real-time sonification system including the call of the calibration routine described in 5.2.2.
If you are using an operation system other than Mac OS X Version 10.5.4 and encounter problems during the installation, please refer to the SuperCollider web site[1] and the SuperCollider help file for further information. The system was tested only on Mac OS X Version 10.5.4 and platform independence can not be guaranteed.

D.3 Instructions on the use of the graphical user interfaces

D.3.1 GUI for live data sonification

Start the graphical user interface for live data sonification by creating an object of the class `LiveSonificationPlayer` as described in subsection 5.1.5. If the sensor is active and connected to the computer via bluetooth, the system automatically connects to it, when the GUI is opened. Otherwise, it notifies the user in the output field on the lower right. In that case the user has to register the bluetooth device and press 'connect' in the upper right corner. As mentioned in subsection 5.1.1 we

[1] `http://supercollider.sourceforge.net/`

experienced that under MaC OS X the bluetooth client sometimes fails to recognise the bluetooth sender. If this occurs, the device has to be reregistered before the GUI is opened. Once the sensor is connected, it can be calibrated by pressing the calibrate button and following the instructions in the window that opens (see Figure 5.8).
After the calibration the sensor has to be attached to the wheel with its y-axis aligned to one of the bars and the wheel should be brought into the position displayed in Figure 2.2. To identify the current angle as the initial angle of the wheel (see section 2.2) the 'set zero angle' button has to be pressed. This assures that any offset of the angle due to the mounting of the sensor is subtracted (See subsection 4.3.1). After these initial steps the desired sonification type (Cartoonification, Vowel synthesis sonification, Event-based sonification or Direct-data sonification) can be chosen from the left drop down menu. Our main implementation for the respective approach is used unless a different sound setting is chosen from the right drop down menu. Before the sonification can be started, the 'load' button has to be pressed to upload the settings to the sound server. The sonification can then be stopped and started arbitrarily by pressing the 'stop' and 'start' button. Additionally the live data player offers the option to save the incoming data. To do so, the path of the output file has to be specified in the text field on the bottom of the control window. The recording is started, when the 'record' button is pressed. The data is saved when the 'stop' button is pressed or when the application is closed by pressing the 'close' button. The recording of data is discussed in more detail in subsection 5.1.4.
An additional information, which is useful for the practical use of the system is the knowledge that if there are no magnetic fields except the earth's, the visualisation of the magnetometer data, as seen in Figure 5.6, usually remains within the boundaries of the window. If the values of the magnetic field vector exceed the scope of the window, this may indicate the presence of other magnetic fields, that can influence the performance of the sonification system.

D.3.2 GUI for recorded data sonification

When an object of the class VideoAndSonificationPlayer is created to open the graphical user interface for the sonification of recorded data (see subsection 5.1.5), the path of a data and a video file are passed to it. The video and the sonification of the data file are played back simultaneously. As an alternative to passing the file paths to the object, the files can be uploaded by entering their paths into the designated text fields in the upper part of the control window and pressing the corresponding buttons. In that case the new paths are used after the 'load' button is pressed. As in the live sonification player the left drop down menu allows to choose a sonification type and the right menu gives access to alternative implementations of the approaches. Again the 'load' button has to be pressed, before starting the sonification by pressing the 'start' button. Pressing the 'load' button also opens 3 additional windows: one which visualises the wheel and its rotational angle (see Figure 5.9), one which visualises the data directly (see Figure 5.6 and one in which the video is played back.

In both players you can change the settings at any point by stopping the playback, doing the respective changes, uploading them and starting the sonification again.

Appendix E

Source code

E.1 Overview of classes

class	implements	section
MagnetometerOscSender	receives live data via bluetooth, splits it into 3-D vectors (x, y, z) and sends them as Osc messages	5.2.1
DataFromFileOscSender	reads recorded data from a file and sends it via OSC messages	5.2.3
Calibrator	collects the data which is necessary for a calibration	5.2.2
Differentiator	implements a numerical differentiation to find the (angular) velocity	5.3.2
MagnetometerWheelModel	calculates the current angle of the wheel using the magnetometer data; also computes the angles of all bars	5.3.1
NullMinMaxfinder	finds minima, maxima and changes of sign in the input data these indicate changes in the rolling direction, lowest and highest points of the wheel or one bar	5.3.3
Sonification	super-class of all sonification classes, creates all feature extraction objects and methods for the recording of data; also sets a basic structure for all sonifications	5.4
SonCartoonification	implements the Cartoonification	5.4.2
SonDirectData	implements the Direct-data sonification	5.4.1
SonEventBased	implements the Event-based sonification	5.4.4
SonVowel	implements the Vowel synthesis sonification	5.4.3
MagnetometerVisualizer	visualises the magnetometer data in a window	enclosed CD
LiveSonificationPlayer	graphical user interface to control and record a live sonification	5.1.5
VideoAndSonificationPlayer	graphical user interface to play recorded data together with a corresponding video	5.1.5

E.2 MagnetometerOscSender

As discussed in subsection 5.2.1, the class MagnetometerOscSender receives the magnetometer data as a continuous ascii data stream via bluetooth, splits it into 3-D magnetic field vectors and sends them as an OSC message for further processing.
The input data stream is accessed using the SuperCollider class `SerialPort`, which receives the name of the serial port device and its baudrate as Arguments:

```
(port = SerialPort(
                "/dev/tty.FireFly-713B-SPP-1",
                baudrate: 115200,
                crtscts: true););
```

