RWTH AACHEN UNIVERSITY

DanSense

Rhythmic Analysis of Dance Movements using Acceleration-Onset Times

Diploma Thesis at the
Media Computing Group,
Prof. Dr. Jan Borchers,
Dept. of Computer Science,
RWTH Aachen University

by
Urs Enke

Thesis adviser:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Karl-Friedrich Kraiss

Registration date: 1 March 2006
Submission date: 1 September 2006

Contents

List of Figures

Abstract (English)

Numerous projects around the world have already dealt with the idea to turn dance into music. The advantage of a system doing this over conventional musical instruments would be the intuitiveness with which even laymen move rhythmically. In comparison, the learning of an instrument is usually an elaborate endeavour. However, past such approaches were more or less rule-based, forcing their users to, again, learn how to master the devices.

The *Sensor Music Project*, as part of which this thesis was initiated, also aims to build a device to turn motion into sound, but based on the latter's inherent rhythm. Envisaging various ways of using this to fittingly find, influence or even generate music, it is hoped that the moving person will appreciate the intuitive control and not perceive the system as something to be learned.

Given a prototype of an acceleration sensor package to use as motion input, this thesis describes a way of turning such sensor data into a representation of rhythm suitable for said further processing.

The questions on how to represent rhythm and which methods to use for its derivation first remain answered ambiguously, studying diverse works in the field of the rhythmic analysis of music. Finally, a pipeline of algorithms considered sensible is built and implemented in a software package called *DanSense*.

Having been intended as one step among several in the attempt to derive music from dance, DanSense turns out to itself require, but also invite to, further development until it can satisfy the probable demands in detection stability and accuracy of the (not yet existing) application that is to use its output.

Abstract (Deutsch)

Zahlreiche Projekte haben sich bereits an der Umwandlung von Tanzbewegungen in Musik befasst. Der Vorteil eines solchen Systems gegenüber konventionellen Musikinstrumenten wäre die Intuitivität, mit der sich selbst Laien rhythmisch bewegen. Im Vergleich dazu ist das Erlernen eines Instruments üblicherweise ein aufwendiges Unterfangen. Diesbezügliche Ansätze jedoch waren in der Vergangenheit mehr oder weniger regelbasiert, zwangen ihre Benutzer also wiederum dazu, das Gerät erst einmal beherrschen zu lernen.

Das *Sensor Music Project*, als Teil dessen diese Diplomarbeit initiiert wurde, zielt ebenfalls auf die Erstellung eines Gerätes ab, das Bewegungen in Töne verwandelt, allerdings basierend auf deren inhärentem Rhythmus. Mit Blick auf verschiedene mögliche Methoden, diesen Rhythmus zum passenden Finden, Beeinflussen oder gar Erzeugen von Musik zu nutzen, besteht die Hoffnung, die sich bewegende Person möge die Intuitivität eines solchen Systems wertschätzen und seine Automatismen nicht als zu erlernen betrachten.

Unter Vorgabe des Prototyps eines Beschleunigungssensoren-Pakets beschreibt diese Arbeit einen Weg, aus Daten solcher Sensoren eine für die genannte Weiterverarbeitung brauchbare Darstellung ihres Rhythmus zu extrahieren.

Antworten auf die Fragen nach einer geeigneten solchen Darstellung und nach den besten Methoden für die genannte Extraktion bleiben zunächst nur vage beantwortet, während vielfältige Arbeiten aus dem Bereich der rhythmischen Analyse von Musik betrachtet werden. Schließlich wird eine als sinnvoll erachtete Kette von Algorithmen erstellt und in einem Programmpaket namens *DanSense* implementiert.

Selbst nur als ein Teilschritt auf dem Weg zu einer Umwandlung von Tanz in Musik gedacht, stellt sich DanSense in seiner gegenwärtigen Fassung als verbesserungsbedürftiger, aber auch als verbesserungsfähiger Ansatz heraus, was die angenommenen Bedürfnisse der noch inexistenten Anwendungen angeht, welche einmal die rhythmische Repräsentation weiterverarbeiten sollen.

Acknowledgements

Working on this thesis was made especially enjoyable by the friendly atmosphere at the Media Computing Group, thanks to Prof. Dr. Jan Borchers probably the faculty's most egalitarian chair. Although exams are probably a bit less attractive than hands-on project work, I regret having previously been oblivious to the group and its interesting field of activity.

Speaking of the field of activity, I imagined it hard to find a secondary examiner, but Prof. Dr. Karl-Friedrich Kraiss' quick acceptance of my request made the search a short one.

Concerning the work itself, the most valuable input came from my supervisor, Eric Lee, regarding procedure, algorithmic content and implementation, as well as such aimed at my pretentious language, especially elaborate hypotaxes, of which this sentence is to be a lucid example, and whose tiring nature the already scared reader can hopefully be sure never to find surpassed in the forthcoming chapters.

Further thanks go to Leo de Jong, without whom this very special thesis topic would not have existed, and Jeroen Groot, who often helped me with questions regarding Max/MSP.

Martin Neuhäußer, Alena Sudholt, Lorenz Merdian, Niels Gürtler and Thomas Rath actively helped me to some recordings of 'real' dance movements that I could hardly have produced myself and even less liked to. Michael Herchenbach, Georgios Dilgerakis and my brother Lutz all took the pains of looking for errors or other deficiencies in the ensuing document. Thanks to all for being my second pair of feet and eyes!

My parents did not feel qualified to contribute corrections. One could say that basically, they contributed everything that was correct in the first place.[1]

[1] This is not meant as a violation of the DPO!

Conventions

Language

This thesis is written in contemporary British English concerning both orthography and grammatical style.[2] This includes the choice to avoid the serial comma.

Single quotation marks are used as ‘scare quotes’, double ones remain reserved for actual quotations.

Highlighting

Source code, command lines and internet links are typeset in `fixed-width` font. For brevity, the latter omit any leading `http://`. In the PDF version, however, the complete address is linked.

Terms or program names being introduced are typeset in *italics*. Names of program classes and methods appear without serifs; constants as well, but in capital letters and in a smaller font. Programming languages and frameworks are not highlighted.

Definitions of technical terms are set off in coloured boxes:

Definition:
Term

TERM:
A word with a precise meaning in a specific context.

They can also be looked up in the glossary at the back.

Figures depicting sensor data always show graphs of the same recording and during the same time interval. Different recordings are never shown

[2] Note deviations from what one might expect: for example, the Oxford English Dictionary gives *program* as the preferred spelling in a computing context and uses *-meter* for measurement devices.

in one figure; where the need for a comparison arises, this takes place across figures.

Hyperlinked Version

This document's digital version (PDF, 5 MB), together with source codes and other related material, is downloadable at
`media.informatik.rwth-aachen.de/enke.html`,
and features clickable

- tables of contents (each chapter has its own)
- references to bibliographic entries,
- internal references to sections and figures and
- internet links.

The latter are especially helpful in the case of the bibliography, as most entries there do have a link.

1 Introduction

1.1 Background

For thousands of years, people have been moving to rhythms, be it as part of a ritual, to improve morale when working together or just as the casual tapping of fingers on an office desk. Formalized dances may be hard to learn, but the latent urge to move to auditory stimuli appears rather instinctive. Creating audible rhythms, which usually is chosen to induce rhythmical movement, is similarly easy, but requires the creator's concentration on producing these sounds.

Creating music by dancing?

Even more so, the playing of melodic music requires an instrument to be learnt and played more than casually. In both cases, the ability to simultaneously dance is hampered by the devices used for creating rhythm or melody. It appears that a device that could turn intuitively performed movements into music could be quite popular and, as with all technological gadgets, become increasingly affordable.

The Sensor Music Project

Initiated by Prof. em. Leo de Jong, the *Sensor Music Project* [dJ04] aims to develop such a device, capturing motion using unintrusive wireless acceleration sensors and processing the taken measurements for rhythmic structures, as well as for hints on which sounds could be fittingly supplemented. Intended uses for this way of deriving music are largely leisure-related, ranging from motivating acoustic feedback during aerobics sessions to an 'automatic DJ' picking music with a rhythm fitting that of the dance, but also include therapeutic ones with autistic children in mind.

The first example should be the least demanding task, if simple sounds are to be played for each high-acceleration movement like a kick with one's foot. The second involves an analysis of several movements in a row, detecting structures that would then allow to find a match in a database of known rhythms.

Example number three raises the question how a melody should be derived from a dance performance, even provided that a rhythm could be properly detected: de Jong's central idea in this context is that both rhythmic patterns and harmonic sounds are characterized by their composition of frequencies. He suggests converting one into the other by playing the rhythm at a faster pace. In addition, a database of rhythms could be set up like for example three, that would be used to choose tonality and instrumentation according to which cultural background is known to belong to the given rhythmic pattern.

Extraction of rhythm more feasible than of melody

Still, the resulting 'melody' would hardly deserve its name if a constant rhythm led to a constant harmonic interval being played. The fact that dance hardly reproducibly captures melodic aspects of music thus makes the question of how to use movements to synthesize a fitting *sequence* of harmonics seem rather artistic. The extraction of rhythm, in contrast, appears more algorithmically feasible in so far as the model solution could be defined rather uncontroversially.

This Thesis' Role

However successful an inference of melody might be, the derivation of rhythmic information from movements is a prerequisite. This thesis deals with this very derivation, using the Sensor Music Project's hardware prototypes as input devices. It discusses ways of how to convert measurements of acceleration into a representation of the underlying rhythm and develops a comprehensive algorithm using the most promising methods.

A Java program, DanSense, was created to demonstrate these algorithms. Due to its modular structure, its example front-end, which graphically displays the rhythm, could be replaced by further processing as envisaged by de Jong. It employs *OpenSound Control* (OSC) for data reception, an open application-layer communications protocol making the system independent from the specifics of the sensors used for development.

DanSense derives rhythm from accelerometer data

A more precise description of the goals set and the steps involved in attaining them is given in the next section, although a clarification of some essential terms, including *rhythm* itself, is deferred to Section 2.2.

1.2 Tasks

Expanding on this thesis' title, its primary aim can be summarised as follows: a person's movements' pace and a representation of their rhythmic structure are to be derived by real-time evaluation of attached acceleration sensors' data.

Primary aim: rhythmic analysis

The method finally opted for employs, as stated in the title, "acceleration onsets", the points in time that each mark the beginning of a significant movement. Thus, the work can be divided into two steps: first, these significant movements have to be detected, then their occurrences analysed to derive a rhythm.[1]

As the resulting real-time algorithm should yield information about a performance's rhythmic patterns in a reusable representation, an additional conversion into sound like envisioned for the Sensor Music Project is just one possible use of the resulting data. The conducting of music is an exemplary other, which appears to be sensible to include in this thesis as a secondary aim: with *Personal Orchestra* [LKD+06], the advising chair (RWTH's *Media Computing Group*) has already developed (and is still in the process of improving) a successful system for conducting recorded audio with the help of optically tracked batons.

Secondary aim: conducting of music

1.2.1 Data Recording and Preliminary Analysis

The first task is to read data from the accelerometers and analyse it with respect to the features that should be extracted for automatic rhythmic analysis.

[1] In retrospect, it would have been better to talk of 'acceleration impulses' as movements' beginnings are not necessarily their most relevant point in time for analysis. Still, the additional line in the title gets across that the detection of pivotal points in time is an important aspect of the employed method.

To visualize data, an auxiliary program of the Personal Orchestra application, called *Beat Tapper*, appears suitable to determine characteristic properties of certain movements' acceleration graphs. Originally showing a video recording and a graphical representation of its audio track to facilitate the annotation of perceived musical beats' position, it can be complemented by graphs of data recorded in parallel to the video.

To be able to align the sensor recording with an audio/video track in the first place, a recording application is required that enriches sensor output with time stamps of the audio file that is being danced to.

All of these implementations will be revisited in Chapter 5, *Implementation*.

1.2.2 Rhythm Extraction

Modular implementation

Using the attained knowledge about the looks of sensor graphs, an analysis pipeline needs to be set up that analyses the most recent data in appropriate intervals for movements and their rhythmic patterns. The pipeline should be modular in the sense that the core implementation is independent from the sensors used as input.

Said intervals' "appropriateness" and the extent of the time window encompassing "recent" events is left for optimisation with respect to quick adaptation to rhythmic changes on the one hand and avoidance of too erratic output on the other. These topics will be discussed more concretely in Chapter 4.

The patterns detected finally have to be output in a useful representation. A useful representation will emerge from the discussions of rhythmical terms in Section 2.2.

Motivation for most techniques employed is derived from the works described in Chapter 3.

1.2.3 Evaluation

Algorithm development and implementation have to occur in continuous alternation with short tests of their effectiveness. For those, as well as for a final evaluation, it is required to make recordings of actual dance movements. The algorithms' application to such recordings is discussed in Chapter 6.

1.3 Structure

In the remainder of this paper,

- Chapter 2, *Setting*, takes a look at the technical conditions set by the used sensor package and clarifies some rhythm-related terms.
- Chapter 3, *Related Work*, then describes approaches taken by researchers in the past that either concerned the processing of dance movements or dealt with rhythmic analyses of audio recordings, e.g., determining their tempo. Their aims are compared with those of this thesis, useful approaches are highlighted and noted for later use.
- The intended pipeline of algorithms for rhythmic analysis is discussed in detail in Chapter 4, *Algorithmic Design*, also explaining the discardment of alternative ideas.
- These algorithms' *Implementation*, but also that of auxiliary programs or such otherwise created as part of this thesis, is the focus of Chapter 5.
- Chapter 6, *Evaluation*, discusses the developed algorithms' effectiveness.
- Chapter 7, *Summary & Future Work* sums up all previous ones and outlines possible aspects of further study.

Finally, the appendices give an introduction into frequency analysis and an overview of the prototyping environment Max/MSP.

2 Setting

2.1 Technical Environment

As mentioned before, Prof. de Jong supplied the acceleration sensor package used for this thesis. It, as well as its driver software, was created by *Radical Solutions Technology* (*RST*), a group of Romanian engineers.

2.1.1 Acceleration Sensors

Used sensors measure acceleration along two axes

As actual sensors, RST used the two-axis accelerometers i*MEMs ADXL-203* of *AnalogDevices*.[1] Although any device that is supposed to be used on a dancing person should not require cables to run to a stationary computer used for processing, the prototype package supplied by RST for the practical tests did require cabling (see diagram and picture in Figs. 2.1 and 2.2).

[1] For detailed specifications, see `www.analog.com/UploadedFiles/Data_Sheets/178749895ADXL103_203_a.pdf`.

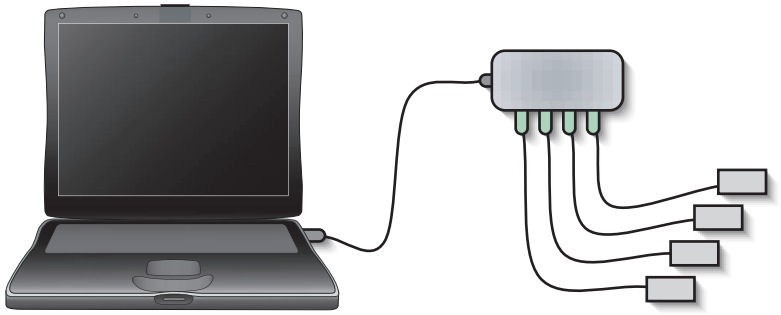

Figure 2.1: The hardware setup of RST's acceleration sensor package. The metal-cased circuit boards with the sensors (represented by the four little boxes on the right) are attached to a hub (rounded box), which in turn is connected via USB to the processing computer (in this case a notebook).

Figure 2.2: RST's acceleration sensors attached to a person. The actual attachment is achieved using velcro fasteners, the USB hub is worn mounted on the person's belt.

The following table summarizes the sensor package's technical specifications and contrasts them with those envisaged for a market-ready version:

	prototype	final version
communication with computer	wired USB	wireless USB
sampling rate	160 Hz	500 Hz
acceleration range[2]	[–2 g, +2 g]	[–2 g, +2 g]
sensitivity	8 bit (16 mg)	10 bit (4 mg)
axes measured per sensor	2	2
number of sensors	4	8

Discussion of Properties

The manufacturer's data sheet gives the acceleration range as $\pm 1.7\,g$, but the output data range attainable by moving the sensors is clearly twice as large than that covered when merely turning them through Earth's gravity field ($\pm 1\,g$). Also, the accelerometers' native sampling period is variable up to 2.5 kHz, merely fixed by RST's driver software, which has to take care of other hardware constraints. In the final version, the software's theoretical limit is 8000 Hz, which gets divided among the used channels. So if not all of the 16 available channels are used, more then the stated 500 Hz are be possible.

All properties may raise questions about their potentially limiting nature. The physically restricting nature of the prototypes' cabling set aside, a discussion of the remaining ones follows.

- *Sampling rate*
  By Nyquist's theorem, a sampling frequency of 160 Hz allows the reconstruction of periodic accelerations at frequencies of up to 80 Hz without aliasing. Considering the rate at which human (dance) movements occur and the spacing of rhythmic elements in music, this appears to be comfortably sufficient to capture their corresponding features even with a certain smoothness. Smith [p. 66 Smi99], for example, calls a period of 200 Hz appropriate.

- *Acceleration range*
  Depending on the orientation of a sensor, even at rest there can naturally be a reading of ± 1 g, making the addition of any significant acceleration away from Earth result in clipped values near ± 2 g. Unfortunately, the sensors used make this worse by not being centered at zero, i.e., actually having a range of about [–2.8 g, +1.2 g]. When moving a sensor with one's hand, this problem can

[2] Note that the italicized g denotes a unit of acceleration equalling that of Earth's gravity at sea level, about 9.8 $\frac{m}{s^2}$.

be offset by looking at the resulting data and tilting the accelerometer into a base position that makes clipping unlikely. In a real dance setting, this is impossible. Whether or not this has an effect on the used algorithms' effectiveness is discussed in Chapter 6.

- *Sensitivity*
  Just like the sampling rate (the temporal resolution), the sensors' sensitivity (the acceleration resolution) is more than sufficient. Fig. 2.3 gives an impression of the graphed sensor data's smoothness.

- *Axes measured per sensor*
  Not being able to read data for all possible directions appears an obvious deficiency: the prospect of having sensors that measure acceleration in all three dimensions may seem to enable an extrapolation of the moving limb's direction as well as a correct scalar value for its acceleration. By double integration over time, one might even hope to derive its position in space. Distinguishing mere turning of the sensors from their directional movement seems to be hard, but perhaps feasible. However, further thought makes it clear that the task can only be about the evaluation of the time and scalar magnitude of accelerations, not their direction and derived values:

  – The sensors' starting positions and orientations would have to be measured beforehand.

  – Imprecision will accumulate, especially when accelerating out of the sensors' range. Gradually, it will degrade any positional reading.

  Should a third axis turn out to be useful, however, one might position two sensors at the same limb, thus acquiring data on 'four' dimensions (one being redundant) and having one sensor less for the other limbs. As mentioned, a third axis would allow the extraction of an movements' direction and absolute acceleration.

- *Number of sensors*
  Admittedly, even with three measured axes per sensor, eight sensor locations are not enough to fully describe a person's movements. But for this thesis' purpose, the evaluation of four sensors' data is sufficient, as it allows the testing of algorithms that combine their readings. Those readings' relevance, in turn, could be improved by determining the most promising positioning setup, given a particular style of dance.

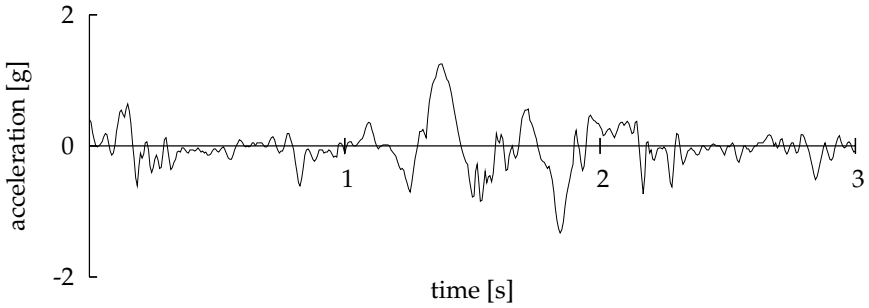

Figure 2.3: A graph showing three seconds' sensor data featuring one significant movement in the middle. As no interpolation of samples was used, the picture gives an indication, though not a proof, that the 256 available acceleration values and the 160 samples per second sufficiently capture the movement.

2.1.2 Driver Software

Currently, only drivers for Windows available

As RST provided the driver software for the accelerometers only for Windows, a Windows PC always had to be involved when reading the data, be it for near-real-time processing or recording. In addition to the actual USB driver for the general-purpose translator unit they had used (a Cypress EZ-USB FX2[3]), they included a program named Sense4Motion[4], a screenshot of which can be seen in Fig. 2.4. It can graph sensor data, record it into a *Matlab*-format [5] text file and offers the possibility to set the sampling rate.[6]

Sense4Motion sends data, but delayed

Most importantly, Sense4Motion acts as a TCP[7] server that allows processing of the data without having to rely on the intermediate step of saving it as a Matlab file. Considering the closed-source nature of the programs provided, this enables a comparably versatile encapsulation of the data. However, in the old version, a data packet is only sent once 32 samples have been collected, thus adding an inescapable delay of 200 ms between a sensor excitation and its processing. Such a delay severely hampers the real-time aspect of the implementation, but poses no limitation to the rhythmic analysis as such.

[3] `www.cypress.com`

[4] Actually, the whole kit of sensors and software is to be marketed under that name, as can be seen on `www.sense4motion.com`.

[5] Matlab is a software for fast numerical computing and its textual file format thus a suitable choice to make. See `www.mathworks.com/products/matlab`.

[6] In the old 8 bit version, a setting of 6 ms per sample (166 Hz) in Sense4Motion actually results in a sampling period of 6.25 ms. Likewise, in the new version, a setting of 3 ms results in a sampling period of 2.46 ms.