If we depict the input data stream as letters rather then ascii values an extract of it could look like this :
`X123Y-300Z335`
`X123Y-300Z337`
`X123Y-305Z337`
`X125Y-308Z338 ...`
The letters X Y and Z (ascii 88, 89 and 90) indicate which axis the following value corresponds to. A line break (ascii 10 ascii 13) indicates the start of a new data item. The routine below uses this knowledge to split the input data into three dimensional magnetic field vectors. It writes the strings representing the x-, y- and z-value into the first second and third index of the array `msg`. when the ascii value of an 'X' arrives (line 7), the routine starts writing into the first index (`i=1`). It keeps concatenating the arriving characters (line 19 to 29) to the string at `msg[1]` until a 'Y' arrives, in which case it starts writing at the second index of the array (line 8) etc. When a line break indicates that a new data-item starts (line 11 and 12), the content of the array is checked for completeness (line 14 and 15) and if the data is complete, it is converted to integers and sent as an OSC message (line 16):

```
 //reads incoming ascii string and splits it into [x,y,z] data items
0 check_serial = Routine.new({
1   var ser;                                    // the current ascii value
2   i=0;                                        //i=the index of the array that is
3   loop{                                       //currently been written into
4    ser = port.read;
5
6     switch (ser)
7       {88} { i=1; msg[1]=""; }                // x value is written in msg[1]
8       {89} { i=2; msg[2]="";}                 // y value is written in msg[2]
9       {90} { i=3; msg[3]="";  }               // z value is written in msg[3]
10      {32} { }                                //space
11      {13} { }                                //line break
12      {10} { i=0;                             //line break-> send data item
13          //because of incomplete data, check if worth sending
14          bool=(not(msg[1]==="")&&{not(msg[2] ==="")})&&{not(msg[3]== "")};
15          if( bool,{
16             net.sendMsg(name, dtime, msg[1].asInteger, msg[2].asInteger, msg[3].asInteger);
17          });
18          msg=["","","",""]; }                 //reset
19      {45} {msg[i]=msg[i]++ser.asAscii;}       //following: all digits 0-9
20      {48} {msg[i]=msg[i]++ser.asAscii;}
21      {49} {msg[i]=msg[i]++ser.asAscii;}
```

```
22      {50} {msg[i]=msg[i]++ser.asAscii;}
23      {51} {msg[i]=msg[i]++ser.asAscii;}
24      {52} {msg[i]=msg[i]++ser.asAscii;}
25      {53} {msg[i]=msg[i]++ser.asAscii;}
26      {54} {msg[i]=msg[i]++ser.asAscii;}
27      {55} {msg[i]=msg[i]++ser.asAscii;}
28      {56} {msg[i]=msg[i]++ser.asAscii;}
29      {57} {msg[i]=msg[i]++ser.asAscii;};
30  }
31 });
```

E.3 Calibrator

For the collection of the calibration data (see subsection 5.2.2), the average of 10 incoming data values on the observed axis is computed. The following routine is called twice for each axis to collect the calibration data: once with the axis aligned to the 'calibration line' and once with the axis antipodal to the 'calibration line'.

```
respond = {|t, r, msg|                                //called for incoming OSC messages(msg)
   if(n<10,{                                          //n= number of data items that arrived
      dataAccum=dataAccum+msg[inputIndex];            //sum up 10 values
      n=n+1;
   },{
      calibrationData[arrayIndex]= (dataAccum/10).asInteger;//divide by 10 write into array
      (dataAccum/steps).asInteger.postln;
      miniResponder.remove;                           //stop receiving messages for this axis
      n=0;                                            //reset values for next axis
      dataAccum=0;
       });
};
```

You can find the full source code of the class `Calibrator` in the file Calibrator.sc on the provided CD.