[7] *Transmission Control Protocol*, specified by the *Internet Engineering Task Force* in their *Request For Comments* (RFC) no. 1122, available on `www.ietf.org/rfc/rfc1122.txt`.

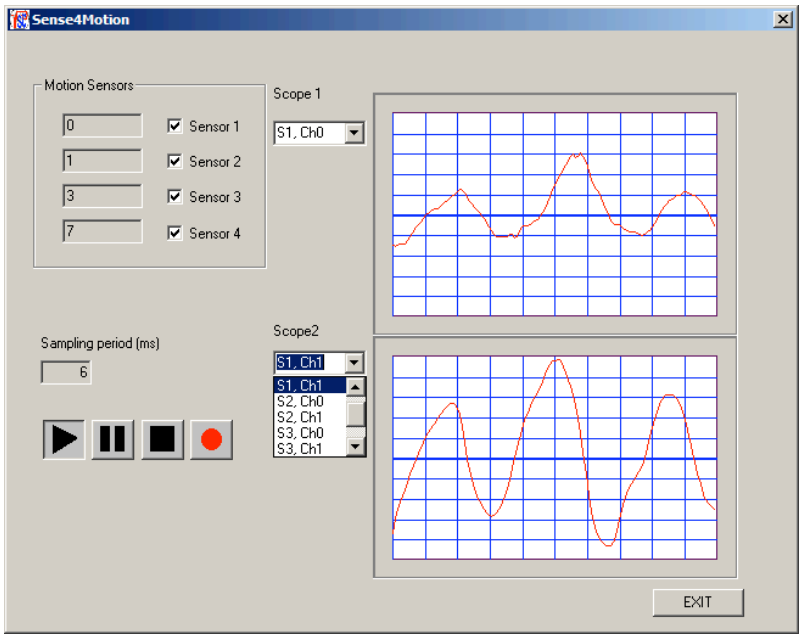

Figure 2.4: The Sense4Motion application. On the top left, the sensors can be chosen whose data should be processed (in the prototype, these must be the four numbers displayed). Below, the desired sampling period can be entered. Both choices are fixed while a recording session is in progress, which can be started and stopped using the black buttons on the bottom right. The red recording button saves the data to disk in Matlab format. In the window's centre, two sensor channels can be chosen whose data should be graphed on the right, covering the past second.

2.2 Rhythm and Other Terms

Given the technical setting, a more precise understanding of this thesis' scope still requires the discussion and definition of various, in part ambiguous, terms. For example, the previous use of the word "rhythm" will hardly have forced any reader into contemplation. Beyond the general agreement that the term somehow captures the temporal structure of music, the question of which information exactly should be yielded by DanSense deserves some thought.

Rhythm is an ambiguous term

In his thesis on rhythmic analysis of music, Seppänen [Sep01] illustrates the vagueness of rhythm-related terms and their thus limited use for formal statements, by quoting the *Oxford Dictionary and Thesaurus* [Aba96]:

"**rhythm** *noun* **1** periodical accent and duration of notes in music. **2** type of structure formed by this. [...] (*see also* **metre**, [...])"
"**metre** *noun* (*US* **meter**) [...] **3** basic rhythm of music."

It seems necessary to find one's own fitting definitions to properly work with this matter. Not only because of the ambiguous wording in encyclopaediae, but also because of their fixation on rhythm as being a property of music.

2.2.1 Rhythm without Music?

Taking on a challenge similar to this thesis in trying to let a dancer influence music, Guedes [Gue05] studied researchers' views on definitions of rhythm in general and the term's applicability to dance in particular. Adopting Parncutt's [Par94, p. 453] definition that "a musical rhythm is an acoustic sequence evoking a sensation of pulse", he tried to justify his hypothesis that "dance movement possesses musical qualities at the temporal level".

Guedes notes experimentally shown ambiguity in both the frequency a listener would tap to and the intuitive counting he would apply. Furthermore, the rhythmic perception when watching a dancer is strongly determined by accompanying music and difficult to attain in silence. But already for physical reasons, a dance cannot capture all of music's rhythmical elements. On the other hand, dance also adds spatial elements to the linear appearance of music, so even a person could not be expected to easily derive a rhythm by watching a dancer.

Consider dancer's felt rhythm as benchmark

As conceded before, it is hard to extrapolate 'fitting' melodic music from mere rhythm, so it will be even harder trying to reconstruct music that led to the dance. What can be aimed for is to capture not a rhythm visually perceived, as Guedes contemplated, but that felt (or intended) by the dancer. As Smith [Smi99] states: "Alternatively in prepared performance situations, the rhythm can be transduced from sensors, measuring the intensity of the instant of the beat, which is generalised as a measure of intended accent." For this, a definition devoid of acoustic references has to be chosen. This could then also be applied to any kind of 'rhythmic' movement, not necessarily music-induced dance.

2.2.2 Rhythm without Order?

Smith, also treating the modelling of rhythm, though in a purely musical context, points out that non-Western rhythms might lack persistent hierarchical structures. Similarly casting into doubt the decisive role of repetition, he quotes Clarke's [Cla85] definition of rhythm as "the grouped organisation of relative durations without regard to periodicity".

Although it may be justifiable to call a non-periodic and per se non-hierarchical sequence of intervals a rhythm, an algorithm detecting such would a) have no means of checking its own results for sanity and b) be of questionable use. Especially keeping in mind de Jong's intent of exploiting rhythm's harmonic properties, this thesis can with a good conscience be limited to Western hierarchical and periodic rhythms. Even those may turn out difficult to detect when the intended rhythm actually consists of periodic variations of what on first sight appears to be the recurring pattern.

2.2.3 Towards a Useful Definition

Another definition is Dowling's and Harwood's [DH86, p. 185]: they call rhythm "a temporally extended pattern of durational and accentual relationships". This fits quite well what is needed for mapping to harmonic patterns. Prior to coining a definition for rhythm, though, the subjects of those 'relationships' should be defined. In this paper, they are called an

Definition: *Impulse*

IMPULSE:
An accented event in time.

In a music setting, impulses would be sounds and their accentuations could be determined by their volume. In a dance setting, they could be movements with their respective maximal momentum. Simplifying the above quote a little, rhythm can now be defined as follows:

Definition: *Rhythm*

RHYTHM:
A repeating series of accentuations of and intervals between impulses.

An intuitive graphical representation of an example rhythm is shown in Fig. 2.5. A representation using traditional (Western) music notation would suffer from a lack of precision concerning the differentiation between impulses' various magnitudes of accentuation. That is probably due to its prescriptive instead of descriptive focus: rather than capturing an interpretation of a piece, it invites the performer to his or her own.

The given definition captures both aforementioned properties of Western rhythm: periodicity by demanding a series that repeats and hierarchy by attributing the series' constituent impulses an accentuation. The hierarchical aspect could be made more explicit by talking of a "repeating hierarchy" (in data-structure terms, a tree). However, deriving such parent/child relationships among impulses appears speculative compared to the physical measuring of a single one's accentuation. Should this information once be needed and not be computable from the mere

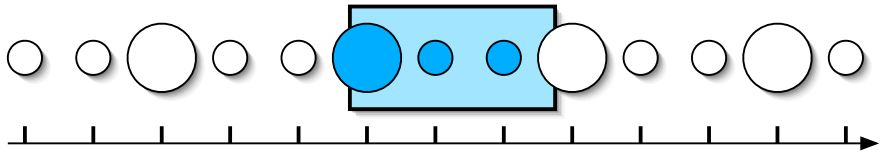

Figure 2.5: An intuitive representation of a rhythm. Each circle along the horizontal time axis represents an impulse, its diameter signifying the impulse's magnitude. (Not some temporal extent: the impulses are supposed to occur precisely at the points in time marked on the time axis.) For example, the impulses could be drum hits of two kinds of strength: every third hit is of the stronger type. As the pattern *strong–weak–weak* is repeating, it can be called a rhythm. One exemplary instance of the rhythm is coloured blue, an underlying rectangle specifying its extent. In fact, any period of the rectangle's length could have been chosen as the rhythm.

rhythm as defined above, it could still be given as additional information.

Also left open is the question of whether to give the intervals between a rhythm's impulses relative to each other or relative to the local timeframe of the current recurrence's period. What is important to note is that in order to actually get intervals between impulses, those have been defined to occur at one point in time, not over a period of time. In case a precise point in time does not reflect the physical realities of the sound, movement or other event in question, this can be additionally specified. Thus, an impulse can be attributed

- a magnitude,
- a point in time (at which it occurs) and, in case the time of occurrence cannot be specified exactly,
- a certain *spread* over time, i.e., an interval over which the actual time can be considered distributed with varying probability.

In the context of accelerometers as this thesis' hardware base, magnitude of movements will depend on the respective movement's acceleration, the precise function employed being a topic of Section 4.1.4. Which point in time (and which spread) to assign to a movement is discussed in the same section. One candidate is the movement's assumed start, another that of its maximal acceleration. Whatever is chosen, the intervals defining a rhythm can be called

INTER-IMPULSE INTERVAL (III):
The time passing between two, not necessarily successive, impulses.

Definition: *Inter-Impulse Interval (III)*

This definition parallels that of *inter-onset intervals* (IOIs), a term used in the rhythmic analysis of music, for which sounds' *onsets* are unquestionably the point in time to evaluate as their most 'striking' moment. IOIs are going to be discussed in Section 3.2.1. To see an example of how an alternating sequence of movements of short and long durations can make a numerical difference between the resulting IOIs and IIIs, see Fig. 2.6.

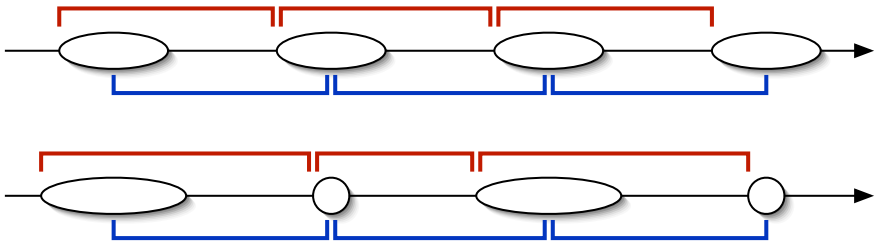

Figure 2.6: When inter-onset intervals (IOIs) and inter-impulse intervals (IIIs) make a difference. The top diagram shows an equidistant sequence of equally spread-out impulses along a time axis, the bottom one a sequence where long and short impulses take turns. As opposed to Fig. 2.5, the horizontal extent of the ellipses depicting the impulses does have meaning: the temporal extent over which they occur. In each diagram, the intervals between the beginnings (IOIs, red) and between the centres (IIIs, blue) of successive impulses are marked by brackets 'looking from' the top or bottom, respectively. In the top diagram, the values of IOIs and IIIs are identical. In the bottom one, however, it can be seen that although the impulses' centres are equidistant and the IIIs thus all equal, the IOIs vary. Therefore, in cases where the temporal extent of beats greatly varies, rhythmic analysis can yield different results when using IOIs versus IIIs. This should be kept in mind later on, as the same differences occur depending on whether a movement's beginning, centre or other position is used as its defining moment.

An III can, in addition to this duration, also be characterized by a magnitude and a spread, derived from the constituent impulses' respective values.

Taking the above-mentioned attributes of an impulse, a rhythm as defined in Section 2.2.3 can be represented by a sequence of 'conceptual' impulses

- whose magnitude represents its accentuation,
- whose 'time stamp' has no absolute meaning, but only in relation to that of other impulses in this sequence and
- whose spread mirrors the uncertainty with which the relative time stamp is given.

In addition to this internal structure, embodied by the mere relative temporal position of the rhythm's conceptual impulses, a kind of duration must be given that indicates after which time the sequence repeats. The (Western) notation of music usually divides a piece into equal intervals called *measures*. As this term is well known, and due to parallels yet to be pointed out, in this thesis' context it is to mean:

MEASURE:
One among a rhythm's repeating instances.

Definition: *Measure*

The term sought for a measure's duration can thus be:

MEASURE LENGTH:
The duration of each of a rhythm's repeating instances.

Definition: *Measure Length*

Whether or not the pace at which a rhythm's instances succeed each other should be considered an integral part of rhythm may be an academic question. But a discussion in Chapter 4, *Algorithmic Design*, will show that in a real-time analysis of an ongoing sequence of impulses, it makes a difference whether the algorithm used treats their relative spacing or their absolute intervals as 'more basic'.

Not only because specific impulses are allowed to be defined imprecisely in time, this also applies to rhythms: slight variations in successive measures' length and in the internal impulse structure should not be taken as a reason to classify the changed version as a different one, rather as one of many expressions of an ongoing rhythm.

2.2.4 Classical Terms and Metric Aspects

As a comparison to and to prevent ambiguity to standard terms of music theory, it should be mentioned how some of such fit in and why this paper concentrates on the ones introduced previously. For an in-depth introduction into classical music theory, one may refer to Michels [Mic05].

The *time signature* given for (a section of) a piece in a musical score specifies how many notes of which kind form one measure, the *tempo* indirectly indicating those specific notes' length in time. For example, a waltz' signature of $\frac{3}{4}$ indicates that each measure encompasses notes and pauses with the combined duration of three quarter notes. These base notes' dominance gives rise to the term

Definition: *Beat*

BEAT:
The dominant regular base pulse in an impulse series.[8]

Here, 'base pulse' is to stress that it usually occurs at a higher frequency than the measure.

A tempo can be given in *beats per minute* (*bpm*): a tempo of 120 bpm clarifies that each beat note is half a second in length. Relying purely on the notational hint on the beat, this tempo gives the waltz example a measure length of 1.5 seconds. But a piece with a signature of $\frac{3}{4}$ may be sprawling with eighth notes, making them the objective beat notes. Similarly, its measures may have alternating structures, the de-facto measure length thus spanning two notated measures.

Metre subsumes the hierarchy of impulse positions

More than merely being a grouping of notes to make them more legible, the time signature foreshadows rhythmic aspects in the sense that recurring emphases are likely to occur on the same positions in each measure. The hierarchy of potential note positions in a measure that results from repeated subdivision of the temporal grid implied by the time signature is called *metre*.[9] One can thus say that the beat forms the main *metric level*. See Fig. 2.7 for an example.

Figure 2.7: An example of a grid of metric levels, or *metric grid*. The notes above are part of a waltz score (ignoring their pitch), the lines below indicate its metric grid: the long ones delimit measures, the medium-sized ones divide measures into three (quarter notes), the short ones are on a sub-beat level allowing for eighth notes. The last note is shaded because it starts a measure not shown.

Rhythm as choice of positions in metre

A rhythm, as defined previously, can be interpreted as a choice of positions in the metre and the assignment of emphases to them. In a waltz, for example, usually the first of the three quarter notes gets the most emphasis, the two others get a smaller amount, both the same.[10]

[8] Some call the individual beat-level impulses *beats*, but for this thesis, the singular events will always be called *impulses* to prevent ambiguity.

[9] From a cognitive viewpoint, Smith [Smi99] defines metre as "the occurrence of regular subjective or objective accentuation". However, with the exception of pieces with continuous Swing-like variations of intervals, these accentuations should occur on said temporal grid.

[10] See former Fig. 2.5 for a graphical representation and Section 3.2.1 for a short discussion of whether one can at all say with which impulse a (physically, not notationally) given rhythm 'begins'.

As noted before, measures, the beat and their surrounding metric hierarchy implied by notation would not necessarily be considered such when applying an objective metric to the music itself. All the more, as this thesis is not concerned with inferring underlying music's notation from movements, the metric levels of measures and the beat should not be rated by their closeness to said notation, but instinctively judged by the moving person.

In any case, defining a specific impulse series' beat may be difficult. As it is characterized by its frequency rather than by individual member impulses, a beat may have gaps like those shown in Fig. 2.8.

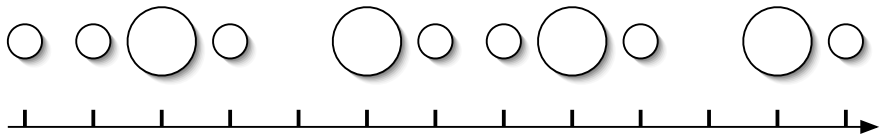

Figure 2.8: A beat's prevalence despite gaps in its impulse series. The picture shows a sequence of impulses, whose dominant mutual interval is shown using tick marks on the time axis below. Apparently, keeping the enumeration *strong–weak–weak* of Fig. 2.5, the third impulse is missing in every second measure. Still, the tick marks can safely be said to indicate the rhythm's beat or, better, the beat level on its metric grid.

Analogously to the interval between a measure's and its successor's beginnings (the measure length), a beat has its

BEAT INTERVAL:
The inverse of a beat's frequency.

Definition: *Beat Interval*

In the case of a beat without gaps, the interval could simply be defined as that between successive beat impulses.

Knowing an impulse series' measure length and beat interval, their quotient describes a subdivision like that implied by the time signature in a musical score. Henceforth, it is called[11]

METRIC QUOTIENT:
The quotient of an impulse series' measure length and its beat interval.

Definition: *Metric quotient*

For example, the metric quotient of a waltz is 3, as there are three quarter notes in each measure.

Two of the three values measure length, beat interval and metric quotient suffice to describe a piece's rhythmic structure and tempo. The

[11] Here, that is. Others, like Gouyon and Herrera [GH03], call this quotient simply "the metre".

time and tempo signatures in music notation, in contrast, appear redundant from a mathematical point of view: a piece given in $\frac{3}{4}$ at 90 bpm can easily be converted into one in $\frac{6}{4}$ at 180 bpm by doubling all notes' durations, as shown in Fig. 2.9.[12]

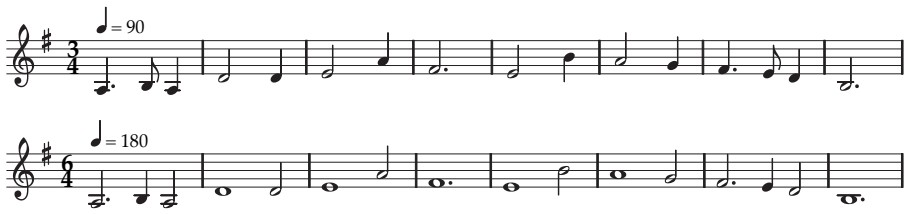

Figure 2.9: The redundancy of classical music notation. Shown are two score versions of same melody, the beginning of *The Inner Light* by Jay Chattaway. Tempo and metric structure are identical, although the time signatures differ.

Some, like Smith, talk of a special *tactus* level within the metric hierarchy as the one to which foot-tapping occurs. Due to man's physical constraints, tapping is not necessarily possible at the beat level or even below, but might also occur more often than once per measure. According to Smith [Smi99, p. 35], tapping usually occurs at the metric level close to the 600 ms interval. In the dance context, however, this tactus level should coincide with the beat level, anyway, as a dancer probably already moves at the pace he would tap to.

An aspect the 'professional version of tapping', namely conducting, brings up is what to consider as the beginning of a rhythmic pattern: independent from local accentuation, a conductor needs to signal the performers when a measure starts. In the description of Fig. 2.5, the positioning of the window specifying one measure within the impulse series was considered arbitrary. But it appears sensible to let a measure's beginning coincide with an impulse on the beat level. Reminiscent of the downward movement of the conducting movement at a measure's beginning, this is called the

Definition: *Downbeat*

DOWNBEAT:
The position on the metric grid's beat level that starts a measure.

To get a unique representation of a rhythmic pattern, one might choose the most accented of beat-level impulses, or in the case of ambiguity at least one of those most accented.

[12] Possible objections stating that the piece's implied counting 'obviously' demanded three beats per measure, could be met by changing the time signature from $\frac{6}{4}$ to $\frac{3}{2}$.

3 Related Work

There are various ways in which past research can be related to this thesis. A multitude of works have dealt with data gathered from a dancing person, though not with the intent of deriving a representation of the underlying rhythm. Others were concerned with rhythmic structures, but did not use accelerometers as data source. This chapter is thus divided into sections dealing with dance-related and with rhythm-related works. Representatives of approaches have been chosen to give an impression of the breadth of methodologies that have been applied and could be useful for this thesis.

3.1 Dance Movements as Input

Among the attempts at using movement as an input device (which, strictly speaking, even includes using a computer keyboard), this section looks at such that use dance-like free movements as input and involves music either as inspiration for the movements or as output somehow

influenced by the input. While the discussion is organized by the type of sensors used, a comparative study of hardware solutions is beyond the scope of this thesis: the used accelerometers are externally provided and thus not subject to improvement. A relevant aspect to look for in the mentioned works, however, is how the data gathered is interpreted.

3.1.1 Sensing Floors

Several approaches of capturing dance movements involve an array of sensors on the floor to be danced on. Johnstone et al.'s *PodoBoard* [Joh91] consisted of aluminium tiles which, to capture movement, depend on special shoes to complete electric circuits. Strips instead of tiles were used by Pinkston et al. [PKM95], who got rid of the shoe requirement by using force sensors. Paradiso et al.'s *Magic Carpet* [PHS+00] (begun 1997) used a grid of piezo-electric wires whose capacitance changes with the force exerted on them, enabling a positional interpretation by measuring voltage across rows and columns. Further information was gained through radar sensors that capture the dancer's upper body. Optical sensors were used in *Litefoot* [GF98] by Griffith et al., either depending on shoes' reflection of lights in the floor or on shadows cast among external lighting. Srinivasan et al. [SBQK05] employed pressure sensors again, but at a density allowing readings to be taken for sub-areas as small as a toe. Also, their set-up was modular, so it allowed the combination of various sensor mats to a large one.

Looking at how spatial and pressure resolution as well as response times were improved over time, allowing evolution from a floor-based percussion system to combinable pressure mats of almost orthopaedic use is not very interesting in this thesis' context. As for the relevant processing of the readings gathered, it turns out that none of the mentioned works tried to extrapolate rhythm. Either they were not concerned with any specific use of the technology in the first place (Srinivasan et al.) or merely used an abstract mapping of positions to instruments, pitches, etc. (Litefoot, Magic Carpet).