E.4 NullMinMaxFinder

The class `NullMinMaxFinder` finds all minima, maxima and changes of sign in a data stream. The source code below shows the method which is called for each incoming data value. The algorithm and its implementation are explained in subsection 4.4.3 and 5.3.3. The complete class can be found on the enclosed CD.

```
findMinMax={|t,r,msg|
value=msg[2];

case

//CHANGES OF SIGN
{(oldValue.sign*value.sign).isPositive.not}
{
   if(kNull>minDistance,{              //unless last one too close
      net.sendMsg(oscOut, msg[1], 0, value); //report change of sign
      oldValue=value;
      kBigger=kBigger+1;               //counters to see how far maxima apart,
```

```
      kSmaller=kSmaller+1;             // how far minima apart
      kNull=0;                         // how far changes of sign apart
      });


   if(kNull==0,{                       //test if 2 successive changes of sign
                                       //to avoid overseeing min or max
      case

      { value.abs <pihalf}             //closer to 0 than pi,lowest point
                                       //no jump, normal min/max condition
         {if(value.isPositive,{        //last value pos.(e.g. 0.1,-0.1,0.1)minimum
            if(kSmaller>minDistance,{net.sendMsg(oscOut, msg[1], 1, value);});
            valueIsHigher=true;
            oldValue=value;
            kSmaller=0;
            kBigger=0;
         },{                           //last value negative->maximum
            if(kBigger>minDistance,{net.sendMsg(oscOut, msg[1], 2, value);});
            valueIsHigher=false;
            oldValue=value;
            kBigger=0;
            kSmaller=0;
         });
      }

      { value>piHalf  }                //value closer to pi>highest point
                                       //>inverted min/max conditions

         {if(kBigger>minDistance,{net.sendMsg(oscOut, msg[1], 2, value);});
         valueIsHigher=false;
         oldValue=value;
         kBigger=0;
         kSmaller=0;
               }

      {value<(-piHalf) }               //current value closer to -pi,minimum
         {if(kSmaller>minDistance,{net.sendMsg(oscOut, msg[1], 1, value);});
         valueIsHigher=true;
         oldValue=value;
         kSmaller=0;
         kBigger=0;
         };
   });
}

//MAXIMUM
{valueIsHigher}                        //last value was higher than the one before

{  if(value<oldValue,{                 //current value lower than last value too
      if(kBigger>minDistance,{         //if distance to last max large enough
         net.sendMsg(oscOut, msg[1], 2, value); //->report maximum
      });
      valueIsHigher=false;
      oldValue=value;
      kBigger=0;
      kSmaller=0;
      kNull=kNull+1;

   },{                                 //otherwise no extremum
      oldValue=value;
      kBigger=kBigger+1;
      kNull=kNull+1;
   })
```

```
}

//MINIMUM
{true}                                  //last value was lower than the one before

{  if((value> oldValue),{               //current value higher than last value too

      if(kSmaller>minDistance,{         //if distance to last minimum big enough
         net.sendMsg(oscOut, msg[1], 1, value); //->report minimum
      });
      valueIsHigher=true;
      oldValue=value;
      kSmaller=0;
      kBigger=0;
      kNull=kNull+1;
   },{                                  //no change of sign or extremum found
      oldValue=value;                   //just update values
      kSmaller=kSmaller+1;
      kNull=kNull+1;
   });
};
};
```

Appendix F

Experiments

F.1 Project description

Dear participant

Welcome to the German wheel sonification experiments. The series of experiments, which you are about to take part in is conducted within the framework of my 'Diplomarbeit', which I write at Bielefeld University in collaboration with the Queen Mary, University of London. During the course of this project we designed a 'closed loop audio feedback system' for German wheel movements. In simple words this means: We measure the motion of the German wheel while someone is performing moves on it. This motion is then represented via sound so that the performer can hear the movement of the wheel as well as feel and see it. In other words: Moving the wheel produces sounds. The experiments which we are conducting analyse how such additional auditory feedback effects the performance of the participants. We test two different sonification approaches, each expressing the motion of the wheel with different sounds.

If you agree to take part in the experiments, we would kindly ask you to complete the questionnaire and consent form below. The experiments will then be structured as follows:

You will carry out the same exercise 3 times : once without any sound and once with each sonification. The exercise will be explained to you in detail before the experiment starts. At the beginning of each of the 3 sets you will get a short time to practice whatever you wish to get used to the respective sound.

If you need some additional time to warm up or prepare yourself before the start of the experiment, please feel free to ask for it. You are also welcome to ask at any time, if something is unclear or you would like some additional information concerning the project, its context or the experiments.

F.2 Pre-questionnaire

In the pre-questionnaire the participants were asked to provide the following information:

- Name
- Surname
- Age
- Gender
- How long have you been learning German wheel for?
- Had you heard about sonification before this experiment?
- If yes, in which context?
- Do you play an instrument or do you have any kind of musical education?
- If yes describe your musical experience (e.g I have learned to play the piano for 1 year).

F.3 Post-questionnaire

The participants were asked to give the following information about the experiment in general after taking part in it:

Please read the following statements and mark them by giving them a number between 1 and 10 (1= I do not agree at all 10= I completely agree)

- The sonification had a positive effect on my German wheel moves.
- I could imagine to use sonification for my training.
- I found the task hard.
- My concentration decreased during the experiment.

In the same way they marked the following statements for each of the sonification approaches:

- This sonification sounds aesthetic.
- This sonification had a positive effect on my German wheel moves.
- I could imagine to use this sonification for my training.
- My concentration decreased during the experiment with this sonification.

Additionally every participant described each sonification in her/his own words.

F.4 Statistics

F.4.1 Means, medians and standard deviations of the swing heights

The following three tables contain the means, medians and standard deviations of the swing height of each expert and novice.

Table F.1: Means of the swing height

	without	Event-Based	Vowel
expert 1	1.260	1.145	1.176
expert 2	0.803	0.651	0.827
expert 3	1.038	0.997	1.032
expert 4	0.375	0.479	0.367
novice 1	0.289	0.307	0.181
novice 2	0.637	0.744	0.600
novice 3	0.637	0.621	0.595
novice 4	0.338	0.305	0.350
novice 5	0.485	0.548	0.566
novice 6	0.452	0.443	0.486
novice 7	0.284	0.299	0.369

Table F.1: Standard deviation of the swing height

	without	Event-Based	Vowel
expert 1	0.137	0.065	0.098
expert 2	0.128	0.056	0.047
expert 3	0.043	0.031	0.037
expert 4	0.110	0.069	0.160
novice 1	0.062	0.024	0.046
novice 2	0.057	0.072	0.080
novice 3	0.074	0.030	0.041
novice 4	0.036	0.028	0.040
novice 5	0.166	0.226	0.287
novice 6	0.040	0.033	0.060
novice 7	0.031	0.027	0.027

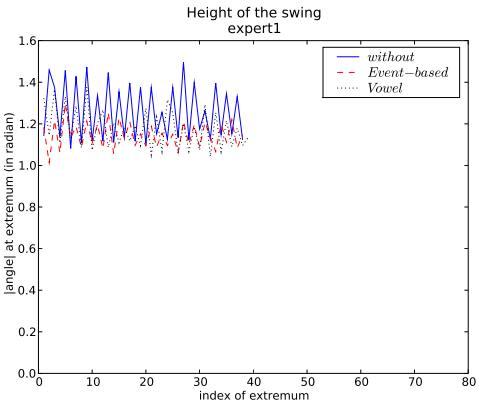
Table F.1: Median of the swing height

	without	Event-Based	Vowel
expert 1	1.255	1.136	1.182
expert 2	0.817	0.638	0.830
expert 3	1.029	0.999	1.0366
expert 4	0.435	0.478	0.304
novice 1	0.297	0.308	0.175
novice 2	0.643	0.760	0.601
novice 3	0.638	0.622	0.591
novice 4	0.331	0.308	0.346
novice 5	0.518	0.530	0.353
novice 6	0.449	0.440	0.485
novice 7	0.285	0.296	0.367

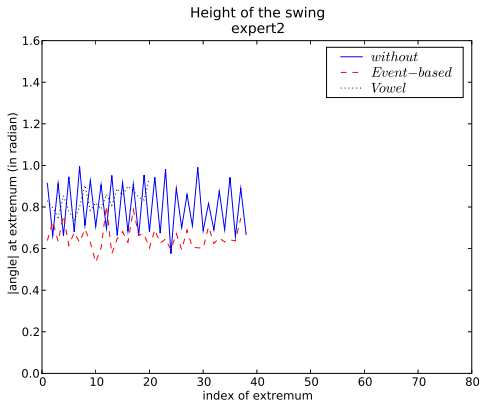
F.4.2 Plots of the swing heights, experts

The following plots visualise the swing height of each expert under the three different conditions, namely without sonification, with Event-based sonification and with Vowel synthesis sonification. The x-value is the index of the change of rolling direction and the y-value is the absolute value of the maximum angle that was reached when the wheel changed its rolling direction. In the data of expert 1 one outlier caused by an error in measurement was corrected by resetting φ for the nineteenth maximum of the Vowel synthesis run. The original value was 0.016564056277275085 and it was set to 1.0669071078300476, the average value of its neighbouring maxima.

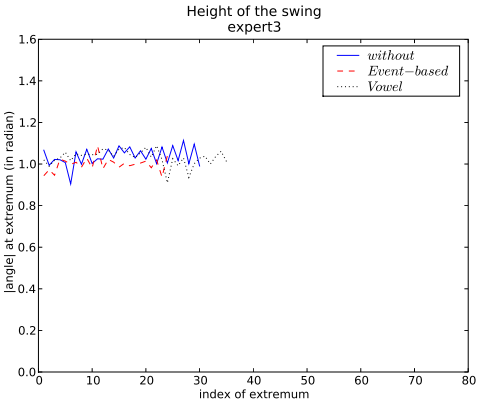
Figure F.1: The experts' height of the swing



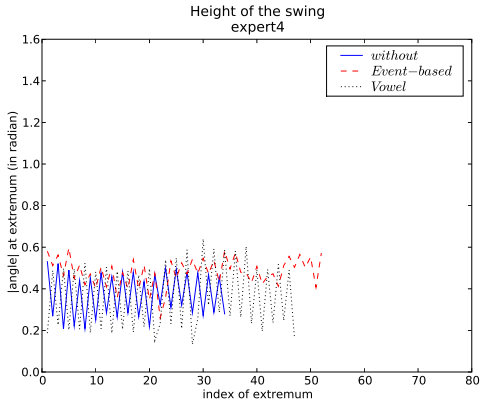
(a) expert 1, one outlier corrected



(b) expert 2



(c) expert 3

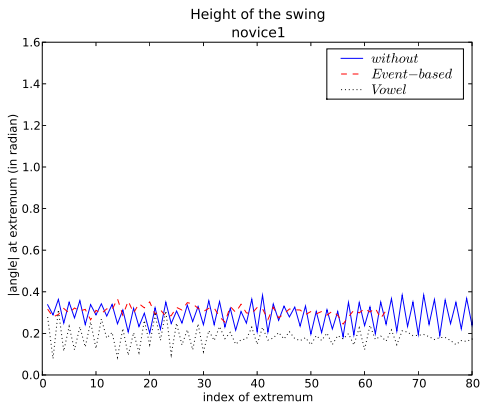


(d) expert 4

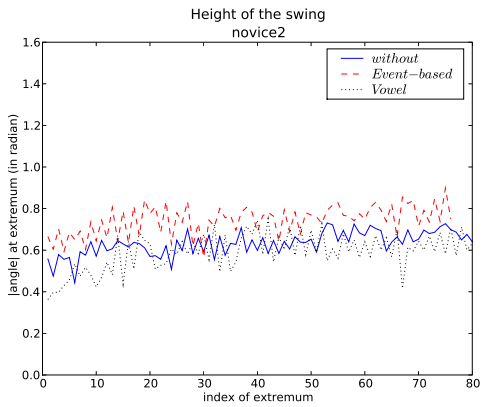
F.4.3 Plots of the swing heights, novices

The following plots visualise the swing height of each novice under the three different conditions, namely without sonification, with Event-based sonification and with Vowel synthesis sonification. The x-value is the index of the change of rolling direction and the y-value is the absolute value of the maximum angle that was reached when the wheel changed its rolling direction. The data of novice 2, which was collected during the run conducted without sonification was corrected by removing the 40th maximum, which was caused by an error in measurement. Likewise the 38th maximum of the run with the Vowel synthesis sonification was removed in the data of novice 5.