Feet could be good sensor location

What is to be kept in mind, though, is the notion to concentrate on the feet as the most expressive part of the body in many types of dances.

3.1.2 Body-Mounted sensors

Just as on the floor, various sets of sensors have been designed to be worn or mounted on a person's body. In the *MidiDancer* project [CS89], Coniglio measured the flexion of limbs' joints. Paradiso et al.'s

DanceShoe [PHBT00], in its latest version, produces readings of 16 different parameters, including those of differently tilted accelerometers and pressure sensors at various positions. Mere accelerometers like intended for this thesis were used by Feldmeier [Fel02] for a project that focused on creating auditory feedback from a large group of dancers' motions instead of just one.

The *Sonic City* [MJ03] project, by employing microphones, light sensors and metal detectors, shows that there is no limit to the complexity of parameters one may want to include.[1] It also shows that the project's intent was not limited to capturing a person's movements (like on a stage), but also its environment (when strolling downtown) in order to facilitate the creation of a fitting musical background. Other projects using a multitude of (wireless) sensors are those of Hromin et al. (*CodeBlue* [HCV+03], which focuses on group interaction) and Park et al. [PCS06]. Barry et al. [BGK+04] mapped *Motion to Emotion* in the context of a Japanese dance improvisation method. Continually determining motion's intensity, direction and continuity using statistical and frequency analyses over a time window, these were mapped to an "emotion space" whose current coordinates would then influence visual effects on stage.

The use of accelerometers in combination with the aim of creating music from dancing raises special interest as to how Paradiso et al. or Feldmeier evaluated the acceleration data. Paradiso et al. gave "improvisational dancers a 'palette' of action-to-sound rules and relationships" of varying nature at different stages of their project, using acceleration sensors either as a kind of potentiometer (e.g., correlating a tilt angle with an instrument's pitch) or as a binary switch (shock movements causing individual sounds or playing a drumroll while doing a handstand).

Use FFT to find important frequencies

Feldmeier dealt with large numbers of dancers and thus had to find an efficient way of evaluation: he did not send actual acceleration measurements from the sensors, but let them transmit a pulse when a certain acceleration threshold was exceeded. The numbers of these pulses received over various time intervals, as well as these numbers' change, are taken as a hint on the dancers' satisfaction with the current piece, among other interpretations. Applying the *Fast Fourier Transformation* (FFT; see Appendix A.1) to the signal of pulses, the most prominent frequency was used to determine the played piece's tempo.[2] As the measure structure of the piece was known, the points in time with most activity could be analysed over some time. Finding their most probable re-occurrence relative to the next measure, a drum could be played at that point.

[1] See `sensorwiki.org` for a technical overview or Bongers [Bon00] for a taxonomic analysis of sensing technologies that could be used in musical applications.

[2] Actually, the DJ was allowed to set the tempo slightly higher or lower than what was measured, in order to gradually influence the group's pace.

The newer multi-sensor projects were rather concerned with technical aspects of their respective wireless standards (Bluetooth and WLAN[3]) than with the question how processing could extend beyond a mere rule-based approach. Barry et al., in turn, applied various mathematical transformations to accelerometer data, but only to use probabilistic models to detect the kinds of movements that could be mapped to their emotion space. Their work did not deal with rhythm, but if one were to attempt the detection of 'classical' types of rhythms, a look at the abruptness of movements, for example, may give hints besides rhythmic structures.

3.1.3 Spatial Tracking

Instead of measuring data where movements take place, others took the approach of remotely tracking people's position. Lima et al. [LMB+96] attached ultrasound emitters on the dancer's body and used corresponding detectors to continuously derive positions. Koppel [Kop98] approached this similarly by using a camera to track the relative position of coloured markers, as done in the film industry for the choreography of animated actors. In *Music via Motion* [Ng04], Ng compares successive video frames without such help, evaluating changes in colour composition and recognizing facial expressions: reconfiguration of costumes may lead to changed instrumentation of music, an open mouth may add a low-pass filter. Dobrian et al. [DB03] tried out various mappings of body movements to music, based on similarly simple rules. Guedes [Gue05] shows that the mere summation of brightness changes over the whole video frame can already be a useful input for determining a dance performance's tempo.

Smooth results before output

The latter work is the only one of those mentioned using a mathematical analysis of frequency components to gather high-level information about the performance instead of converting it into effects. Guedes used a bank of 150 band-pass filters to determine the brightness changes' main frequency, as well as its harmonics. This means that for 150 frequency ranges, he filtered the signal to that respective range. Comparing the resulting signals' amplitudes, he could see which frequencies had contributed the most to the original signal. He gives efficiency reasons for not employing the FFT, pointing out that only a small range of frequencies was actually of interest. Basically, frequency analysis seems a useful means also for this thesis' aim. Another aspect to keep in mind

[3] WLAN is short for *Wireless Local Area Network*. The two standards have been designed by the *Institute of Electrical and Electronic Engineers* and are formally named *802.15.1* (Bluetooth, downloadable from `standards.ieee.org/getieee802/802.15.html` and *802.11* (WLAN, downloadable from `standards.ieee.org/getieee802/802.11.html`).

from Guedes' work is to have the derived tempo only 'reluctantly' react to changed input, thus yielding a more constant output.

3.1.4 Combined Approaches

Kwon and Gross [KG05] combined body-mounted accelerometers with visual tracking, aiming to build a system for motion training. Primarily with martial arts in mind, movements could be recorded as an example or their proximity to the optimum evaluated. For their system to learn exemplary movements, they modelled them as probabilistic processes.

Find out how significant motion looks on sample graph

As classification of singular movements is not part of this thesis' scope, machine learning aspects have to be discarded. Kwon and Gross' notion of 'motion chunks' (i.e., that singular movements are flanked by periods of limited activity) is especially useful when dealing with the very controlled and sequential movements of martial arts. It appears prudent for this thesis to also check arbitrary (dance) movements' acceleration graphs for such features.

3.2 Rhythmic Analysis of Music

Symbolic vs. recorder music

As could be seen, works concerning dance movements as input greatly differ in the kinds of sensors used for capturing motion. Depending on the means, also the kind of data available for analysis differs, ranging from single impulses for a specific event to quasi-continuous streams of measurement values. Similarly, music can be represented in two ways: either as a symbolic description like a score, or as an audio recording. Digitally, the former is usually stored according to the *MIDI* standard[4], containing information on when to play which instrument for how long, at which pitch, etc. Audio recordings physically represent a series of air pressure readings as captured by a microphone.

Keeping in mind this thesis' aim of deriving rhythm from acceleration sensor readings, the derivation of rhythm from microphone recordings appears the closer-related task. Studying the means others have used for this should thus prove most helpful. The algorithm finally employed does include a conversion of accelerometer readings into a symbolic representation of movements, so for the remaining task of deriving a rhythm, the methods used for MIDI data could also be of use.

Before looking at specific approaches, it should be noted that in music, various properties of notes influence the perceived rhythm. Fig.

[4] *Musical Instruments Digital Interface*, specifications can be ordered on `www.midi.org/about-midi/specshome.shtml`.

3.1 shows a simple example of equidistant notes of the same length, whose grouping stems only from their pitch. A dancer may individually choose how to express this pattern in his (or her) movements, yet the physical property of actual signal frequencies changing cannot be expected to occur in his accelerometer readings. Likewise, changes in instrumentation or chords' harmonic composition (Smith [Smi99, p. 22] lists other possible accents) have a profound effect on a listener, but have no directly corresponding feature in motion. Consequently, algorithms extracting rhythmic hints also from the dynamics of the "acoustic carrier" (Smith) have to be examined as to how well they would work without this possibility.

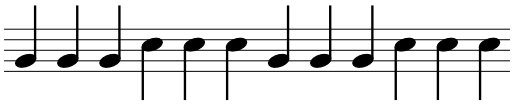

Figure 3.1: Pitch-induced rhythm. The apparent arrangement of the shown notes in groups of three is founded merely in their pitch. An algorithm trying to extract rhythmic information by looking at pitch changes would be successful with an audio recording of this score. If a dancer moved to this rhythm (or had it in mind), he would perhaps move more vigorously at each triplet's start, yet the pitch information itself would be lost and an algorithm based on that thus be rendered useless.

The works mentioned are neither necessarily those pioneering their fields, nor represent the most recent refinement available. They have been chosen to illustrate the approaches they represent and to explain the relevant aspects in a way minimizing the number of terms not explained. For a comprehensive and concise overview, yet without explanations, please refer to Collins [Col04].

3.2.1 Processing of Symbolic Representations

Probabilistic MIDI Quantization

One traditional application of rhythmic analysis is the *quantization* of MIDI data. There are keyboards (meaning the piano-like musical instrument) that can send MIDI information on how long which key was pressed to a computer. If this information were typeset as a score without removing all the imprecision induced by the performer, it would be barely legible. To attain the intended score, accidental pauses have to be eliminated and notes' durations corrected. For this, knowledge of the played piece's rhythmic structure is helpful: if a note only slightly (compared with its own duration) extends over the end of a measure, it

can be assumed that it should have ended exactly on the border between the measures. Also, measure bars can only then be inserted, guiding the score's reader instead of confronting him with just a sequence of notes.

Cemgil and Kappen [CK03] employed various probabilistic methods for quantization and tempo tracking built around Bayes' theorem. In the given context, this describes the relationship between

- the probability that, given some performance data P, a specific score S was attempted to be played and
- the probability that, given S, the performance P would result.

The authors assigned so-called *prior* probabilities for note positions based on their binary representation's length, thus generally considering simple scores more likely. Deviations from the positions of the score's notes was assumed to follow a Gaussian distribution. Adding similar assumptions concerning the tempo and expected deviations, several real-time algorithms were designed to continually yield what, given the performance data, they considered the most likely hypothesis as to which score was probably being played. Due to performance reasons, also these hypotheses were 'guessed' using probabilistic methods instead of doing an exhaustive analytical search.

Remove noise by determining a probable impulse grid

Besides the use of probabilistic tools allowing the tracing of tempo hypotheses throughout a piece, one noteworthy aspect of this approach is the declaration of a priori probable note positions. Likewise, in rhythm detection from accelerometer readings, the detection of impulses might be helped by expecting them at typical positions within measures.

Determination of Metre using Auto-Correlation

Trying to determine a musical score's metre, Brown [Bro93] first converted the given piece into a series of low-level samples: to every 5 ms interval of the piece's duration, she assigned a sample value proportional to the duration of the note beginning there, zero if none did.

On the resulting signal, she then applied auto-correlation, i.e., looked for similarities between all pairs of samples of a certain distance (*lag*) in time. See Appendix A.2 for a formal definition.

Brown then analysed the most prominent lag times, checking whether the piece's 'official' metric structure could be derived. The lag time with the maximal peak was interpreted to indicate the measure length; the distance between smaller, yet still significantly peaking lag times was taken as the beat interval. She was successful with most of her tested scores, though sometimes, the auto-correlation array was maximal for only half the measure length.

Applying her method to a performance's MIDI recording instead of a perfectly timed score, Brown no longer got a correct indication of the measure length. Already having to 'spread out' note onsets into neighbouring samples ($\pm$ 10 ms) in order to offset some of the performance's imprecision, the example piece's measure length of 3 s proved long enough to add imprecision beyond the 10 ms corrected for.

Derive measure length and beat interval using auto-correlation

Despite this, auto-correlation generally is a method worth trying out to extract rhythmic information. Assuming that imprecise performances' timing information can be blurred to an extent that still enables the distinction between peaks in the graph resulting from auto-correlation, those can provide information on both beat interval and measure length, vital properties of any rhythm.

Rule-Based Downbeat Induction

Besides tempo and metre, another property of rhythm researchers set out to extract was the downbeat[5]. For music, Woodrow [Woo51, p. 1233] notes that intensity and duration both favour a beat's perceived position as the first in a series.

Eck [Eck01] experimentally compared several rule-based approaches as to how well they identified those positions on the metric grid as the downbeat that human listeners' intuitively tapped to. The generalized detection algorithm assumed the metric hierarchy to be known and could thus simply check all metric levels at all possible offsets as to whether they were likely to be tapped to. Fig. 3.2 shows the candidate series, or *clocks*, as Eck calls them.

Each clock gets assigned a score according to the rule used, the highest-scoring clock being output as that indicating the downbeat. All rules computed their score as a function of the number of times that the given clock coincided with accented onsets, unaccented onsets and rests. For example, one rule computed a negatively weighted sum of the latter two. Another normalized the number of accented onsets by multiplying it with the clock's period as to not per se favour the fastest clock, which naturally coincides with the most such onsets. This rule was also the one that in most cases proved to be closest to human perception.

Compare impulse series' collective accentuation

Although in this thesis' case, metric hierarchies are not known a priori, the idea to look at candidate series of impulses to find the most dominant one deserves consideration.

[5] Some authors use the term *beat induction* for *tempo induction*. Here, the former always includes the induction of phase information, i.e., the actual position of beats in addition to their interval.

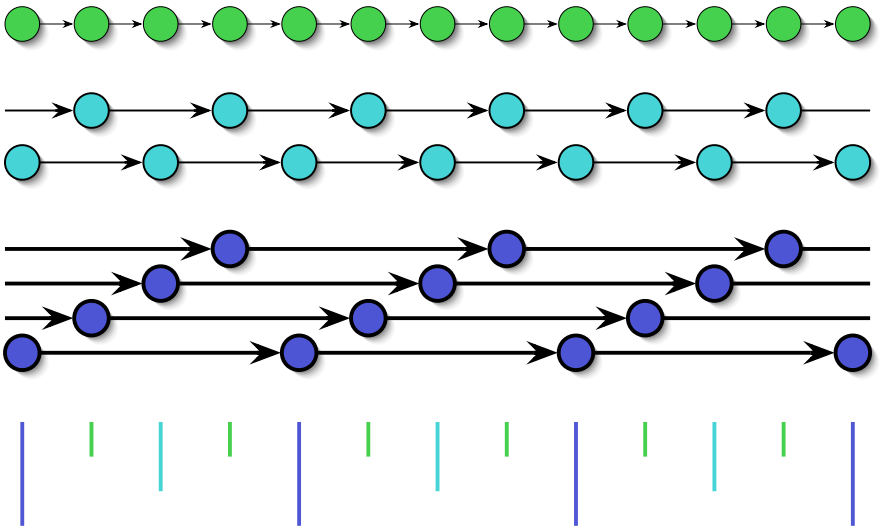

Figure 3.2: Candidate clocks for the downbeat. Given the metric hierarchy shown at the bottom, human listeners' tapping is likely to occur according to one of the series of points in time shown above.

Analyses of Inter-Onset Intervals

A popular method of inferring metric information from symbolic sequences of impulses is to analyse them for frequently intervals between them. The idea behind this is that intervals so fundamental as the beat interval or the measure length are to be found quite often when arbitrarily collecting all intervals. In the case of rhythmic analysis of music, the onsets of sounds are most relevant, so research papers usually talk of inter-onset intervals (IOIs). Thus, a statistical analysis of the IOIs of a series of onsets could give an indication of that sequence's beat interval and measure length.

Dixon [Dix01] clustered the IOIs, calculating weights for each cluster, which would then become the basis of a tempo hypothesis. The weighting was done in a way that enabled pairs of clusters to support each other one in case one's average interval was approximately an integer multiple of the other's. The tempo hypotheses were used to create so-called agents that tried to predict future onsets and were ranked according to their success in doing so.

Another way of summarizing occurrences of numbers is to employ a histogram. Seppänen [Sep01] used one for IOIs and consequently called it *IOI histogram*. Jensen and Andersen [JA03] chose the term *beat probability vector*, hinting on its intended interpretation.

Basically, the histogram is set up by

- choosing a range of intervals one is interested in, e.g., from zero up to a value comfortably exceeding the expected measure length,
- dividing that range into sub-ranges (*buckets*) of a 'width' chosen according to the intended precision,
- initializing a counter for each of these sub-ranges,
- iterating through all pairs of onsets (or those with a magnitude beyond some threshold) and
- for each such pair incrementing the counter of the sub-range that the pair's IOI belongs to.

Formally, given an n-tuple of onset times $t_0, \ldots, t_n$, a maximal expected interval $r_{\max}$ and a desired precision of m sub-ranges, the latter are

$$r_i = [\frac{r_{\max}}{i} - \frac{r_{\max}}{m}, \frac{r_{\max}}{i}) \quad \forall i \in \{1, \ldots, m\}$$

and the m histogram counters reach the values

$$H_i = |\{t' - t \in r_i | t, t' \in T \wedge t < t'\}| \quad \forall i \in \{1, \ldots, m\}.$$

Fig. 3.3 shows the formation of an IOI histogram under ideal conditions, given a sequence of equidistant onsets. It can be seen that this distance between neighbouring onsets stands out in the histogram as the first and highest peak, multiples of this distance being represented by smaller peaks. The larger multiples they represent, the smaller the peaks become.

In the setting of a complex musical score, onsets are anything but equidistant, but still likely to occur on exact positions of the underlying metric grid. Fig. 3.4 shows the effect this can have on the histogram: the peaks are no longer monotonous and the first is not the highest. In the setting of a musical performance, not even the metric grid can be assumed to be hit exactly, resulting in less clearly defined peaks.

In this setting, Seppänen tried to find the interval of the most detailed level of the metric grid, what he calls the *tatum* period, in real-time. In a sense, this period should be the greatest common divisor (GCD) of all other periods, so he computes a weighted GCD of histogram values. To find the 'current' tatum period, he factored in previous results as to gain consistency: for every onset, a histogram of the past few seconds was computed and, in a non-linear combination, combined with the previous step's resulting histogram. The factors were chosen so that the old histogram receives the smaller a weight, the older it is.

Attempting to counter errors in the context of onsets imprecisely detected in an audio recording, Jensen and Andersen built their IOI histogram by filling not only one of its counters, or *buckets*, per IOI, but also neighbouring buckets to a lesser degree. Altogether, each IOI added a

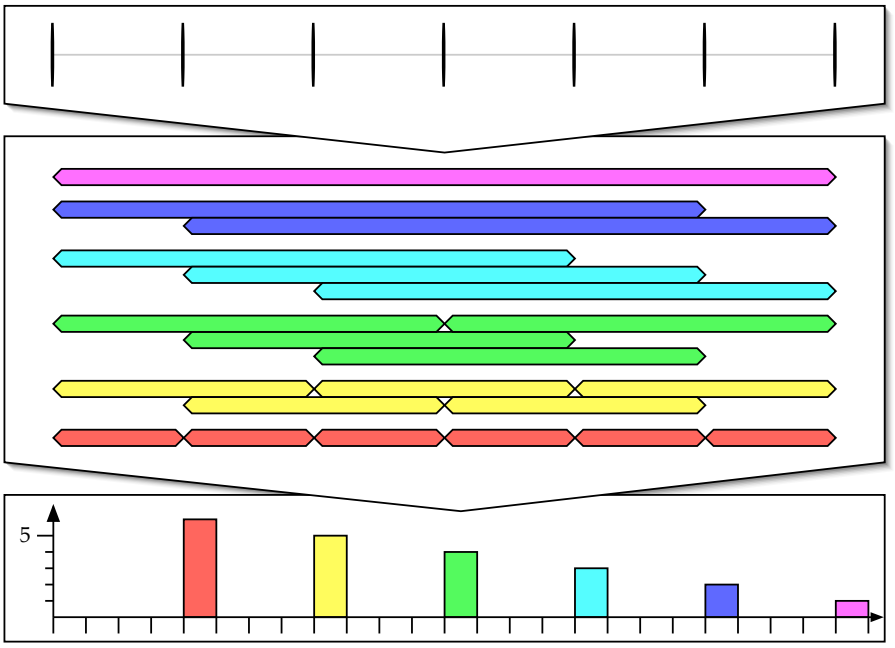

Figure 3.3: Idealized formation of an IOI histogram. On the top, onsets are depicted as vertical markings on an horizontal time axis. Below, all the mutual intervals (IOIs) between these onsets are shown as arrow bars whose ends are positioned exactly below their respective onsets. The bars are grouped and shaded according to their length. Counting the IOIs for each length and assigning the count to the buckets indicated by the tick marks at the very bottom yields the histogram: the shortest IOI occurred most often, namely six times.

Gaussian shape[6] to the histogram: its width mirrored the IOI's uncertainty, derived from the temporal uncertainty of its constituent onsets. The Gaussian's height, i.e., the value added to the bucket representing the most probable IOI, was derived from the constituent onsets' magnitudes. The latter, in turn, stemmed from various frequency-dependent properties of the onset in question.

Try histogram of weighted movement IIIs

Especially with Seppänen's smoothing notion of an histogram trajectory, IOI histograms seem to be suited to find the interval of at least one level of the metric hierarchy. In the accelerometer case, is appears helpful to copy Jensen and Andersen's idea to not merely count inter-impulse intervals (to keep the question open which point in time to assign to impulses), but to assign weights to them and blur an III's contribution to the histogram according to its uncertainty.

[6] A bell-like shape resembling the Gaussian probability distribution.

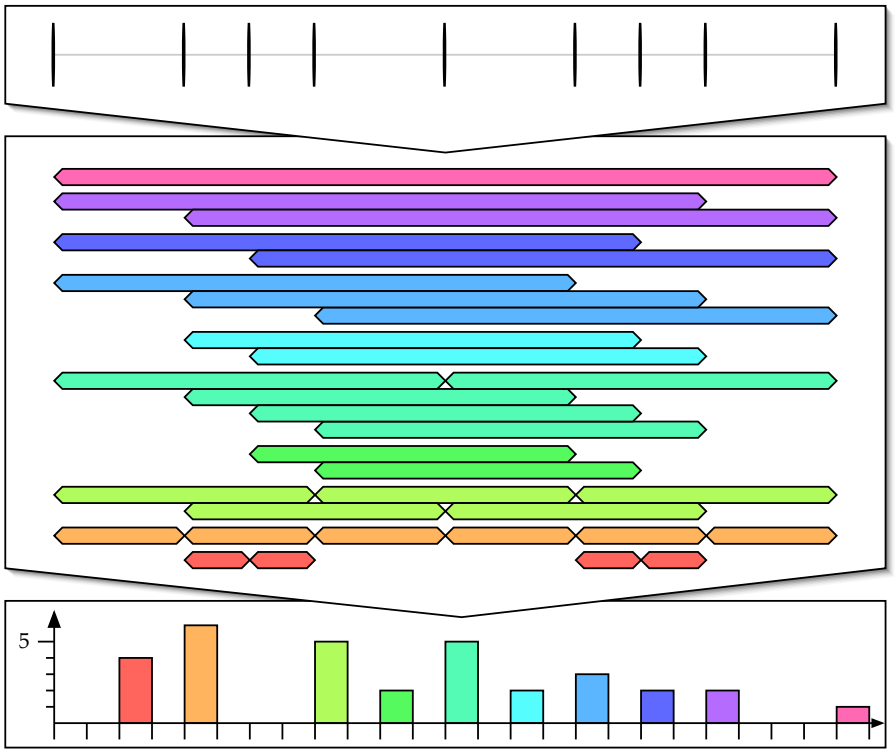

Figure 3.4: Formation of an IOI histogram from a musical score. As in Fig. 3.3, a sequence of onsets is shown, followed by representations of their IOIs as well as the resulting IOI histogram.

Wavelet Transformation

Smith [Smi99] applied the (short-time) *wavelet transformation* to MIDI recordings. Like the Fourier transformation, it can be used to produce a signal's spectrogram, a kind of spectral map that shows the signal's local frequency composition in small intervals. Due to its high resolution in both time and frequency, it yields graphical representations to which Smith then applies special image-processing methods. These extract diagrams that, to his trained eyes, convey much information on the underlying rhythmic structures (see Fig. 3.5 for an example).

Find frequency trajectories in spectrograms

His approach, however, is only partly suited for this thesis, as the recordings are considered as a whole. The application of wavelets as such to do a Fourier-like frequency analysis is certainly transferrable to a real-time[7] setting, but Smith's processing of spectrograms is perhaps not. Especially not to the accelerometer case, where sudden changes and

[7] In the literature, also the term *causal* is used for algorithms that only require information up to a point at which to, e.g., a tempo is to be determined. Such that require knowledge of the future as well are called *non-causal*.

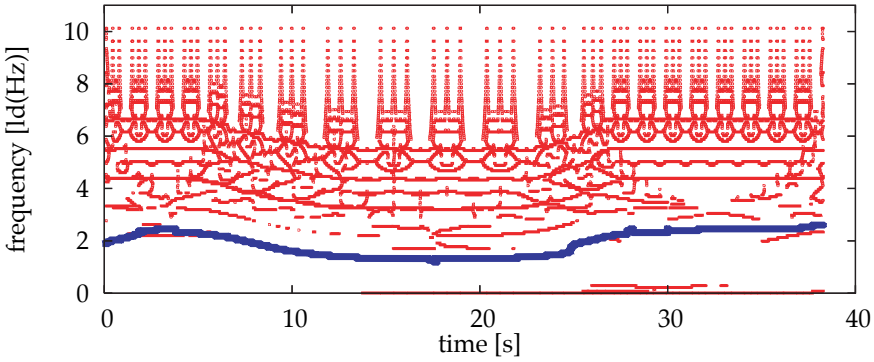

Figure 3.5: Rhythmic analysis using wavelet transformation. Smith, from whose thesis [Smi99] the above graph is adapted, used this method to analyse the frequencies across musical pieces. From the analysis, he used image processing techniques to extract candidates for the tactus (thin red lines and dots). From those, further processing led to a final estimate of the tactus trajectory (graphed as thick blue line).

imprecision compared to MIDI recordings would make the extraction of subtle lines prone to noise, giving an automatically interpreting algorithm a hard time yielding reliable results.

3.2.2 Derivation of Symbolic Representations from Audio Recordings

To process a symbolic representation like onset times as described above, they first have to be detected in the audio recording. This is usually done by filtering the audio signal for specific properties, looking for local maxima or otherwise distinguished features.

As mentioned before, Jensen and Andersen used frequency-dependent properties of small blocks of the recording to assign magnitudes to onsets. Among those properties were absolute and relative high-frequency content and the rate at which frequency composition was changing. Those same properties, determined by Fourier analyses, were used to detect onsets in the first place.

To find onsets for detecting the tatum period, Seppänen filtered the audio signal into several frequency components. In each of these, he looked for instances of rapid increases in volume, which he declared as onsets of a magnitude equalling the measured volume increase. These per-frequency-range onsets were then clustered across all ranges according to their onset times. The resulting clustered onsets were assigned

the onset times' median and their sum of magnitudes as aggregated properties.

How increases of volume could be found deserves some attention, as one cannot simply look for local maxima in the signal's derivative. Considering that an audio signal vibrates between positive and negative values up to several thousand times per second, such short-lived pseudo onsets could be found all the time. Instead, an *amplitude envelope* like shown in Fig. 3.6 must be computed.

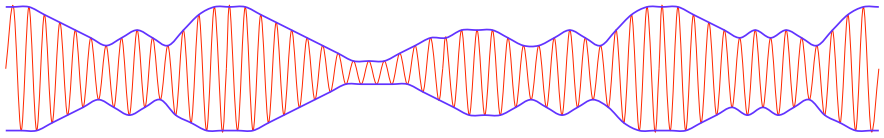

Figure 3.6: An continuous signal's amplitude envelope. The red graph shows a continuous signal, now interpreted as as an audio recording of an amplitude-modulated base frequency of 500 Hz. Changes in volume as perceived by a listener cannot be detected by looking for steep upward slopes of this graph, as it features them 500 times a second. Connecting the local maxima and local minima and thus abstracting from the carrier frequency waves, one gets a shape (red) whose height does indicate volume over time and whose slopes could thus be taken as indicators of volume change. In a real sound setting, though, amplitudes would usually vary far slower than shown in this simplification. Furthermore, in signal processing one would not connect peaks, but first rectify the signal (i.e., negating negative values in order not to deal with positive and negative sides) and smoothing it by convolution with a *Hann* window. About convolution and the Hann windowing function, see Appendix A.1.

If A is the amplitude envelope's array, Seppänen then searched the envelope for positions i where the quotient $\frac{A_{i+1}-A_i}{A_i}$ exceeded a specified threshold.

Detect movements by looking for acceleration change

For accelerometer signals, approaches evaluating frequency content appear not suited: in the audio case they make sense by highlighting changes in timbre or cadence, but these have no obvious counterpart in human movement. On the other hand, the computation of an envelope might be omitted due to the same lack of a carrier frequency. The notion of extracting onsets by searching for at sudden increases in volume could then be directly copied as a search for sudden increases in acceleration.

3.2.3 Processing of Audio Recordings

One possibility of finding patterns in sampled audio data is not trying to convert it into a symbolic representation in the first place, but doing the analysis directly on the level of the individual samples as recorded by the microphone. In essence, this is what Guedes did with his data on the "quantity of motion" between video frames. Besides filtering out all potentially interesting frequencies using band-pass filters and then surveying each frequency band's 'energy', there are various other possibilities.

Comb Filters

Just as Brown transformed a musical score into a signal which she could then auto-correlate to find prominent lag times, audio recordings themselves can be analysed for repeating patterns. Besides auto-correlation, one method is to apply *comb filters*. Put plainly, they tell whether shifted echoes of a signal strengthen or dampen it. Testing various shifting intervals, that with the strongest emphasising effect can be considered the most fundamental. A formalization is given in Appendix A.

Scheirer [Sch98] used comb-filters to find an audio recording's tempo. He divided the recorded signal into frequency bands to approximate a separate analysis of the musical instruments heard on the recording. Then he computed the derivative of each band's amplitude envelope (discarding negative values) and (independently, not successively) applied comb filters to it, one for every candidate tempo to be tested.

Now having filtered signals for each combination of frequency band and comb filter, these signals' energy (the sum of the squared sample values) was computed. Summing energies across frequency bands according to the comb filter used then yielded a score for all those filters. The 'echo interval' of the comb filter with the highest score could then be considered the audio recording's beat interval.

In a sense, this method goes almost all the way to deriving a symbolic representation, but stops just before and instead applies the said comb filter to find regularities in the signal.

Comb filters can test intervals for being the beat interval

As mentioned in Section 3.2.2, an accelerometer's signal at sampling rates as low as those used for this thesis should not require enveloping; a comb filter could thus immediately be applied.

Auto-Correlation

Brown created a kind of amplitude signal from her given series of note onsets to then auto-correlate it. Cuadra et al. [CMS01] used auto-correlation for pitch detection on raw audio signals. A combination of two approaches would be to compute auto-correlation on the amplitude envelope of the audio signal itself. This should be oblivious to tonal vibrations and emphasize correlations on a rhythmic scale.

Longer-term periodicities should be ignored by setting the maximal lag accordingly, thereby also increasing efficiency. To do this in a real-time setting, it would be required to analyse windows of samples over the past few seconds, thus limiting the maximal lag time, anyway. If large deviations from one such window to the next can be excluded, local searches around previously dominant lag times can be employed, as done by Cuadra et al.

Use high-level auto-correlation to determine metric quotient

Gouyon and Herrera [GH03] propose a higher-level auto-correlation to derive information on a piece's metric structure. Assuming that the beat interval is already known, they divide the recording into segments centered around the beat and do an auto-correlation on properties of these segments. Considering that most (Western) rhythms' measure level can be subdivided into two or three units of the next finer level, the authors limit auto-correlation to only consider a lag of up to eight. High values for lags two, four and eight are interpreted as hinting on a division of the measure level in two, high values for lags three and six as hinting on a division in three.

Both the low- and the high-level auto-correlation approach do not make use of audio-specific properties and should thus be applicable to accelerometer data, as well.

Fourier and Wavelet Transformations

Before concentrating on wavelets, as described in Section 3.2.1, Smith discusses the suitability of the Fourier transformation for the derivation of metric information from MIDI recordings. As he actually worked with the audio representation of MIDI recordings, his efforts were hampered by artifacts introduced by windowing and complex carrier-frequency patterns.

In general, both kinds of analyses are certainly applicable to both audio recordings and accelerometer data. They would provide strong hints on the signal's dominant frequency. In addition to auto-correlation, Cuadra et al. also present a supplemented version of the Fourier transformation, computing what they call the *Harmonic Product Spectrum*. As shown in Fig. 3.7, downsampling the frequency-domain signal by integer values

and adding the resulting smaller arrays at the main array's beginning yields a clearer indication of the main frequency.

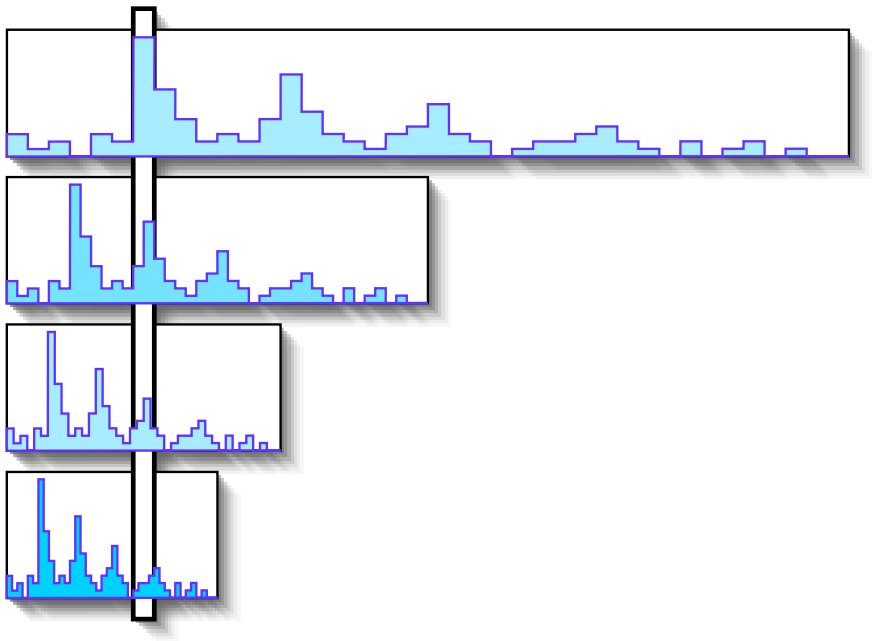

Figure 3.7: Computation of an Harmonic Product Spectrum (HPS). To compute a signal's HPS, its frequency spectrum is computed and then downsampled by integer factors. Instead of showing the spectrum resampled, this diagram merely shows it squeezed to make the shorter copies recognisable as such. The original spectrum is shown on the top, versions squeezed by the factors two, three and four are shown below. The bar that stretches across all of them indicates the position of the respective signal's main frequency and shows that its position also features bulges in the smaller copies. Adding all copies up, this frequency would thus be emphasized even more by the help of its multiples. Those multiples are part of the signal because a regular series of impulses does not only feature intervals between successive impulses, but also such with a multiple of this interval's length.

Consider frequency analyses complementary to impulse analyses

But, concerning the central intent of this thesis of deriving a representation of the underlying rhythm, raw signal analyses alone like the Fourier or wavelet transformations can only take a complementary role. Even if they were used to derive both measure length and beat interval, the inference of the rhythm's constituent conceptual impulses (in their relative position and accentuation) appears hardly feasible. Results of frequency analyses may, however, contribute to reliability and short-term stability of any output derived using impulse-oriented methods.

3.3 Intended Contribution

Contribution: application of rhythm extraction to accelerometer data

There are two main aspects in which this thesis goes beyond the related works presented. The first is the application of rhythmic extraction to motion data. Among those dealing with dance inputs, only Guedes was interested in the extraction of 'defensible' rhythmic information, instead deriving representations that seemed fit for their current (mostly artistic) intent.

The other projects dealing with the extraction of tempo and metric information did not deal with motion data at all. The added difficulty posed by the accelerometer setting stems from the absence of harmonic or other tonal hints, also from the imprecision and complexity of even professional dancers' movements compared with the (professional) playing of instruments. Even bare finger tapping by a layman by far lacks the precision that can be expected from a musical recording.

Contribution: extracting whole rhythmic patterns

The second aspect is the transformation into a representation of rhythm itself. Most of the mentioned works concentrate on finding the tempo or limit other output to low-level information like, in Guedes' case, the tempo's harmonics. Smith's rhythmic diagrams, in turn, do contain a wealth of information, but do not fit de Jong's requirements for real-time applicability and the required post-processing.

Especially considering the aimed-at representation of rhythm as a pattern of impulses, it appears fitting to also determine such a rhythm's structural parameters, measure length and beat interval, by a method that deals with individual impulses. The algorithms described in Chapter 4 will thus be modelled around the notion of III histograms, but complemented by Fourier or auto-correlation analyses in order not to rely on the impulse detection algorithm too much.

4 Algorithmic Design

Among the required steps listed in Section 1.2, the acquisition of data and the derived rhythm's graphical output arouse purely implementational questions and are thus deferred to Chapter 5, *Implementation*. This one focuses on the analysis of a certain time window's accelerometer data with respect to rhythmic properties.

Remembering that rhythm means a "recurring series of accentuations of and intervals between impulses", the task at hand may be taken on by

- either looking for recurrences in general and then for common impulse structures among these,
- or looking for impulses and trying to find recurring patterns in their sequence.

Keeping in mind the approaches described in Chapter 3, the former could start by applying standard scientific methods like the Fourier or autocorrelation transformations to find out which frequencies dominate the sensor data and hopefully get an indication of the underlying measure length. Then, however, just like the latter option starts, individual impulses would have to be identified to additionally determine the rhythm's internal structure.

The next section will thus treat the important question of how relevant impulses can be derived from the accelerometer data. Later, the actual rhythmic analysis is dealt with, both with and without using results of raw-data frequency analyses.

4.1 Movement Detection

Assuming that whenever the acceleration sensors produce a new reading, they immediately supply this as input to the algorithm for the detection of movements, the latter can keep a history of those sensor *samples* that arrived within a certain time window in the past, or keep track of quintessential information about them.

4.1.1 Simple Movements

To find out what information to keep, a sensible thing to do is exemplarily analysing the graph of the acceleration values reported in the course of simple (e.g., up-and-down) movements of the sensor. In the following, such observations are made on only one axis of only one of the sensors, a 'sample' thus being a scalar value until further notice. To maximize the sensor's response, it is moved along this measured axis as closely as possible. Once conclusions are drawn, they have to be reconsidered to also cover arbitrary movements.

Fig. 4.1 shows the graph corresponding to a mere drop of the sensor. Its shape of two oppositely-signed successive bulges seems to make sense: initially at rest (zero acceleration), the sensor first accelerates and then decelerates to finally come to rest, again. As could be expected, the time at which the attained velocity is maximal (the zero-crossing just before deceleration starts) is located near the middle of the movement.

A down-and-up movement is shown in Fig. 4.2: there are now three bulges. Again, this is not surprising, considering what a single downward and a single upward movement would have looked like. If they,

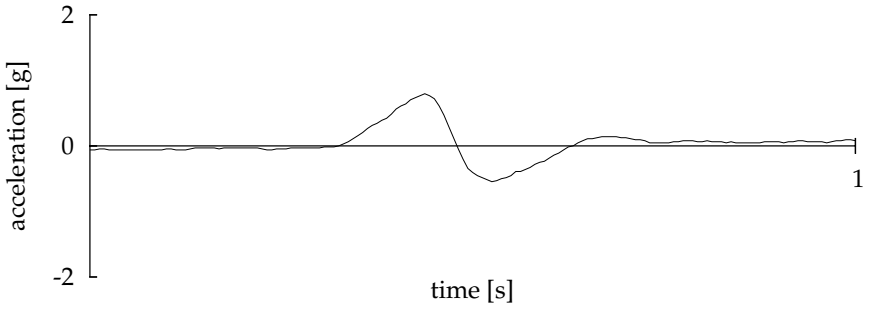

Figure 4.1: Acceleration graph of a sudden drop of the sensor.

as the case with the graphed movement, occur in immediate succession, the downward deceleration merges with the upward acceleration, which, of course, have the same direction.

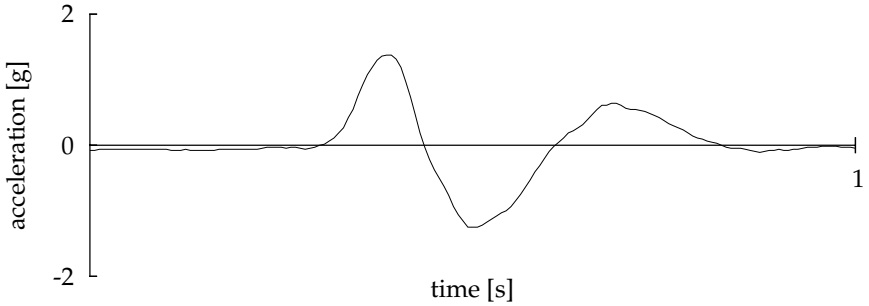

Figure 4.2: Acceleration graph of a down-and-up movement.

A sequence of continuous up-and-down movements will thus yield the alternation of positive and negative bulges seen in Fig. 4.3.

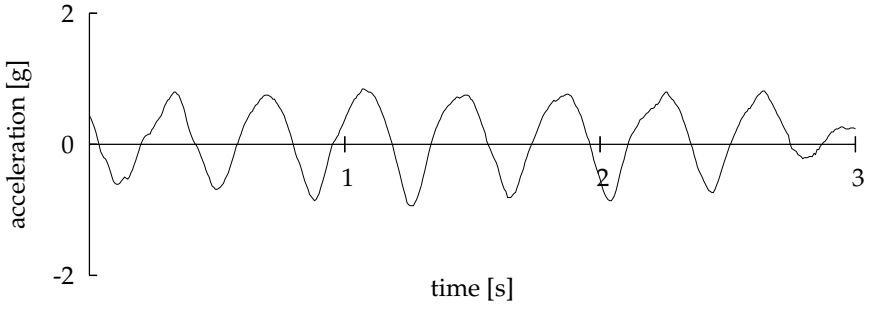

Figure 4.3: Acceleration graph of a continuous up-and-down movement.

Detect bulges in sensor graphs

Of course, the precise shape of the bulges may deviate from the sine-like appearance pictured, but the alternation of bulges as such is a common and explainable feature. They appear a useful feature to look for when attempting to detect singular movements. It should be kept in mind, though, that they occur in groups, so ways must be found not to count all of them separately.

Varying Accentuation

Moving the sensors in a waltz fashion, i.e., moving in a constant interval and accented according to the sequence *strong–weak–weak*, results in a graph (Fig. 4.4) that indeed displays this accentuation. Fig. 4.5 also shows the accentuation, but as the sensor has been kept at rest in-between movements, the resulting graph is not reminiscent of a sine, but a mere succession of triple-bulge groups of the 'down-and-up' kind of Fig. 4.2.

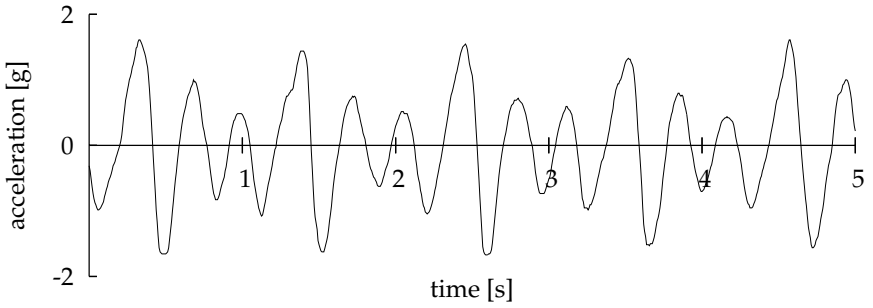

Figure 4.4: Acceleration graph of a continuous waltz-like movement of one sensor.