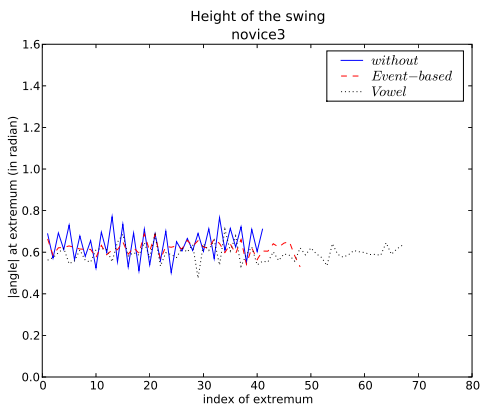
Figure F.1: The novices' height of the swing



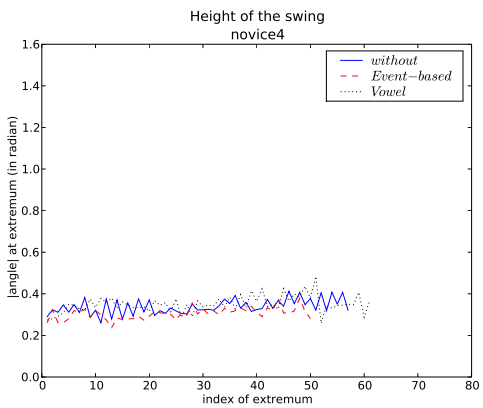
(a) novice 1



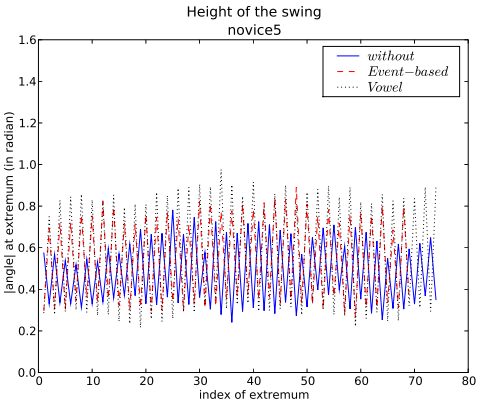
(b) novice 2, one outlier corrected



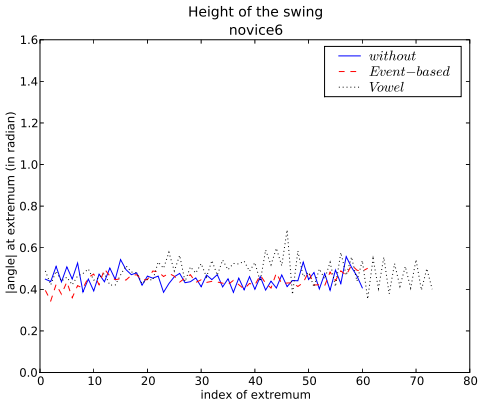
(c) novice 3



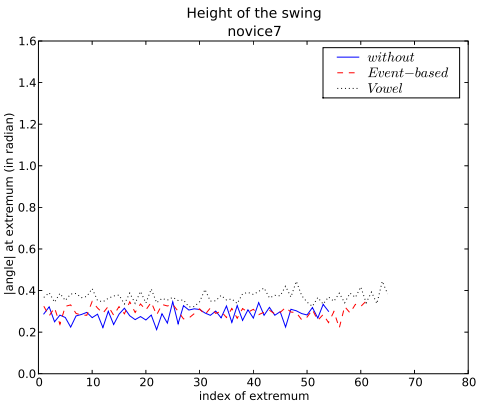
(d) novice 4



(e) novice 5, one outlier corrected



(f) novice 6



(g) novice 7

F.4.4 Means, medians and standard deviations of $\Delta\varphi_{eval}$

The following three tables contain the means, medians and standard deviations of $\Delta\varphi$ of each expert and novice.

Table F.2: Means of $\Delta\varphi$

	without	Event-Based	Vowel
expert 1	0.256	0.112	0.168
expert 2	0.247	0.077	0.056
expert 3	0.060	0.036	0.035
expert 4	0.209	0.090	0.304
novice 1	0.111	0.028	0.0649
novice 2	0.054	0.087	0.082
novice 3	0.134	0.038	0.049
novice 4	0.046	0.026	0.0425
novice 5	0.316	0.445	0.569
novice 6	0.053	0.028	0.080
novice 7	0.043	0.033	0.033

Table F.2: Median of $\Delta\varphi$

	without	Event-Based	Vowel
expert 1	0.260	0.103	0.178
expert 2	0.252	0.067	0.046
expert 3	0.059	0.025	0.030
expert 4	0.197	0.080	0.297
novice 1	0.109	0.022	0.054
novice 2	0.050	0.086	0.068
novice 3	0.132	0.036	0.036
novice 4	0.043	0.021	0.040
novice 5	0.317	0.446	0.572
novice 6	0.057	0.021	0.072
novice 7	0.038	0.032	0.032