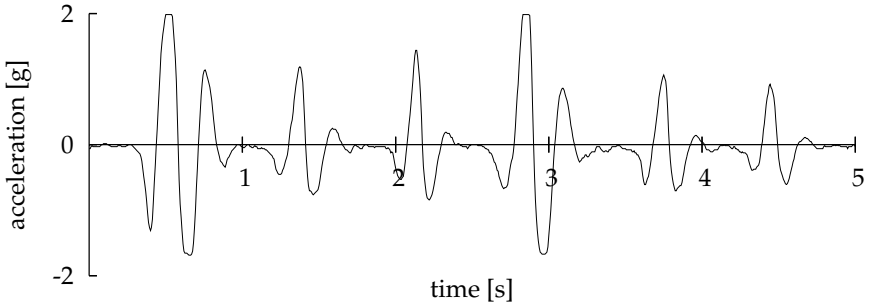

Figure 4.5: Acceleration graph of a staccato waltz-like movement of one sensor.

Concerning the magnitude attribute of detected bulges, their maximal

amplitude or the 'area' covered appear to be suitable quantities to evaluate.

Consider only pairs of bulges

The difference between these two waltzes shows that one cannot simply assume an n:1-relationship between graph bulges and 'movements with rhythmic significance'. From the study of simple movements, it can be seen that a singular motion is characterized by a group of two or three bulges. The third bulge occurs in case the motion is stopped, otherwise the graph transitions into the next motion's group of bulges. One apparent feature of the third bulge is that it has no successor on the other side of the zero line. So when detecting bulges, one might only consider pairs of them, discarding trailing ones by interpreting their lack of a partner as a sign of being a mere aftershock of the main movement.

4.1.2 Dance Movements

Disproportionately more complex patterns can arise when looking at data that is not measured at a hand moving in a precisely choreographed way, but at limbs' movements as they occur even in a standardized dance. Fig. 4.6 graphs the sideways acceleration of a *Cha Cha* dancer's right and left legs' shins. The larger scale chosen shows the repetitiveness as well as the groups of two movements being repeated. On the left leg, the sensor apparently had been affixed with some more slack, resulting in the addition of a vibration to the bulges resembling the steps.

A different dance, *Samba*, is to exemplify the differences between the measurements taken from one sensor's two axes: Fig. 4.7 shows the data from the dancer's left leg's shin sideways and along the leg. The latter's graph features clearer bulges because Samba steps emphasize the forward or backward directions. Sideways movement is less pronounced, leaving the other graph with a quite noisy appearance.

Especially in more 'chaotic' dances, it becomes rather futile to attempt an alignment of the sensor axes with the conceptual 'main axes' of a dance's movements. Fig. 4.8 shows all sensors' data from all axes of such a dance. The dancer's movements affect the sensors' axes simultaneously, which in addition to the more 'liberal' style of dance itself leads to a lack of clear series of bulges. A periodicity, especially one in line with the underlying music's beat, can also hardly be seen.

The question results how to proceed under these circumstances. Frequency analyses may detect regularities in seemingly noisy data, but the representation as a rhythm that this thesis aims at cannot be derived

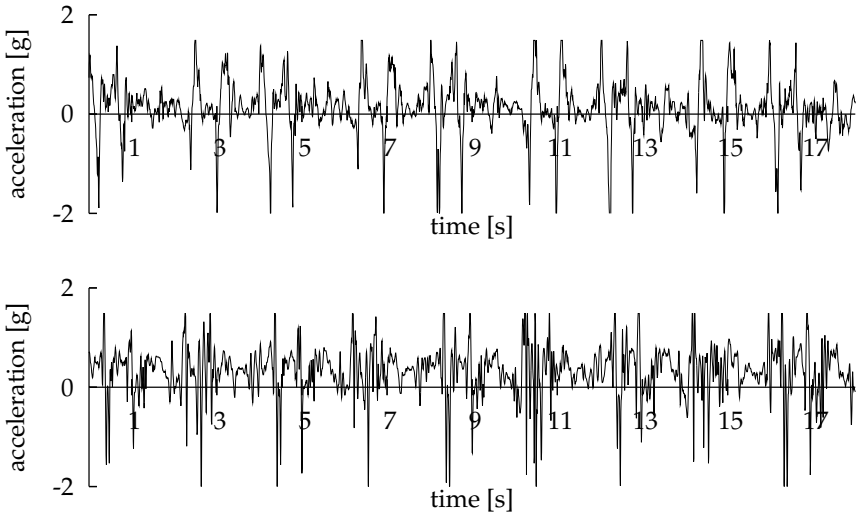

Figure 4.6: Acceleration graphs of the shins (that of the right leg is the top one) of a dancer moving to a *Cha Cha* rhythm. The main movements are not equidistant because the dancer alternated step sequences to the left and to the right, so that for some seconds the right leg moved first, for the next few the left, etc. The left shin's sensor (bottom graph) was apparently not fixed as tightly as that of the right, leading to vibrations in addition to the step movements themselves.

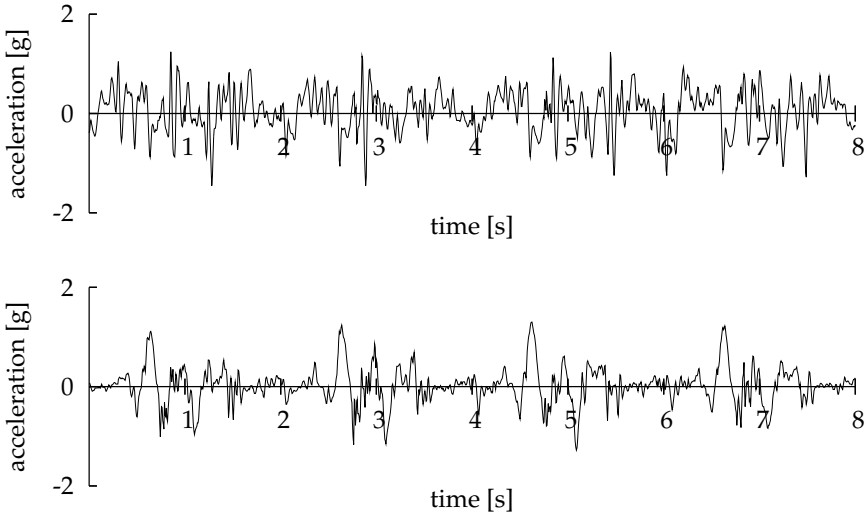

Figure 4.7: Acceleration graphs of sideways (top graph) and longitudinal (bottom graph) leg movement in a *Samba* dance. As typical for the dancer's movement, the longitudinal movement is more pronounced and thus results in a graph with clearer structures.

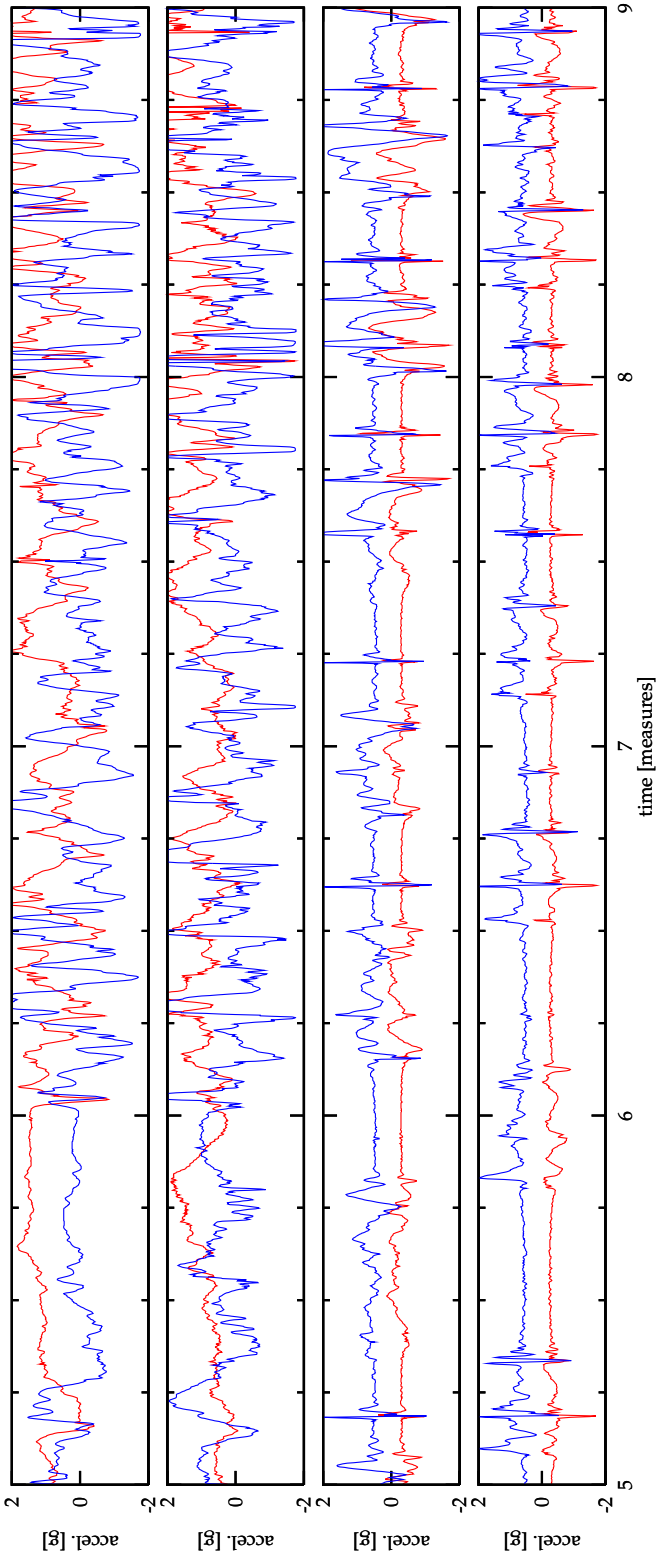

Figure 4.8: Acceleration graphs of all limbs of a dancer moving to the piece *Rhythm Is A Dancer* (by *Snap*, 1992). From top to bottom, the graphs belong to the right forearm, left forearm, right leg and left leg. The blue lines represent the longitudinal axis, the red ones the sideways one. The time axis is subdivided into the underlying music's measures: the labels indicate the number of measures completed at that position, the smaller ticks its beat. The tempo of 124.3 bpm makes each measure about 3.9 s long. At the beginning of the ninth measure (at label "8"), the piece's vocal and base-drum part begins, resulting in increased activity of the legs (bottom graphs). Perhaps the dancer anticipated this, which could explain the change in his arms' movement pattern at the end of measure six.

from a mere such analysis. Movements have to be detected, also in noisy data.

Experiments with variations in the positioning of the sensors affirm that this may be a parameter useful for optimization: in case the dance is very 'vivid', the sensors may be placed where movements are low-frequent, e.g., at the head. In how far such an optimization is possible in an application scenario like one of those proposed by de Jong for the Sensor Music Project, is outside the scope of this thesis. The algorithm presented will work on a best-effort basis, trying to also in noisy data detect the, as put above, 'movements of rhythmic significance'.

Try mastering complexity with heuristics

The latter term may prove hard to define when not dealing with movements that result from the adherence to rules like "Cleanly move the sensor down and up again for each perceived impulse!". Accounting for all differences in dances, personal styles and sensor positions appears hardly possible. Theoretically, a movement should be called 'important' simply when its consideration benefits the ensuing rhythmic analysis. Practically, certain heuristic rules will have to suffice that guess this importance.

4.1.3 Criteria for Movements

The shown graphs seem to indicate that despite the lack of an n:1-relationship between bulges and movements, the former are the most prominent indicator of the latter. Assuming that at rest, the signal is at zero, a bulge can be defined as the part of the graph between successive zero-crossings. If that is not the case, the signal can be offset by subtracting the average of a number of recent sample values. If this number is chosen greater than the interval expected for movements to be detected, the sample values will oscillate around zero without seriously spoiling the signal's frequency composition or individual bulges.

But, as could also be seen in the acceleration graphs of actual dances, there are far too many zero-crossings and thus too many bulges for them to all be important indicators of the intended rhythmic structure. Hence, criteria have to be determined for such a bulge to be relevant.

Judge relevance by strength, duration and context

Various phenomena may add noise to the sensor signal: involuntary twitches, interference of various limbs' and other body parts' movements or the influence of Earth's gravity. To distinguish important movements from those, it seems that the bulges' size should be considered, three prudent criteria being

- area,
- absolute amplitude and
- amplitude relative to the current level of activity.

The "level of activity" can be the average or median of recent sensor samples' absolute values, the relative amplitude the quotient of absolute amplitude divided by said activity level. Formally, given $s_1, \ldots, s_n$ as the n most recent samples, a bulge of absolute magnitude m has the following relative amplitudes:

$$m_{\text{rel\_average}} = m \cdot \frac{n}{\sum_{i=1}^{n} |s_i|}$$

$$m_{\text{rel\_median}} = m \cdot \frac{1}{\text{median}(\{|s_1|, \ldots, |s_n|\})}$$

Such criteria will make minor movements count during very static periods, but omit them in case so much is going on that any sensible demand on a mere absolute amplitude would yield 'important' movements all the time.

Determination of a bulge's area, i.e., integration of the signal, will briefly be discussed in the next section.

Another standard approach to feature detection is the differentiation of the given signal, as suggested in Section 3.2.2 when discussing ways to extract symbolic representations from audio recordings. The bulges' peaks, for example, could be found by looking for zero crossings in the signal's first derivative. Their inclination, on the other hand, is maximal where that derivative has a local maximum. As the numerous steep inclinations in the more dynamic parts of the graphs shown in Fig. 4.8 show, differences of successive sample values carry little meaning. A low-pass filter would have to be applied and differences considered between values that are several samples apart.

4.1.4 Properties of Movements

In order to supply analytic algorithms with information on recent major movements, they have to be assigned certain properties: they take place over a certain period of and in time and have a magnitude. Remembering the comments to the definition of *impulse*, they can alternatively be attributed

- a point in time,
- a magnitude and
- a spread value mirroring the first attribute's imprecision.

Point in Time

Movements' timing is ambiguous

The first question arising when 'time-stamping' a movement is whether to consider its beginning, its end, or some time in-between as its defining moment. Thinking of a drummer or someone clapping his hands, the movement's end obviously is its climax, yet in the context of a ballroom dance, it seems far from clear that the most climatic sounds played by the orchestra should always coincide with the ends of swings and turns. Then again, for the extraction of rhythmic patterns their constituent impulses' absolute point in time is not important, anyway, so with the fear of imprecision caused by artefacts at bulges' rims and in the hope that such statistically cancel each other out in-between, some prominent feature in a movement's 'interior' should be used.

Looking at one pair of opposite bulges as deemed typical of an important movement, the zero-crossing between the two appears like a natural temporal anchor, yielding the required point in time. However, graphs like that of Fig. 4.2 show that the numerically positive and negative bulges characteristically differ, at least in the case of an up-and-down movement performed with a hand. Instead of amalgamating the two bulges to one movement, it may thus be preferable to record and analyse positive and negative bulges separately. In that case, either the respective bulge's maximum or the middle between its two flanking zero-crossings are candidate points in time. Considering a movement's point of maximum acceleration as more defining than its mere temporal centre, the former should be chosen.

Magnitude

Aspects of motion intensity

Naturally fitting the choice of the maximal amplitude's point in time would be to use that very amplitude value as the movement's magnitude. Yet tests showed a movement's intended accentuation to also be mirrored in its duration, and the sensor data's integral (the bulge's area) turned out to be suited better for rhythmic analysis. The prime benefit seems to be that spikes in the data graph, caused by a collision of the sensor or the moving limb it is attached to, would count less.

Remembering that a bulge's relative amplitude (amplitude divided by activity level) was previously named as indicative of its significance when deciding whether or not to consider that bulge at all, it appears another logical choice as embodiment of a movement's magnitude. However, in the case of hardly moving sensors, small bulges would get assigned a very large amplitude, as the divisor approaches zero. In subsequent sections, a sensor's bulges' combined magnitude will play a role, so in order not to require a separate count of magnitudes without relative factors, they are left out completely, for now.

This discussion brings up the question why, if a bulge's area is a useful indicator of magnitude, it should not also have an influence on the decision whether to consider it at all. It would benefit long, but shallow, bulges. These are often due to changes in the sensors' orientation in relation to Earth's gravity field, as experience shows. They are even 'artificially produced' by the previously mentioned offsetting technique that is required for the detection of bulges, so area should only be an additional criterion.

Spread

Record timing's uncertainty

Taking a movement's point of maximum acceleration as its defining moment, the previous and successive points in time where acceleration is zero are the natural borders of that movement. The uncertainty in said choice of 'defining moment' thus has to be a function of those two other points in time, for example their difference. How exactly this function should look like depends on how it will be used later on, but as a rule of thumb, its result should usually remain clearly below any interval expected to be met between movements in the analysis.

4.1.5 Separating Directions of Acceleration

In Section 4.1.1, it was noted that singular movements' acceleration graphs are characterized by either

- an acceleration and a deceleration bulge, or
- an acceleration, a counter-acceleration and a deceleration bulge.

The consequence was to consider pairs of them as one movement and to discard bulges with neither counter-acceleration nor deceleration partners.

The ensuing discussion of movements properties and which features of bulges use as indicators for those properties, however, only considered single bulges. On possibility to resolve this apparent mismatch would be to simply discard all bulges of negative acceleration. But, as the simple movements' graphs show, there is no perfect symmetry between 'positive' and 'negative' bulges. At least when moving the sensor by hand, acceleration and deceleration do not necessarily mirror each other. Discussing or testing which side of the graph might generally yield more useful data makes no sense because acceleration and deceleration can switch signs anytime, depending on the orientation of the sensor relative to the movement.

One way of keeping all bulges' information is to analyse both sides of the graph separately, feeding the detection results to rhythmic analysis as if they originated at different sensors. Rhythmic analysis could additionally be told about these circumstances and improve its output quality via that knowledge.

A variant of this approach would be to distinguish not by sign of the bulge, but by the temporal order of the two bulges considered a pair. On first sight, this would actually be more correct, thinking of cases like the direction-wise alternating sequence of separate movements shown in Fig. 4.9: an analysis of the positive and negative bulges on their respective own would detect an alternation in the length of intervals between successive bulges. Intuitively, though, the movements would be called temporally equidistant.

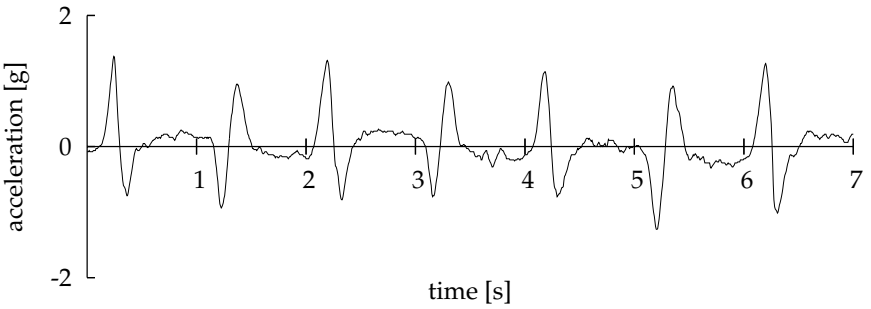

Figure 4.9: Accleration graph of a direction-wise alternating sequence of separate movements. Computing the intervals between the graphs positive peaks, they turn out to be varying, as do the negative ones. To the producer of the movement sequence, in contrast, his movements followed a steady beat, merely alternating in direction.

Acceleration and deceleration are hard to distinguish

Should, on the other hand, a bulge 'erroneously' occur ahead of a sequence of actually significant movements, it may change the perceived ordering of successive pairs: a sequence of signs "+ − + −" would, prepending a negative bulge, result in "− + − + (−)", the final bulge being discarded as it lacks a partner. Classification of bulges as representing an acceleration or a deceleration is thus risky.

Antedating the resulting algorithm's final evaluation, 'false positives' in the bulge detection are admittedly more common than directionally alternating staccato movements like those shown in Fig. 4.9 can be expected to occur in an average dance movement. Thus, the seemingly more naive sign-oriented approach of recording bulges is preferred.

In retrospect, this also means that relative amplitudes can be counted separately for positive and negative bulges. Again given $s_1, \ldots, s_n$ as the recent samples to consider, setting $P = \{s_i | 1 \leq i \leq n \wedge s_i > 0\}$ (for

the positive case) yields:

$$m_{\text{rel\_average}} = m \cdot \frac{n}{\sum_{s \in P} s}$$

$$m_{\text{rel\_median}} = m \cdot \frac{1}{\text{median}(P)}.$$

Another possibility is the merging of both bulges to some common properties. Then, the zero-crossing would be the prime choice as point in time and the bulges' magnitudes could be averaged or the difference between positive and negative peaks used. The time difference between those peaks then forms a natural base for the spread value, but could also be chosen as a function of the time between the start of the first and the end of the second bulge.

4.1.6 Combining Axes of Measurement

Instead of considering the samples measured by the sensors on a per-axis basis, they could be combined to one signal, in which then bulge detection could take place. Combining three dimensions' sample values s_x, s_y and s_z, one might derive the physically meaningful value of absolute acceleration as

$$\text{acc}_{\text{abs}} = \sqrt{s_x^2 + s_y^2 + s_z^2}.$$

In this thesis' setting, however, only that value's projection into the plane spanned by the two available axes could be computed.

Treat axes separately

Considering that movements into the two measured directions may indeed differ for rhythmic reasons, is seems prudent to evaluate both axes' signals separately. A common value may improve detection stability, primarily in the case of dances in which significant movements can be expected to occur in any direction at any time.

4.1.7 Summary

The analysis of simple movements' accelerometer graphs, together with a look at likely problems when dealing with more complicated data, makes it appear sensible to, for each sensor axis,

- detect peaks and zero-crossings to derive bulges,
- consider two successive bulges of opposite signs a significant movement and

- for rhythmic analysis, output the bulges' properties in separate 'output channels' or, after combining them to one impulse, in a common channel.

Abstracting, in pseudo-code, from the actual data structures managing the information on positive and negative bulges, Alg. 1 summarizes movement detection.

Algorithm 1 Detecting movements from a stream of acceleration measurements

```
while there is an unread sensor sample do
  if any stored bulges have become too old then
    delete their information
  end if
  if current sample value has a different sign than the previous then
    store current time as previous bulge's end
    if both signs' most recent bulges were sufficiently large then
      output them as one (or two) impulse(s)
      delete all information on both bulges
    end if
    store the current time as start of a new bulge
    if the current sample's absolute value is 'large' and exceeds the
    current bulge's record then
      store it as the current bulge's new maximum amplitude
      mark the current bulge as having been of sufficient size
    end if
    increment the current bulge's area by the current sample value
  end if
  store the current sample value as the previous
end while
```

The mentioned impulse tuple for each detected bulge consists of three values:

- magnitude = area $\times$ peak amplitude
- time stamp = average (starting time, peaking time)
- spread = starting time $-$ ending time

Whether to output positive and negative bulges separately is left as an option, also in the implementation. When outputting only one impulse per pair of bulges, then the time stamp could rather be chosen as the bulge's connecting point in time. The spread might add both bulges' extent, the magnitude could have the combined area and added amplitudes as its factors.

Less than having been reduced to pseudo-code, for the sake of clarity the above description does not even mention some additional tasks required. Among those is the management of samples' 'histories' for offsetting

the signal and computing the median. The length of these histories is a parameter discussed when evaluating the algorithms' effectiveness in Chapter 6. Similarly undefined are, as of yet,

- the absolute or relative threshold whose exceeding defines a bulge as "sufficiently large" and
- the time after which to consider a bulge as having become "too old".

More than tweaking parameters, the algorithm can be slightly altered for reasons of efficiency: as the algorithm is to be applied to all available sensors' axes, it turns out to be sensible to collect the detected impulses and only periodically output them for processing. Such aspects, though, are left for Chapter 5, *Implementation*. Before, the next section discusses how the resulting sequence of detected movements can be analysed for rhythmic patterns.

4.2 Rhythmic Analysis

Before describing the chosen approach, a short summary of the main ideas collected in Section 3.2 will help motivate it:

1. The context of undoubtedly significant movements may give hints on the significance of doubtful detections. (Section 3.2.1)
2. Candidate beat intervals can be tested using a comb filter. (Section 3.2.3)
3. Using auto-correlation, a signal's measure length and beat interval can be extracted. (Section 3.2.1)
4. Finding equidistant series of strongly accented impulses gives a selection of important intervals between them and candidates for the downbeat. (Section 3.2.1)
5. Peaks in histograms of inter-impulse intervals (III) indicate prominent such intervals. (Section 3.2.1)
6. Spectrograms of past data can be analysed for trajectories of main frequencies. (Section 3.2.1)
7. Given the beat-interval, a high-level auto-correlation can yield the metric quotient and thus the measure length. (Section 3.2.3)
8. Frequency analyses like the Fourier transformation can be used to determine the (or validate a differently derived) beat interval. (Section 3.2.3).

Central notion: III histograms

For this thesis, the creation of III histograms (5.) has been chosen as the central approach because they are based on the movement detection that seems required, anyway, in order to yield a sequence of conceptual impulses as final result. The indication of beat interval and measure length that the histograms give are, by definition, to be found in the sequence of detected movements.

Find downbeat via accentuation

For a similar reason, the detection of not only intervals between pairs of impulses, but of equidistant series (4.) appears a notion helpful for finding the downbeat.

Add frequency analyses as backup

Frequency analyses (any of 2., 3. and 8.) should be used as a check for the other methods, as they are unimpaired by potential problems in movement detection.[1]

Discarding impulses not fitting the assumed metric grid (1.) appears hard in a setting not as formalized as the keyboard performance of a musical score. Together with the image processing of spectrograms (6.) and a number of other ideas presented in Chapter 3, they will be revisited when discussing possible future work in Section 7.2.

4.2.1 Overview

Detect movements, derive their structure

A first overview of the general pipeline of algorithms to be used for rhythmic analysis is shown in Fig. 4.10. On the one hand, movements are detected in the sensor signal. Considering a time window of a certain extent, they form a sequence of impulses, the intervals between which can be analysed for hints on the signal's metric structure, i.e., deriving measure length and beat interval.

Add results of frequency analysis

On the other hand, a frequency analysis like the Fourier transformation is applied to the signal. Together with the results from the analysis of the impulse sequence's intervals, a final determination on the metric structure is made.

Fold impulses into representative measure

Now knowing the measure length, the sequence of impulses can be divided into measures and all measures' impulses superimposed. In the diagram and henceforth, this procedure is called *folding* of impulses because of its reminiscence of a strip of paper being shortened by multiply folding it onto itself.

Cluster impulses to rhythm

Once this is done, the numerous impulses get clustered to those positions within the measure that, according to the assumed metric structure, form the beat level in the metric hierarchy.

[1] Preliminary tests of a one-frequency-band version of Scheirer's comb-filter approach in Matlabshowed, however, that simple averaging of the tempi it derives from data of the given sensor package produces too erratic results.

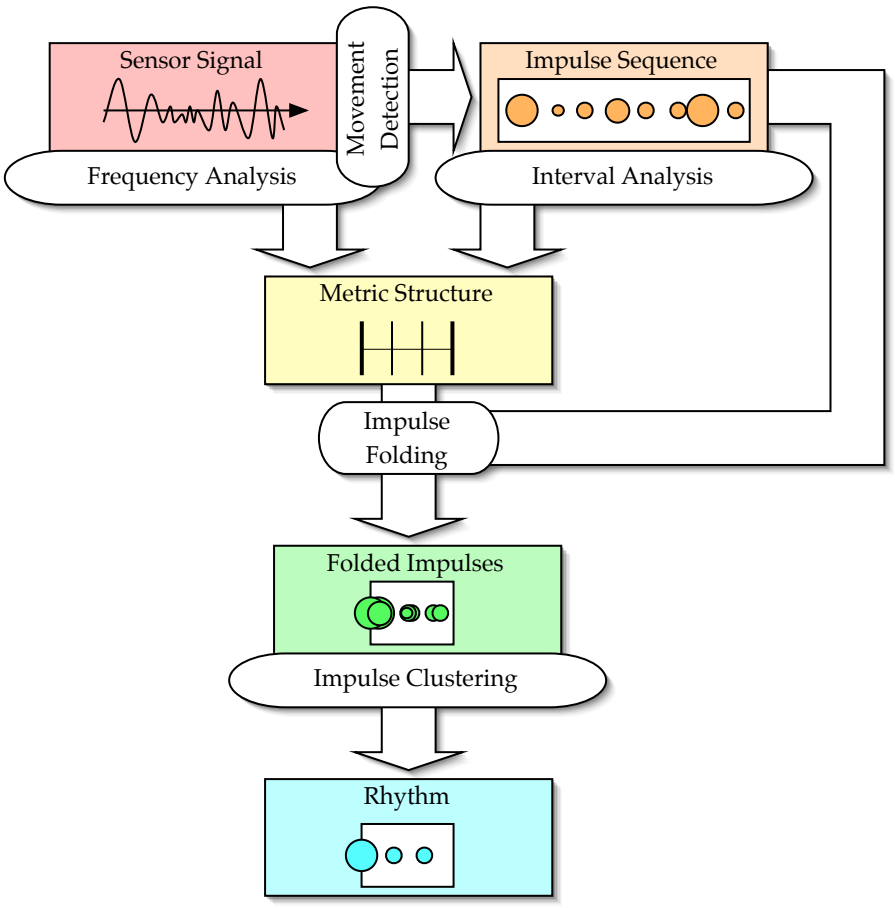

Figure 4.10: Rhythmic analysis of sensor signals. For an explanation of the diagram, please refer to the text. The colours used for the sensor data (red), detected movement impulses (orange), information on the metric structure (yellow), impulses folded into one measure (green) and the rhythm with its conceptual impulses (blue) will be recurring in future figures. So will the use of boxes with corners for data and rounded ones for processes.

The above description was to give an impression of the task at hand, but is quite imprecise. Foremost, it ignores that there are several sensor signals to evaluate and from which to derive a common rhythm. Furthermore, the movement detection algorithm described before may yield a series of impulses for each side of the graph of each sensor's both axes. In the given case of four two-dimensional sensors, there are eight axes and thus sixteen such sides. These sides or axes are called *channels* from now on whenever their origin does not matter.

An important aspect regarding the tasks described above is the level on this hierarchy that they apply to. Signal processing approaches, for

example, make no use of movement detection and can sensibly only be applied as far down as the axis level, but perhaps also operate on combined signals of both axes or even several sensors.

So one main question that arises is for how long in the algorithmic pipeline to treat (levels of) channels and sensors separately and when to combine their data.

4.2.2 Determining the Metric Structure

Fig. 4.11 answers that question for the first part of the algorithmic pipeline, the most involved in this respect. The diagram assumes there to be four sensors with two axes each, labelled X and Y. As decided, their data is fed into movement detection (MD) and (Fast-) Fourier transformation (FFT). The former exemplarily yields impulse sequences for both sides of the graph (the orange boxes labelled + and –), but, as described in Section 4.1, this will just be one option looked at. The FFTs are searched for their respective most prominent frequency, which is then output as the beat interval (BI).

III Histogramming

Leaving the FFT-determined beat intervals aside for a moment, the next important step is the analysis of all channels' impulse sequences. The aim of the analysis is to find beat intervals and measure lengths (ML).

Basically, the mutual intervals between all recently detected impulses are computed and the 'most important' of those intervals determined using a histogram. However, the histogramming technique described in Section 3.2.1 will not suffice, as it treats all added intervals alike.

Weaker impulse determines III magnitude

In Section 2.2, it was stated that an inter-impulse interval (III) could, beyond the actual time difference, be assigned a magnitude and a spread value that mirror their constituent onsets'. But no function was given how to compute them. Assuming that in a rhythmic sequence of impulses, each measure features a prominent downbeat, the pairs of impulses with large magnitudes are likely to be the ones whose mutual interval approximates the measure length. Hence, it appears plausible to choose a function that rewards such pairs. The minimum function does just that: employing it, only pairs of large magnitudes yield a large-magnitude interval. Fig. 4.12 shows this effect for the example of an equidistant sequence of impulses.

Given two impulses I_1 and I_2, their III's magnitude can now be defined as:

$$\mathrm{mag}\left(\mathrm{III}(I_1, I_2)\right) = \min\left(\mathrm{mag}(I_1), \mathrm{mag}(I_2)\right).$$

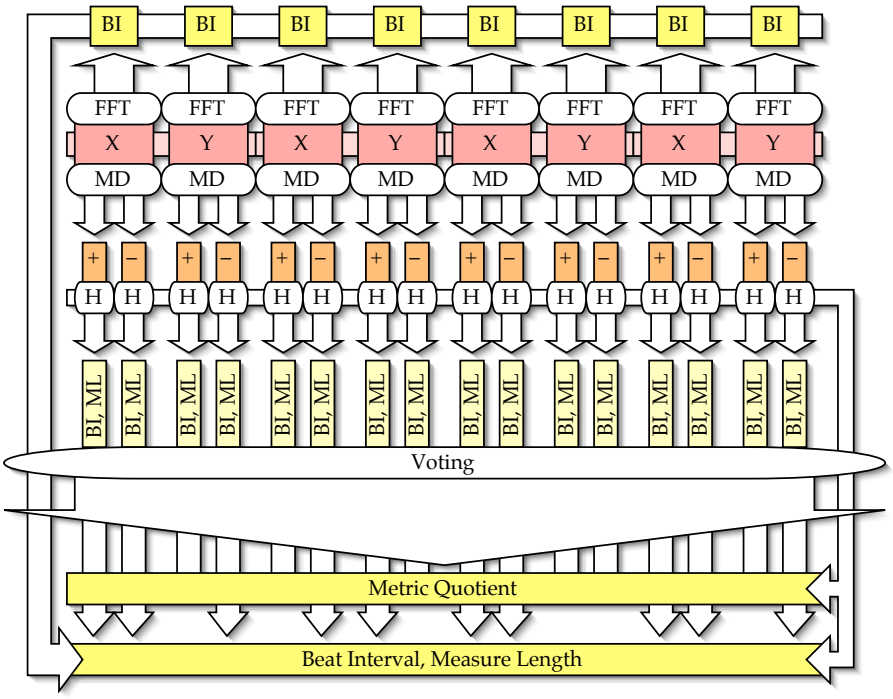

Figure 4.11: Derivation of metric information from sensor data. The diagram shows the flow of four two-dimensional accelerometers' data through various analyses to finally yield a common decision on the beat interval and measure length. The used abbreviations are: "BI" for the beat interval, "FFT" for the Fast Fourier Transformation, "MD" for movement detection, "H" for histogramming and "ML" for the measure length. For a detailed explanation of single aspects, please refer to the text.

The spread value, the uncertainty at which an interval's length is specified, could be the maximum, the sum or the average of the constituent impulses' spread values. Defining one impulse's spread as the width of its bulge(s) in the signal graph, the averaging function turns out to be a good choice that does not to blur the resulting histogram too much:

$$\text{spread}\left(\text{III}(I_1, I_2)\right) = \frac{1}{2}\left(\text{spread}(I_1) + \text{spread}(I_2)\right).$$

Fill range of buckets to mirror uncertainty

To account for magnitudes in the histogram, it suffices to add the magnitude to the correct bucket instead of merely incrementing its counter, just as done in Fig. 4.12. To consider the spread, however, neighbouring buckets must be filled as well.[2] One possibility is to practically add a

[2] Besides merely accounting for the fact that a movement's climatic point in time cannot be defined precisely, this technique makes it possible for similarly intervalled movements (by 'error' of the moving person) to support each other in terms of building peaks in the histogram.

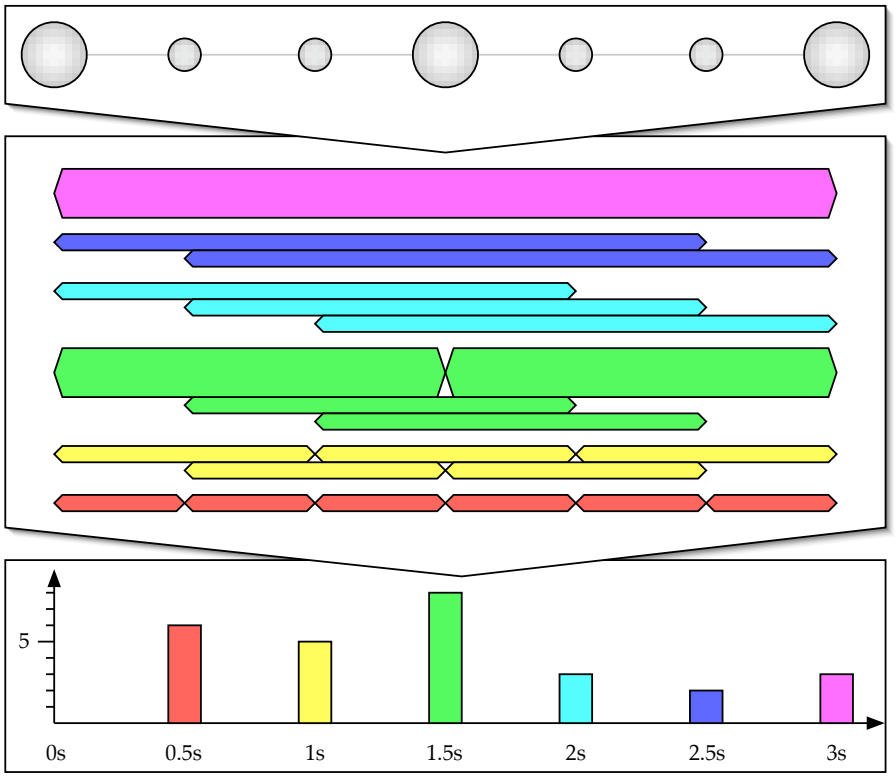

Figure 4.12: An III histogram mindful of intervals' magnitudes. Similarly to Figs. 3.3 and 3.4, a series of impulses is shown along a time axis at the top. The coloured bars represent the intervals between those impulses that their ends are situated below. In contrast to the previous figures, these bars differ in thickness, which has been chosen proportional to the minimum of the magnitudes of the impulses whose interval the bar represents. In the actual histogram at the bottom, this leads to shift in the highest peak: had the buckets only been filled according to the number of respective intervals, the red interval would have had the highest count. Here, however, the green interval comes out as the most important one. This is in line with the intention, as indeed this interval is the apparent measure length of the impulse sequence's rhythm. The beat interval can still be found by simply looking for the first (in a noisy case: significant) peak.

triangle to the histogram: the magnitude is added at the time stamp's bucket, the neighbouring ones get smaller values until, at a distance of half the spread value, contributions reach zero.

To simplify the formal definition of the buckets' values, they are indexed with the range of intervals they represent, not with a serial number. Given a tuple of impulse times $t_1, \ldots, t_n$ with their magnitudes $m_1, \ldots, m_n$ and spreads $s_1, \ldots, s_n$, the histogram bucket with centre c

has the value

$$H_c = \sum_{i=1}^{n} m_i w_i(c),$$

where the triangular shape is determined by the weight function

$$w_i(c) = \begin{cases} -\frac{1}{s_i}|t_i - c| + 1 & |t_i - c| < \frac{1}{2}s_i \\ 0 & \text{otherwise} \end{cases}.$$

Theoretically, a Gaussian shape could be used instead of a triangle, as a more mathematically justified means of mirroring uncertainty. But, considering the partial arbitrariness with which the values for time stamp, spread and magnitude could be chosen, this would hardly add any precision.

Having filled the histogram with triangles for all recent IIIs, the most important intervals can be read as the position of the highest peak and the first significant peak. As could be seen in Fig. 4.12, the former resembles the measure length and the latter the beat interval. Channels which yield very improbable values for these are excluded from the remaining analyses.

Examples of how histograms fed with real accelerometer data look will be shown in Chapter 6, *Evaluation*.

It turns out that dances' or even fingers' movements produce such erratic data that more than just the past few IIIs should be used. In order not to make the histogram respond too slowly to changes in rhythm, however, old intervals should not count as much as current ones.

To achieve this, one could soften the age restrictions on impulses considered and multiply their weight by some age-dependent value that penalises old impulses. However, Seppänens approach, described in Section 3.2.1, has a similar effect and works without filling innumerable IIIs into the histogram each time a new impulse arrives: he only fills a histogram with the intervals of a small time window, but computes a linear combination with the previous histogram, thus preserving past information. Such a histogram can be called *inert* because of its resistance to change.

The factors are chosen in order to exponentially decay the old histogram according to its age. Be H_i^j the ith bucket of the jth stored histogram, h_i^j the ith bucket of the jth newly computed histogram and t_j the jth histograms' time stamp, the next histogram to analyse and store is computed as follows:

$$H_i^{j+1} = 0.5^{c(t_{j+1}-t_j)} H_i^j + d \log_2 2 \cdot h_i^{j+1} \quad \forall i,$$

where c and d are constants that influence the rate of the old histogram's decay and the new one's initial proportion. When computing this for every new impulse, the new histogram h only needs to consider the intervals that this new impulse is constituent of, as the others are already represented in the stored histogram H.

The inert histogram is, on first creation, filled with zeroes. The discussion of data structures is deferred to Section 5.3.

Voting on the Metric Quotient

The many beat intervals and measure lengths (shown yellowish in Fig. 4.11) can differ greatly. So, too, can these values' quotient. To arrive at a common metric quotient, a histogram of such quotients is created at the beginning of all processing. Similarly to the III histogram, it is being updated by linearly combining its old contents with a new version. Fig. 4.13 shows a simple example in which the linear combination happens to be averaging.

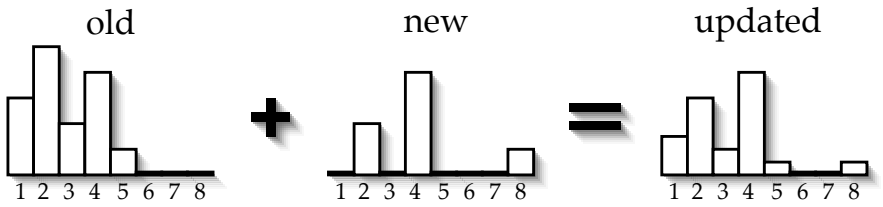

Figure 4.13: Updating an inert histogram of metric quotients. This histogram considers metric quotients up to the value eight. The one on the left is supposed to be the old histogram that has resulted from previous such updates. That in the middle is new and has been derived from III histograms' peaks. The one on the right is a combination of both, in this case a result of a linear combination with both factors being 0.5.

Favour clear rhythms

Instead of simply counting the number of channels that consider a certain metric quotient correct, their votes have a weight according to the sum of all their III histogram's buckets. As those histograms only grow when pairs of strong and thus clear impulses are detected, their sum is interpreted as an indication of the channel's reliability in rhythm matters.

Having mixed the new counts with the old histogram, the metric quotient whose bucket contains the most weight is taken as this iteration's official figure.

Voting on the Beat Interval

Next, a beat interval needs to be determined. Whereas among metric quotients only a few are relevant, and they are so distinct that an inert histogram can be set up, this is not the case with beat intervals. Here, a single value shall be inert, being initialized to a value in a suitable range and being applied changes in every iteration, i.e., whenever a rhythm is to be derived.

While the updating procedure may be similar or identical to that for the buckets of the metric quotients' histogram, the question remains how to derive the new beat interval, a common value representing all channels.

Homogenous group of channels decides

One possibility is to find groups of channels that can agree on a common denominator and then choose as beat interval the most important group's compromise. A sensible measure of agreement between two channels is the quotient of their beat intervals: if one is close to an integer multiple of the other, then a certain relatedness can be assumed. Given two beat intervals $b_1 > b_2$, one might demand that

$$\left| \frac{b_1}{b_2} - \text{round}\left(\frac{b_1}{b_2}\right) \right| < 0.2$$

for their respective channels to support each other.