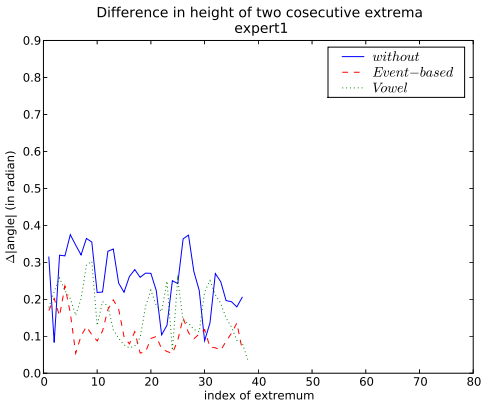
Table F.2: Standard deviation of $\Delta\varphi$:

	without	Event-Based	Vowel
expert 1	0.080	0.046	0.065
expert 2	0.056	0.054	0.030
expert 3	0.034	0.031	0.028
expert 4	0.038	0.042	0.039
novice 1	0.038	0.019	0.054
novice 2	0.040	0.055	0.057
novice 3	0.045	0.031	0.044
novice 4	0.028	0.019	0.038
novice 5	0.076	0.043	0.040
novice 6	0.028	0.022	0.058
novice 7	0.028	0.021	0.023

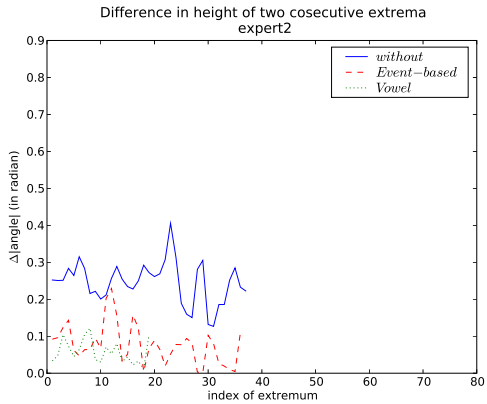
F.4.5 Plots of $\Delta\varphi_{eval}$, experts

The following plots visualise $\Delta\varphi$ for each expert under the three different conditions, namely without sonification, with Event-based sonification and with Vowel synthesis sonification. The x-value is the index of the change of rolling direction and the y-value is $\Delta\varphi_{eval}$, which is the difference between the (absolute) maximum angle reached in the current and the previous change of rolling direction. In the data of expert 1 one outlier caused by an error in measurement was corrected by resetting φ for the nineteenth maximum of the Vowel synthesis run. The original value was 0.016564056277275085 and it was set to 1.0669071078300476, the average value of its neighbouring maxima.

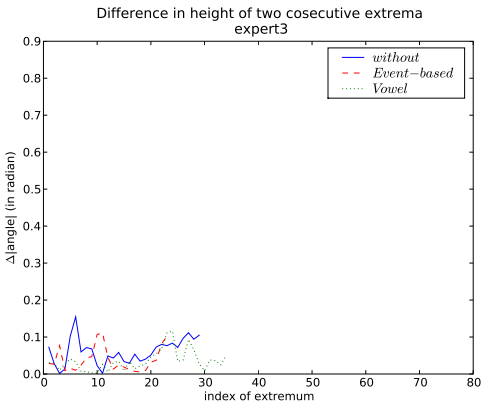
Figure F.-1: $\Delta\varphi_{eval}$, experts



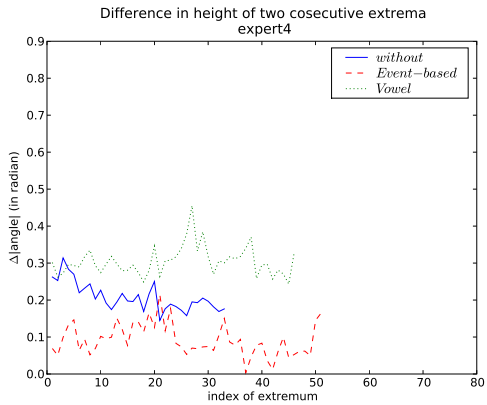
(a) expert 1, one outlier corrected



(b) expert 2



(c) expert 3

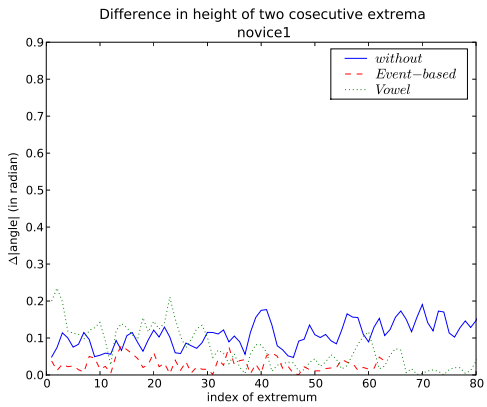


(d) expert 4

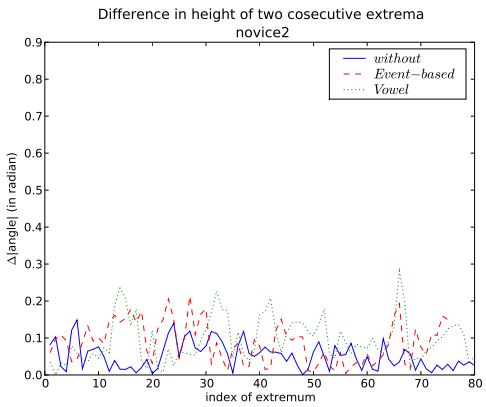
F.4.6 Plots of $\Delta\varphi_{eval}$, novices