For each channel i, all others can be checked for agreement. Has a fitting channel j been found, its beat interval b_j can be adapted to b_i by removing the integer quotient between the two. For example, if $b_i > b_j$, the adapted beat interval is

$$b_{ij} = b_j \cdot \text{round}\left(\frac{b_i}{b_j}\right).$$

In order not to let a group of incidentally agreeing noisy channels give a verdict detrimental to the whole procedure, not the most numerous group should be chosen, but that whose combined 'rhythmic clarity' is maximal. An indication of the latter is a channel's III histogram's sum of all buckets' contents. This is because these histograms get filled the more, the more regular their respective channels' impulses occur and the larger their magnitudes are.

Computing each group's (they may not be mutually exclusive) combined such *histogram weight*, the most significant group can be determined. Also, among the group, the different adapted beat intervals can be weighted according to the mentioned measure when agreeing on a common value.

Another approach is to use a frequency analysis method on the sensor signal itself. For example, computing a channel's Fourier transform

allows finding the beat interval by determining the smallest significantly peaking frequency. Doing this for all channels, the histogram weights can, again, be used to give different emphases. An advantage of signal analysis (auto-correlation would be a different possibility) is that it is not prone to errors stemming from false movement detection.

Like histograms, Fourier transforms can be mixed to an inert version. In comparison to a widening of the underlying analysis window, this is not only more efficient, but also sensibly emphasizes more recent parts of the signal.

Whichever way is chosen to get a current and representative value of the beat interval it gets, as mentioned above, mixed to an inert version. Consequently, the result of this mix, multiplied by the metric quotient is the official measure length.

This explains the aspects of Fig. 4.11 that had not yet been covered: the voting on the metric quotient, beat interval and measure length (the latter two only by some channels), the respective influence of the histogram weights and the possibility of extracting the beat interval from the individual FFTs' information.

One word of caution: using inert values can lead to quite wrong intermediate results: switching from a long to a short beat interval, its inert version will continually decrease and may, during that process, jump from one metric quotient to another. However, this appears preferrable to having erratic jumps in the output that could indeed be justified with some singular detected movements.

4.2.3 Deriving the Rhythm

Looking back at Fig. 4.10, what is still missing to a rhythm is the folding of impulses into a measure and their clustering into conceptual impulses, as shown in Fig. 4.14.

Folding of Impulses

As impulses were derived for each channel, be it one or two per sensor axis, the question arises which impulses to fold into which measure lengths and with which offset. On the one hand, all channels' impulses (or, better, only those that formed the winning group in the vote on the beat interval) could be combined into one sequence, which is then folded into windows with a measure length's width. On the other, they could be folded separately and then combined.

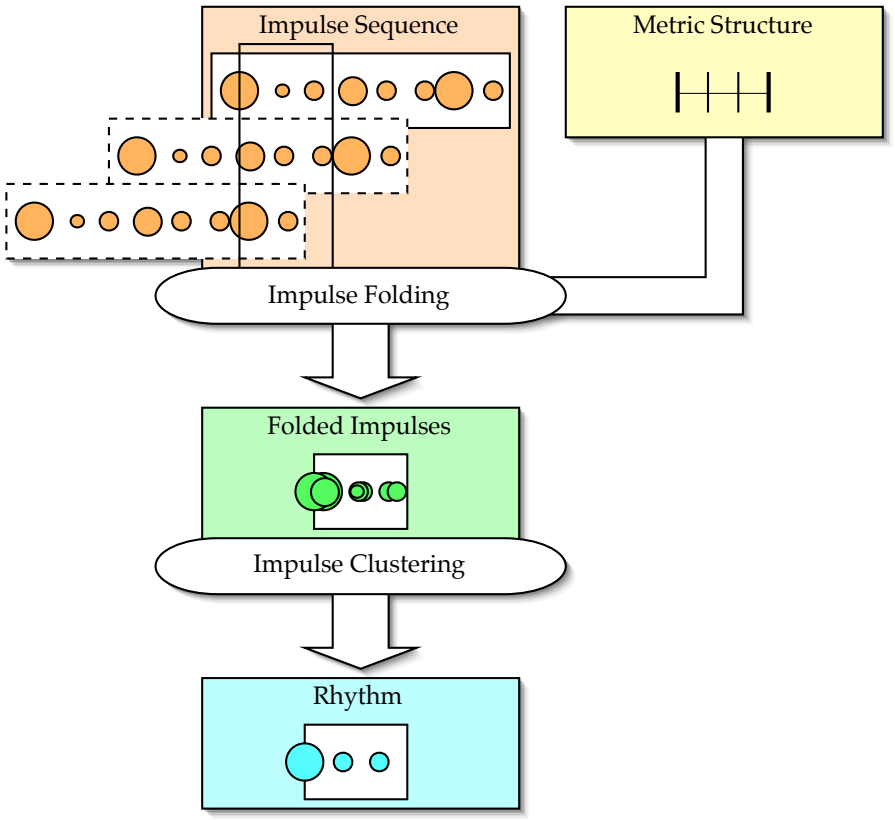

Figure 4.14: The derivation of a rhythm, given an impulse sequence and its metric structure. Knowing the metric structure, the impulses can be folded into a time window of a measure's length and then clustered to yield a rhythm. The actual folding can be imagined as visualised on the top left. There, an impulse sequence (orange) is shown alongside copies (with a dashed border), each shifted by a measure's length to the left. Starting at the original sequence's leftmost impulse, a rectangle extending vertically over the copies encloses the onsets that get superimposed in the folding procedure. As not each successive measure in the example is exactly identical, the resulting measure with all impulses (green) shows the impulses not perfectly aligned. However, after clustering to the supposed positions in the metric hierarchy (yellow), a waltz rhythm is clearly visible (blue).

Which of these orderings is better and when they make a difference at all, depends on how the actual folding takes place. For each of the two, a fitting folding variant will now be described.

Fold all channels' impulses together…

After having folded all impulses into one sequence, there is no way of accounting for channels' different de-facto beat intervals, or counteract-

ing phase differences.[3] So what remains is to get the sequence with a length of, for example, 3.5 measures, into one measure's window. This is possible by subtracting multiples of the measure lengths from the impulses time stamps, effectively executing a modulus operation.

Considering the meanwhile familiar picture of a rhythmic pattern starting with its most significant impulse, the downbeat, a re-arrangement seems to be in order. Dividing the measure length into a number of sub-intervals, these intervals' respective counts of contained impulses' magnitudes can be ranked and the whole impulse sequence shifted to move this prominent position to the measure's start, as shown in Fig. 4.15.

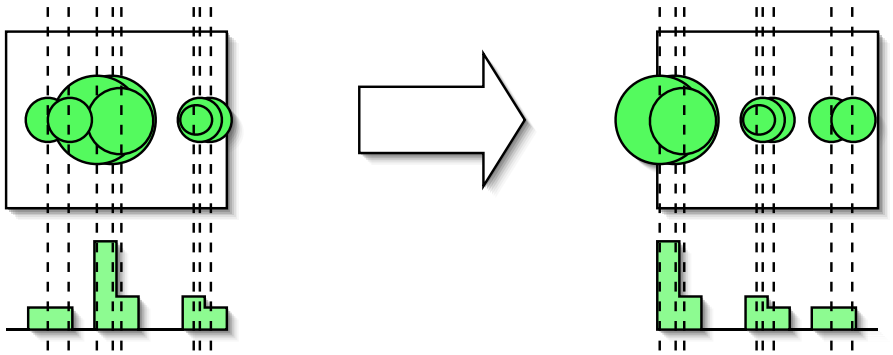

Figure 4.15: Histogram-based downbeat induction. The set of impulses on the left has most of its magnitude centered in the middle. This is shown by a histogram below, whose bar heights are justified by dropping dashed perpendiculars through the impulses' points in time. In order to get the significant impulses to the left, all need to be shifted. Anything moved out of the measure during this process is moved back in on the other end.

...or treat them separately

In case the various channels are to be folded separately, the same folding and histogramming techniques can be applied. Combining the resulting representations means to have eliminated any information on phase differences between the channels, because each was shifted by potentially very different amounts to start with its own downbeat. This may yield a more coherent picture when combining all channels than the first method, but losing the phase difference also means losing rhythmic information, as shown in Fig. 4.16.

On the other hand, treating channels separately allows to make use of knowledge over their respective de-facto beat intervals. Instead of

[3] Although one can imagine some form of pre-processing in the style of the other algorithm, yet to be described.

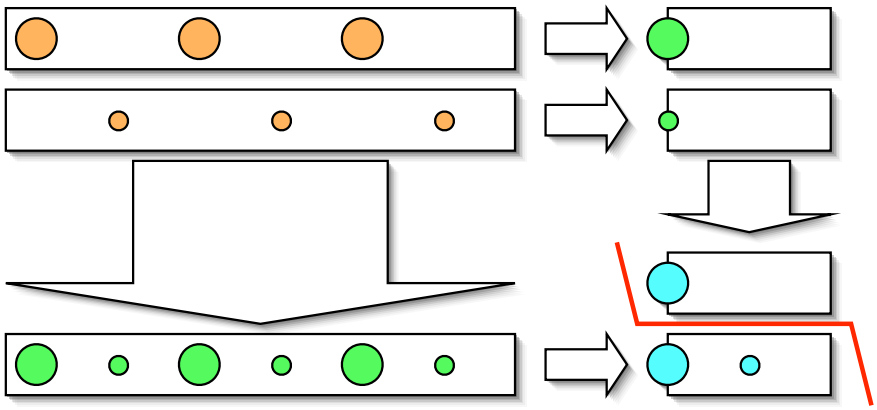

Figure 4.16: Loss of information by synchronization of channels. The two impulse sequences shown on the top left yield different rhythms with different folding methods. When first combining all channels, phase differences are considered (bottom left), when separately ordering channels' rhythms by their downbeat's position, this information gets lost.

blindly folding all impulses into the derived common intervals, channels' mutual irregularities can be dampened by locally stretching impulse sequences to fit the target beat interval or measure length. See Fig. 4.17 for an example.

To be able to do this, downbeat impulses must be identified before folding and thus without a histogram method. This is possible by looking for all inter-impulse intervals that are close to the respective channel's measure length. Trying out all chains of these intervals, those with a length near to the length of the complete window of observation are sorted and that consisting of the largest-magnitude impulses taken as the sequence of downbeats. An example sequence is shown in Fig. 4.18.

Clustering Impulses

Once all channels' impulses have been folded one way or the other, only their clustering to conceptual impulses is left. This could occur freely, wherever actual clusters are found and with various clustering methods that usually require at least one parameter that determines the closeness within which to cluster.

Cluster impulses to the beat level

Alternatively, all impulses can get clustered to the nearest position on the metric grid at the beat level. This results in a number of conceptual beats equalling the metric quotient. Fig. 4.19 shows the possible results the two ways of clustering.

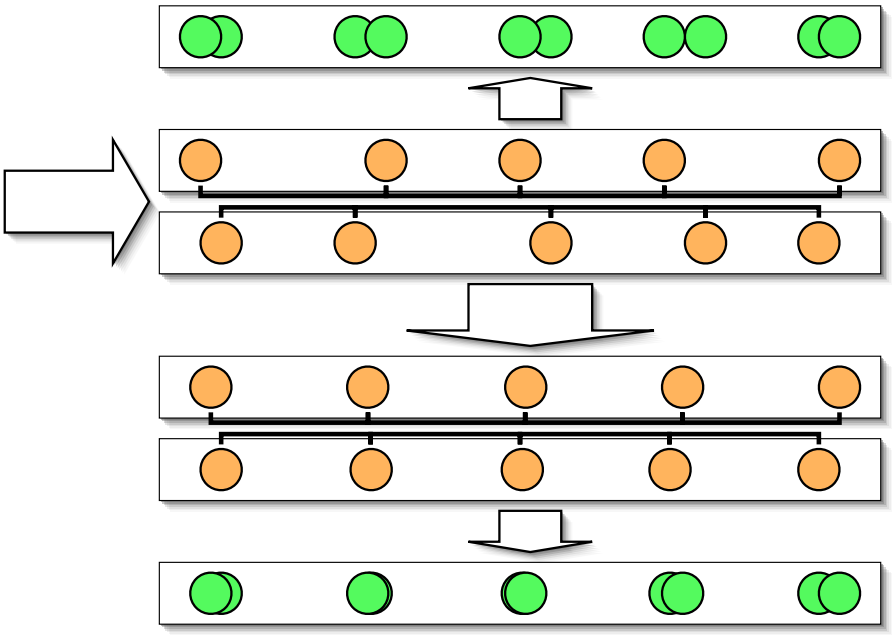

Figure 4.17: Fitting channels to each other by stretching. The upper rows of orange impulses pointed at by the arrow from the left are to be the rawly detected sequences. By superimposing them, their uneven distance shows (green, top). First stretching each channel so that downbeats are equidistant leads to a better positioning in the combined sequence (green, bottom). Non-downbeat impulses could also have been stretched depending on their position between two neighbouring downbeats, but have been omitted for clarity.

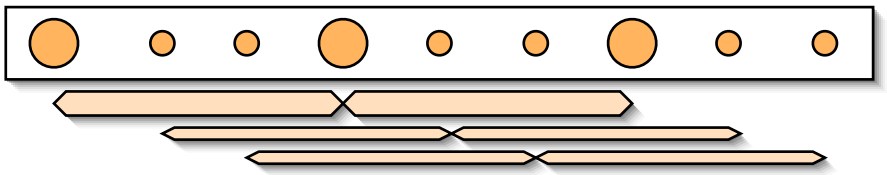

Figure 4.18: Finding the maximal sequence of downbeat impulses. Like in the figures concerning III histograms, the intervals between impulses are entered as double-sided arrows, but this time only those equalling (in a non-model case: approximating) the measure length. Among those sequences of intervals extending almost entirely through the window of observation, the one with the highest combined (added) magnitude is considered the one connecting the downbeats.

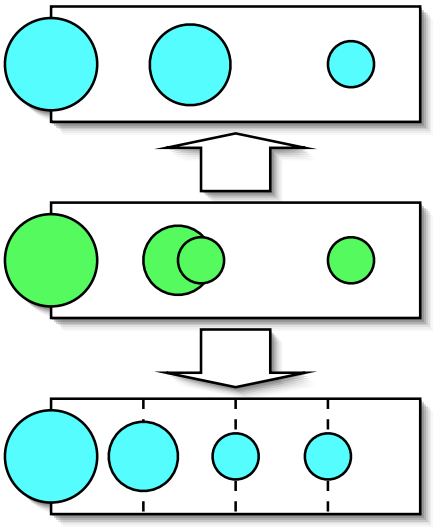

Figure 4.19: Two methods of clustering impulses. The green impulses in the centre are shown clustered freely, where they occur, on the top. A clustering along the metric grid is pictured at the bottom.

In case the clustering along the metric grid is chosen, an inert version of the magnitudes on the beat level can be used to smoothen the output. In the other case, a dampened version of the old clustering result might be added to any new one, but this can create artifacts due to the unpredictability of the free clustering.

In both cases, it needs to be taken care that clustering also takes place across the measure's border. For example, when clustering along the grid in a waltz example, impulses need to be clustered at count 1 (the measure's beginning), count 2, count 3 ($\frac{1}{3}$ and $\frac{2}{3}$ through the measure, respectively) and at a conceptual count 4 (the measure's end), whose cluster's combined magnitude then gets added to that of count 1. Fig. 4.20 shows a changed version of Fig. 4.19, in which this procedure is relevant.

4.2.4 Summary

Again, a pseudo-code abstraction (Alg. 2) is to give an overview of the general course of action suggested in this section.

More details, especially on the choices to be made, will be given in the next chapter, in Section 5.3.

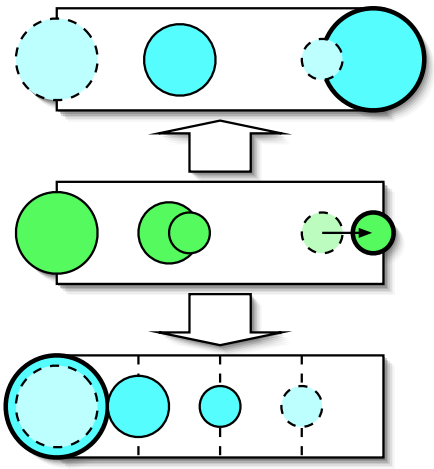

Figure 4.20: Clustering across measure boundaries. In comparison to Fig. 4.19, the impulse that is now indicated by the dashed green circle has been moved to a new position (with a thick border). This results in changes for the clustering: the free method (above) now considers the first and last impulse close enough to be counted as one in the resulting rhythm. The combined conceptual impulse now appears at the very end, as that is the two contributing impulses' average position, considering the measure as a loop (which it is to represent). Clustering on the grid changes in so far as the first conceptual impulse grows, while the rightmost one disappears.

Algorithm 2 Deriving a rhythm from sensor signals and from a sequence of movements detected in them

```
while there is another detected motion impulse do
  update all inert III histogram with any new intervals
  analyse all III histograms for a beat interval and a measure length
  update inert histogram of metric quotients
  vote on a common beat interval
  update inert beat interval
  if channels are to be synchronized then
    for all channels do
      fold all impulses using maximal downbeat sequences
      determine the combined downbeat via histogramming
      reposition the impulses accordingly
      cluster to conceptual impulses
    end for
  else
    modulus-fold all channels' impulses into one measure
    determine the combined downbeat via histogramming
    reposition the impulses accordingly
    cluster to conceptual impulses
  end if
end while
```

5 Implementation

5.1 Overview

In Section 1.2, some less algorithmically involved sub-tasks have been named besides the rhythm detection. This chapter deals with all of them, although the emphasis lies on the rhythmic analyser DanSense itself.

ProxyS4M receives from Sense4Motion

Fig. 5.1 shows a diagram of DanSense in relation to those other programs. On top, there is Sense4Motion as the source of all sensor data. At least for this thesis, but future projects may use other sensors as input. For this purpose, DanSense has been designed to receive its input via an established application-layer protocol: *Open Sound Control* (OSC).[1] The thus required translation between Sense4Motion's data format and the OSC messages expected by DanSense is done by ProxyS4M.

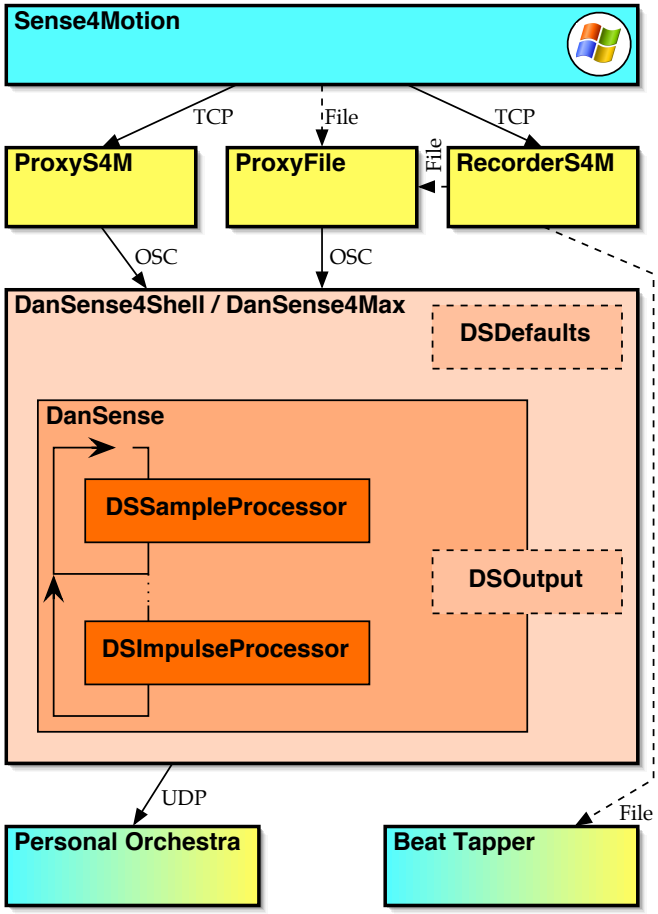

Figure 5.1: Overview of DanSense and related programs. Components of DanSense appear in shades of red, auxiliary programs in yellow and 3rd-party software in cyan. Arrows indicate data flow. For a detailed description, please refer to the text.

RecorderS4M writes files, ProxyFile reads them

Similarly, ProxyFile can read Matlab files written by Sense4Motion and send them to DanSense, getting rid of the requirement to do a 'live' analysis. It also reads files written by RecorderS4M, a wrapper for Sense4-

[1] For the specification see `www.opensoundcontrol.org/spec-1_0`. Of course, there is still a DanSense-specific data format to be adhered to within the OSC context, which will be discussed in Section 5.2.1.

Motion that was necessary in order to also include time stamps of music. The benefit of those time stamps is that it later allows a synchronous analysis of the accelerometer readings and the 'underlying' music. That analysis, in turn, can take place using an adapted version of BeatTapper, where a video recording, a sound waveform and accelerometer data can be viewed synchronously.

DanSense processes received samples

The essential part of DanSense itself is a Java class of the same name: it receives sensor samples, analyses them and, along with control values, outputs a representation of the detected rhythm. In order to make this class a potential module for other software projects, both the setting of parameters and the output of results works via method calls. To test the analysis algorithms, two front-end programs have been developed, which can also serve as examples of how to use the class DanSense:

- DanSense4Shell constitutes a command-line wrapper, allowing the passing of parameters as command-line arguments and displaying the output in a rudimentary textual and pseudo-graphical way on the console.
- DanSense4Max is an *external object* for Max/MSP and as such is used as one of a number of functional boxes in a processing graph. One such graph (called *patch*) is supplied that enables parameter setting through mouse controls and outputs the resulting rhythm, as well as intermediate data, graphically.

Unless the front-end used is supplied parameters, it uses the values defined in DSDefaults.

Employed by DanSense's namesake class, two main other classes exist, dividing their work just as Chapter 4 suggested: DSSampleProcessor looks at each successive sensor sample and detects bulges in the various sensor channels' graphs. Found bulges are returned to DanSense, which hands them over to DSImpulseProcessor for rhythmic analysis.