The following plots visualise $\Delta\varphi$ for each novice under the three different conditions, namely without sonification, with Event-based sonification and with Vowel synthesis sonification. The x-value is the index of the change of rolling direction and the y-value is $\Delta\varphi$, which is the difference between the (absolute) maximum angle reached in the current and the previous change of rolling direction. The data of novice 2, which was collected during the run conducted without sonification was corrected by removing the 40th maximum,which was caused by an error in measurement. Likewise the 38th maximum of the run with the Vowel synthesis sonification was removed in the data of novice 5.

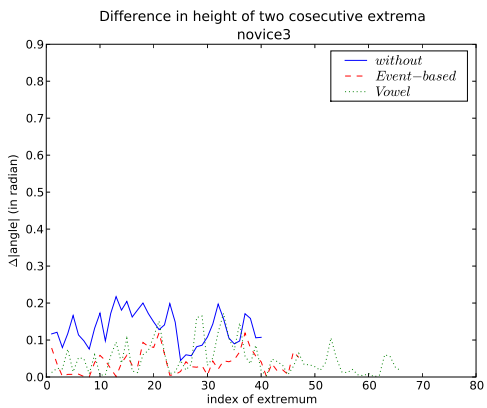
Figure F.-1: $\Delta\varphi_{eval}$, novices



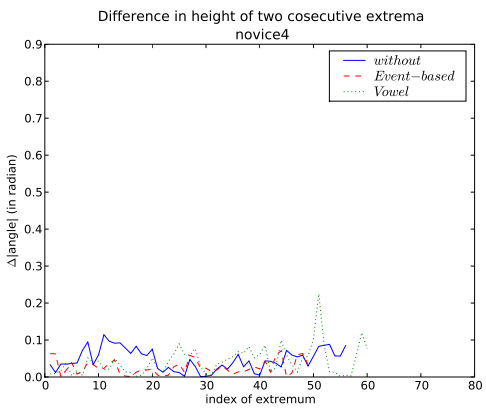
(a) novice 1



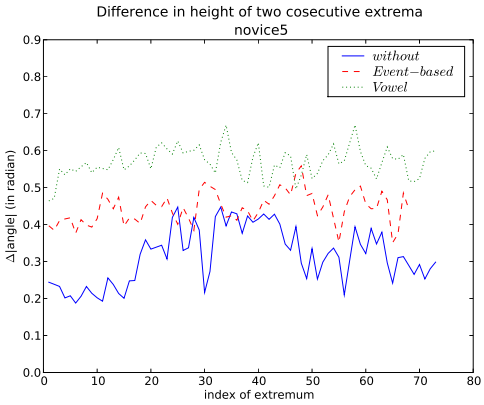
(b) novice 2, one outlier corrected



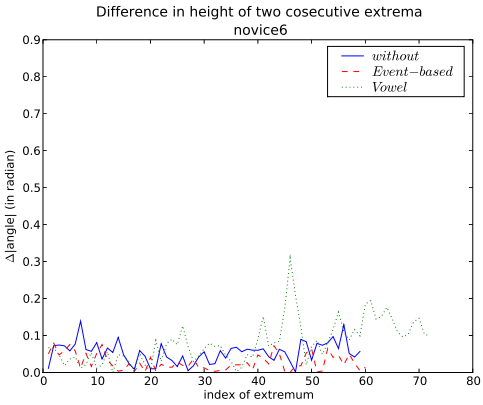
(c) novice 3



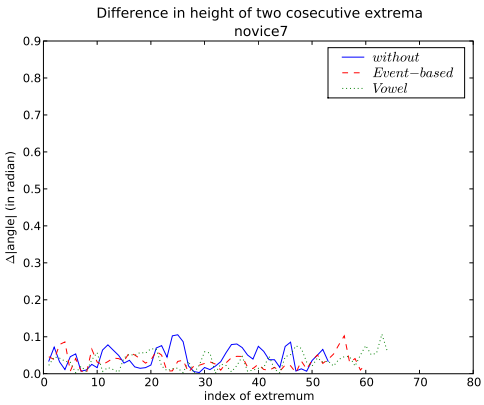
(d) novice 4



(e) novice 5, one outlier corrected



(f) novice 6



(g) novice 7

Bibliography

[Bar98] S. Barrass. Auditory Information Design. 1998.

[BDU+99] ER Bachmann, I. Duman, UY Usta, RB McGhee, XP Yun, and MJ Zyda. Orientation tracking for humans and robots using inertial sensors. In *International Symposium on Computational Intelligence in Robotics & Automation (CIRA 99)*, pages 187–194, 1999.

[BGdC+07] T. Bovermann, J. Groten, A. de Campo, G. Eckel, and G. Weiz. JUGGLING SOUNDS. *In Proceedings of the 2nd International Workshop on Interactive Sonification, York, Uk, February 3rd 2007*, 2007.

[Bis95] C.M. Bishop. *Neural networks for pattern recognition.* Oxford University Press, USA, 1995.

[BSG89] M.M. Blattner, D.A. Sumikawa, and R.M. Greenberg. Earcons and icons: Their structure and common design principles. *Human-Computer Interaction*, 4(1), 1989.

[CBL+04] R.J. Cassidy, J. Berger, K. Lee, M. Maggioni, and R.R. Coifman. Auditory display of hyperspectral colon tissue images using vocal synthesis models. In *Proceedings of the 2004 International Conference on Auditory Display*, 2004.

[CHX+06] Y. Chen, H. Huang, W. Xu, R.I. Wallis, H. Sundaram, T. Rikakis, T. Ingalls, L. Olson, and J. He. The design of a real-time, multimodal biofeedback system for stroke patient rehabilitation. *Proceedings of the 14th annual ACM international conference on Multimedia*, pages 763–772, 2006.

[Coo02] P.R. Cook. *Real Sound Synthesis for Interactive Applications.* AK Peters, Ltd., 2002.

[CW92] S. Context Williams. Perceptual principles in sound grouping, 1992.

[dC07] A. de Campo. Toward a data sonification design space map. *Proceedings of the International Conference on Auditory Display (ICAD2007)*, pages 342–347, 2007.

[Eff05] A.O. Effenberg. Movement Sonification: Effects on Perception and Action. *IEEE MULTIMEDIA*, pages 53–59, 2005.

[FCB05] J. Fox, J. Carlile, and J. Berger. SONIMIME: SONIFICATION OF FINE MOTOR SKILLS. 2005.

[Gal90] SI Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191, 1990.

[Gav92] W. W. Gaver. *Using and creating auditory icons.* SFI studies in the sciences of complexity. Addison Wesley Longman, 1992.

[HBSR08] Thomas Hermann, Gerold Baier, Ulrich Stephani, and Helge Ritter. Kernel regression mapping for vocal eeg sonification. Paris, France, 2008. inproceedings.

[Her09] Thomas Hermann. Sonifikation hochdimensionaler Daten - Funktionaler Klang zum Erkenntnisgewinn. In Georg Spehr, editor, *Funktionale Klänge*, number 2 in Sound Studies, pages 67–85. transcript Verlag, Bielefeld, 05 2009.

[HHG04] T. Hermann, O. Honer, and C. Grunow. Sonification of Group Behavior for Analysis and Training of Sports Tactics. *Proceedings of the international workshop on sonifiction, Bielefeld*, 2004.

[HHR05] T. Hermann, O. Honer, and H. Ritter. Acoumotion-an interactive sonification system for acoustic motion control. *Gesture In Human-Computer Interaction and Simulation–6th International Gesture Workshop,(GW'2005), Berder Island, France*, 2005.

[Kla80] D.H. Klatt. Software for a cascade/parallel formant synthesizer. *Journal of the Acoustical Society of America*, 67(3):971–995, 1980.

[Koh98] T. Kohonen. The self-organizing map. *Neurocomputing*, 21(1-3):1–6, 1998.

[Kra94] G. Kramer. Auditory display: sonification, audification, and auditory interfaces. Santa Fe Institute (Santa Fe NM), 1994.

[KRE88] B.W. Kernighan, D.M. Ritchie, and P. Ejeklint. *The C programming language.* Prentice-Hall Englewood Cliffs, NJ, 1988.

[KTVB+05] A. Kapur, G. Tzanetakis, N. Virji-Babul, G. Wang, and P.R. Cook. A FRAMEWORK FOR SONIFICATION OF VICON MOTION CAPTURE DATA. *International Conference on Digital Audio Effects*, 2005.

[KW06] J.B.M. Kleimann-Weiner. The sound of one arm swinging: A model for multidimensional auditory display of physical motion. *Proceedings of the 12th International Conference on Auditory Display. ICAD*, pages 278–280, 2006.

[Lee07] J.C. Lee. Head Tracking for Desktop VR Displays using the Wii Remote. *Published online at http://www. cs. cmu. edu/johnny/projects/wii*, 2007.

[LL06] M. Liljedahl and S. Lindberg. DIGIWALL–AN AUDIO MOSTLY GAME. *Proceedings of the 12th International Conference on Auditory Display, June 2006*, 2006.

[Loe92] H.T. Loeser. *Sonar engineering handbook.* Peninsula Publ., 1992.

[Lou06] Peter William Lount. smalltalk dot org. http://www.smalltalk.org/main/, 2006.

[PB52] G.E. Peterson and H.L. Barney. Control methods used in a study of the vowels. *The Journal of the Acoustical Society of America*, 24:175, 1952.

[PFTV92] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. Numerical recipes in C: the art of scientific programming. *Cambridge U. Press, Cambridge, England*, 1992.

[PV92] W.H. Press and W.T. Vetterling. Numerical recipes in C: the art of scientific computing. 1992.

[R+01] P. Ripka et al. *Magnetic sensors and magnetometers.* Artech house, 2001.

[Sim61] H. Simon. Instrumentenflugkunde und Navigation, 1961.

[Sny00] B. Snyder. *Music and Memory.* MIT Press, 2000.

[Tak93] Takeuchi, T. Auditory information in playing tennis. *Percept Mot Skills*, 76(3 Pt 2):1323–8, 1993.

[Wil05] J.S. Wilson. *Sensor technology handbook.* Newnes, 2005.

Declaration

Herewith I affirm that I have written this Diplomarbeit independently and that I have used no other sources or resources than the ones named.

London, May 14, 2009 .....

Jessica Hummel