Results of this analysis are handed back to DanSense, which then sends the output to an implementation of DSOutput that it had been given as a parameter by DanSense4Shell or DanSense4Max: this interface specifies output functions for different data types, as well as semantic constants to differentiate between various outputs of the same type.

DanSense can conduct music

In addition to the mentioned textual output, DanSense4Shell optionally sends information to a specifically adapted version of the 'virtual conducting' software Personal Orchestra via UDP[2], allowing the conducting to take place using one of the acceleration sensors.

The detailed descriptions that now follow are sorted in the same way as they have just been mentioned.

[2] *User Datagram Protocol*, specified in RFC 768, available on `tools.ietf.org/html/rfc768`.

5.1.1 Development Environments

Java

Most programming done in Java

For cross-platform compatibility, Java[3] was chosen as programming language. One benefit of this was that data acquisition could take place both on the Windows notebook supplied by Prof. de Jong's company *MultiPro* (because the driver software was only available for Windows), and on the chair's Macintosh computers. The same applies to processing, which, due to the relatively low sample rate, does not suffer from Java's potential lack of efficiency.

All Java programs and classes that are part of this thesis have been fully commented according to the *JavaDoc* standard. For more detailed explanations than the rather conceptual descriptions given in this chapter, please refer to the HTML documentation generated from those comments (in `Programs/JavaDoc` and on the web site about to be mentioned), as well as the additional comment lines within methods' code.

Programming conventions adhered to include the following:

- Class variables start with an underscore (_).
- (De-facto) Constants are written in uppercase letters.
- In case parameters are being changed by the called function, the JavaDoc comments explicitly say so.

Each program's directory contains a bash script `run.sh` that contains examples of the lines required for compilation and execution.

A final note on portability to other languages (though it should not be required): apart from Java's garbage collection, the code relies on the automatic initialization of arrays on the heap and lazy evaluation of boolean conditions.

Max/MSP

Max/MSP used for output of results

To facilitate a quick visualization of intermediate data structures and results, the actual rhythmic analysis was prototyped in *Max/MSP*[4]. This software suite allows a graphical modelling of algorithms using functional boxes (*objects*) with multiple inputs and outputs and that can, in addition to its native functions, execute one's own Java or C objects. A short introduction into Max/MSP is given in Appendix B.

[3] To be precise: Java Platform 2, Standard Edition, version 1.4.2.
[4] `www.cycling74.com/products/maxmsp`

Objective-C

Third-party adaptations written in Objective-C

Another environment used was Objective-C in combination with the Cocoa framework on the Macintosh, when adapting existing programs for use with the acceleration sensors.

Regarding programming conventions, the extensions have been kept similar to the existing code.

5.1.2 Download

The Java codes can be downloaded from the Media Computing Group's web server, at
`media.informatik.rwth-aachen.de/enke.html`.

BeatTapper's and Personal Orchestra's sources are closed and thus only available internally.

5.2 Auxiliary Programs

5.2.1 ProxyS4M

ProxyFile translates between Sense4Motion and DanSense

Consisting of a single, runnable class, ProxyS4M's job is to receive data as provided by Sense4Motion and to re-transmit it as OSC packets suitable for DanSense.

At least in its 8-bit version, Sense4Motion formally sends out TCP packets of 16 samples, but does this in bunches of two, resulting in bursts of 32 samples and the unfortunate delay of at least 200 ms in-between that was already mentioned when introducing the program in Section 2.1.2. As a demonstration of this, as well as Sense4Motion's deviation from the set sampling interval, ProxyS4M outputs two figures on the console for each received TCP packet: the average interval between samples and the time that has elapsed since the last packet.

Command Usage

When started, ProxyS4M can be given a number of parameters that each have their default values:

	code	possible values	default value
bits per sample	b	8 or 16	8
Sense4Motion host	i	any IP address	localhost
Sense4Motion port	p	any port number	7000
DanSense host	o	any IP address	localhost
DanSense port	q	any port number	57110 [5]

The codes given must directly precede the given parameter, a valid command thus being:

```
java -classpath ".:..." ProxyS4M b8 i7000
```
[6]

All this information is also given by the program itself when specifying no arguments.

Receiving from Sense4Motion

Apart from the parameter evaluation, the code's central part reads TCP and sends OSC packets.

The format of the packets sent by Sense4Motion depends on its version, determined by the number of bits per sample. As Sense4Motion versions correspond to versions of the used sensors (at least the two variants coming up during the course of this thesis), a number of other conclusions are also drawn about the data:

bits/sample	max. channels	samples/message	max. sampl. interval
8	8	16	6 ms
16	10	$\frac{120}{\text{channels}} = 12$	2 ms

Besides this, also the arrangement of values within one packet varies. The following lines each show how the integer value of sample number s and channel number c can be derived from `packet`, given the number of sensor channels as `numChannels`:

8 bit: `packet[s*numChannels+c]&0xff`

16 bit:
```
packet[s*2*numChannels+2*c]&0xff)-254)*256
+ packet[s*2*numChannels+2*c+1]&0xff
```

The bit-wise AND stems from the fact that Java considers all types signed, by default. This way, the resulting integer ranges are 0..255 and 0..1023, respectively.

[5] This is a standard port for OSC packets.

[6] Java's *classpath* must, also for compilation, be set to include the parent directory, because that contains the JavaOSC library. This also applies to the other command-line programs.

Sending to DanSense

These read values first have to be normalized, as DanSense expects them to be floating-point values in the range $[-1, +1]$. Then, using the library *JavaOSC*[7], for each received packet from Sense4Motion, an OSC message is created consisting of four elements:

1. an Integer object, the number of samples contained in this message,
2. another Integer, the number of channels per sample,
3. a Float object, the sampling interval and
4. an array of Float objects: the received TCP packet's sample values, grouped by sample, within those groups ordered by channel number.

Formally, the value of sample number `s` in channel number `c` can be found at the latter array's position `s*numChannels+c`. Precisely this extraction is done when DanSense receives the messages.

5.2.2 ProxyFile

With ProxyFile, DanSense analyses recordings

Just as ProxyS4M, ProxyFile consists of only one runnable class and sends OSC packets to DanSense, as described above. The difference is that it does not listen for TCP packets, but reads data from a text file in the specific format written by Sense4Motion or RecorderS4M. As described in Section 2.1.2, Sense4Motion can record sensor readings to files in the Matlab format, making them readable by mere parsing for numbers.

Command Usage

Said input file is among the parameters that ProxyFile accepts:

	code	possible values	default value
bits per sample	b	8 or 16	8
name of input file	f	any IP address	localhost
sampling period	m	any number ms	6 [8]
DanSense host	o	any IP address	localhost
DanSense port	q	any port number	57110

Like with ProxyS4M, there may be no spaces between codes and parameters. A valid command for processing a file named *recording.m* would be:

```
java -classpath ".:.." ProxyFile b8 frecording.m
```

[7] `www.illposed.com/software/javaosc.html`
[8] 6 ms in the 8-bit case, 2 ms for 16 bit.

Information on the required arguments is given in case the user fails to supply them.

Reading Files

When reading an input file, the first step is the determination of the number of lines, which, subtracting the known header lines, is indicative of the number of samples stored. The number of channels is determined by the number of bits per sample: as many channels are expected as is possible with the respective version of Sense4Motion.

Knowing these figures, a buffer is then created that can store all of the input file's sample values. To show their format, the following are drastically shortened examples of Sense4Motion's and RecorderS4M's variants of Matlab output:

Output of Sense4Motion

Following an initial line about the sampling interval, Sense4Motion writes the sensor samples (here: eight) and a time stamp relative to an unspecified time scale:

```
Ts = 6e-3;
Macq = [170, 172, 172, 207, 211, 212, 164, 167, 528461957;
171, 172, 172, 207, 211, 212, 164, 167, 528461963;
171, 172, 172, 207, 211, 212, 164, 167, 528461969]
```

Output of RecorderS4M

Besides the additional comment line at the beginning and the slightly different ending, RecorderS4M's output differs in the numbers it outputs after the sample values (for an explanation, see Section 5.2.3):

```
% Columns:  s1x s1y s2x s2y s3x s3y s4x s4y delay sensorTimestamp[ms]
  musicTimestamp[µs] systemTimestampDelta[ms]
Ts = 6e-3;
Macq = [190, 208, 203, 223, 109, 166, 128, 169, 109, 82113526, 838639, 94;
189, 209, 204, 222, 109, 166, 131, 172, 109, 82113526, 838639, 94;
188, 210, 204, 222, 109, 166, 134, 177, 109, 82113526, 838639, 94;
];
```

Both programs start their lines with the actual sample values, so parsing can take place identically for them[9] and quite easily, using Java's StreamTokenizer.

[9] For future versions of Sense4Motion, this cannot be guaranteed.

Sending to DanSense

The sending of OSC packets is not quite the same as before in the case of ProxyS4M: there are now no incoming packets according to which the outgoing messages could be clocked and sized. As DanSense has to cope with bursts of data, anyway, it could theoretically be sent the whole file's data in one message.

Larger values than about 30 samples per message, however, tend to crash the OSC transmission library used. But, considering that there is practically no delay between those messages, this parameter has practically no detrimental influence on the fact that a file gets sent to DanSense in manifold the time it takes to record it.

5.2.3 RecorderS4M

Sense4Motion saves time-synchronized recordings

As mentioned before, the time stamps included by Sense4Motion for each sensor sample are relative to an unspecified absolute time scale. For analysis purposes, it seems helpful to be able to assign each sample a point in the local time frame of a musical piece playing in the background. This way, a graphing of the sensor recording can be annotated with properties of the music recording, like the subdivision of the time axis in measures as done in Fig. 4.8.

RecorderS4M, created to this purpose, expects to receive 8-bit data from Sense4Motion, plays an audio recording of the user's choice and saves the sensor data along with the time index at which the audio was playing at the moment of the respective sensor sample's reception. Fig. 5.2 shows the main dialogue and describes the required settings.

There are only two classes, RecorderS4M for the dialogue box and the thread RecorderS4MThread to do the actual work without making the dialogue box unresponsive.

Among Java's libraries, the program uses *Swing* for its user interface and *Sound* for loading the audio file. Due to the latter's limitations, only a few audio formats are supported. As those include the popular *WAV* format, compatibility is not an issue. The file size limit (experienced to be around 2 MB) was a disadvantage. But comparisons to underlying musical rhythm were not essential for this thesis, as discussed in Section 2.2, so perfecting RecorderS4M for daily use was not required.

Should this change in future extensions of this project, not only the audio solution would have to be improved (using Java's *QuickTime* libraries, for example), but also the data reception be made independent from specific sensor solutions.

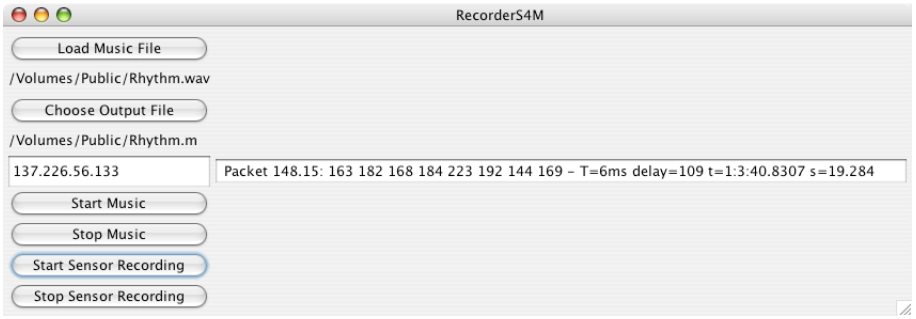

Figure 5.2: The main dialogue of RecorderS4M. The buttons on the left allow, as their labels say, the definition of audio input and data output files as well as their respective access control. In the text box on the left, the IP number of the Sense4Motion server can be entered, by default it is assumed to run on the same machine as RecorderS4M. Also by default, output files are named after the audio input. When receiving data, the text box on the right outputs the packet and sample number, the values received, the sampling interval, the delay between packets, Sense4Motion's time stamp and the chosen audio file's current time index. In this example instance, sample 15 of packet 148 bears a time stamp of about one hour, three minutes and forty seconds, and the music has been playing for about 19 s. One value mentioned in Section 5.2.2, "systemTimestampDelta", is not displayed: it merely shows the time difference between successive packets. Note that RecorderS4M's time stamps are identical over one packet, while Sense4Motion's are not, because in transmission, Sense4Motion only provides one per packet. Still, RecorderS4M could be changed to interpolate successive packets' time stamps.

5.2.4 BeatTapper

BeatTapper visualizes sensor data

To record the exact time indices of an audio recording's downbeat, BeatTapper was developed at the Media Computing Group. While playing a QuickTime video, it accepts clicks on a button or hits on the space key as commands to remember the instant at which that user event occurred, as a time stamp indicating the current playing position of the video. Each such recorded position is indicated as a thin line on the audio signal's graph, which is scrollingly displayed in parallel to the video. When the video is paused, the lines can be repositioned using the mouse. Together with the possibility to set the audio graph's zoom level, downbeats can be quite accurately positioned. Their positions can then be saved to a file and later be used for, e.g., beat synchronization in Personal Orchestra like described in Section 5.4.

For this thesis, BeatTapper has been adapted to also display sensor data in parallel: a Matlab file, as saved by RecorderS4M from an 8-bit recording, can be read and is then being displayed next to the video's

audio track. A screenshot is shown in Fig. 5.3.

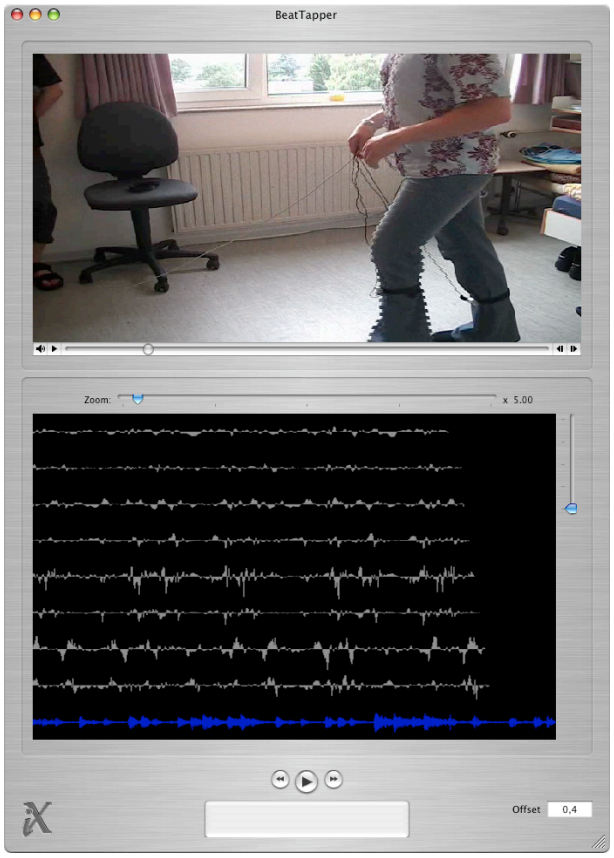

Figure 5.3: A screenshot of BeatTapper. At the top, a video of a dance performance can be displayed. (The picture is skewed to fit the already large program window.) At the very bottom, there are playing controls, as well as a light grey button to enter downbeats. The black area in the bottom half shows the sensor data in grey and the audio data in blue. The screenshot omits a vertical line that would normally show the current position in those data graphs. At the bottom right, the user can enter an offset value in seconds in case the video and sensor recordings did not start at the same time. Note that the sensor graphs' premature end is due to problems the screenshot program had with the way BeatTapper draws the screen.

This way, bulge features can be analysed while watching the video, potentially giving hints on how certain movements could be detected in the sensor signal.

Changes to the Program

Apart from the visual appearance of the user interface, most source code files of BeatTapper have been altered to include the display of sensor data:

- CBTController contains the methods called by menu commands, thus methods for opening and closing sensor data files have been added.
- CBTWaveView is responsible for drawing the graphs and has thus been changed throughout, supplementing the drawing of sensor data in addition to the sound file.
- CBTMotionContainer is a wholly new class, modelled after CBTMovieContainer, that does the actual parsing and loading of the sensor data.

The respective sections are marked and should be comprehensible to someone who knows his way around in BeatTapper.

5.3 DanSense

Besides indicating DanSense's relationship with the other programs treated in this chapter, Fig. 5.1 already gave a glimpse of its internal structure:

- DanSense4Shell and DanSense4Max are front-end wrappers that create an instance of DanSense. Using their input or the default parameters in DSDefaults, they start the processing, providing an implementation of interface DSOutput for any values DanSense may want to output.
- DSSampleProcessor and DSImpulseProcessor do what their names claim, the former being called by DanSense for each sample, the latter roughly for each detected bulge in the accelerometer graphs.

After an overview of the class structure, the central classes and then auxiliary ones will be discussed in more detail (Sections 5.3.2 to 5.3.6). An overview of the algorithmic and parameter-related decisions taken, and how those can be altered, is given in Section 5.3.7. Finally, Section 5.3.8 presents the mentioned wrapper programs and shows examples of their representation of DanSense's output.

5.3.1 Class Structure

Besides the three classes with algorithmic control structure (DanSense, DSSampleProcessor and DSImpulseProcessor), there are a number of general-purpose classes, as well as auxiliary ones specifically for use with DanSense. Their formal relationships in terms of relative numbers of instances are shown in Fig. 5.4 as a rudimentary UML diagram.

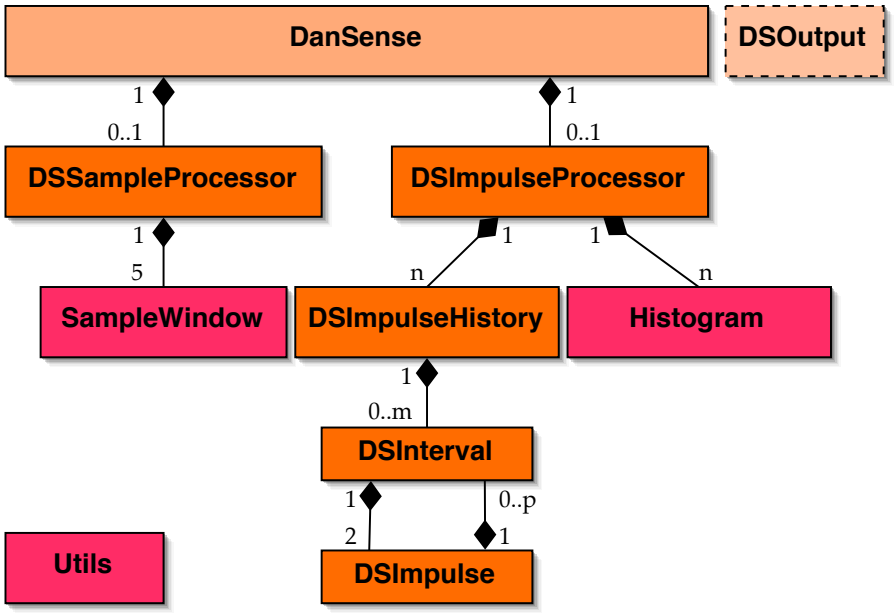

Figure 5.4: DanSense's class structure. As in Fig. 5.1, shades of red mark classes inherently associated with it. Those coloured magenta could also be used in other applications, and contain no reference to DanSense. DSOutput has a dashed frame to signal that it is an interface. With the exception of the static class Utils, which is used in various places, the other boxes are connected by lines that indicate numerical instantiation relationships. The black rhombus marks the object that owns the reference. For example, DSInterval and DSImpulse are the only classes to reference each other.

DanSense contains at most one reference each to objects of type DSSampleProcessor and DSImpulseProcessor. This is due to the fact that they only constitute 'out-sourced' code of the main class. Theoretically, they could be instantiated directly upon DanSense's own instantiation, but in order to allow later changes of parameters, it is possible to destroy and recreate them.

As repeatedly mentioned, DSSampleProcessor is called for each sample received by DanSense. To do offsetting, averaging, to prepare sample windows for frequency analyses, etc., it creates several instances of

SampleWindow, which basically is a circular buffer.

Once DanSense got reports of sufficiently many bulge (i.e., impulse) detections, DSImpulseProcessor is called to do rhythmic analysis. The thus needed track of past such impulses is kept in one instance of DSImpulseHistory per channel, the III histograms derived from those lists are each contained in one Histogram object. DSImpulseHistory itself manages a mutually linked structure of objects of types DSInterval and DSImpulse, each III referencing its constituent impulses and each impulse referencing the intervals it is constituent of.

5.3.2 Control Structure: DanSense

Initialization

DanSense receives data and controls processing

Being the central class, DanSense's two main tasks are the reception of data and the control of its processing. As a concession to Max/MSP, both are not immediately started once an instance is created, because this would then happen right after the loading of the respective Max/MSP patch. Instead, the method start needs to be called, allowing the user to change parameters from their defaults before anything happens.

start gets those parameters that need to be known in advance as arguments, others can be supplied during processing. All will be discussed summarily, later. Besides registering parameters, start creates an instance of OSCPortIn, part of the previously mentioned JavaOSC library. It is told to listen for data on the *address*

`/DanSense_data`

In OSC nomenclature, this is a name for the stream of messages to expect. This name must thus also be, and is, assigned to their messages by ProxyS4M and ProxyFile.

Also, an object must be specified whose class implements the interface OSCListener and thus the callback method acceptMessage to receive the data. As such object, the DanSense instance refers to itself. Its acceptMessage expects messages of the form described in Section 5.2.1.

On receiving the first message, the method initialize is called, where instances of SampleProcessor and ImpulseProcessor are created and handed the parameters they require to, for example, allocate memory. As there is at most one instance of each, these classes' names are, for simplicity, from now on used synonymously with their unique instances.

Processing is independent from reception rate

Also, an instance of Timer is created and, via an instance of the helper class DanSenseTimerTask, scheduled to run the method processSample as often as specified by the OSC message's sampling frequency value. In

a way, not immediately using incoming samples has the function of a buffer that not only smoothes variations in reception frequency[10], but also makes it possible to get files sent in chunks from ProxyFile.

Now, the data samples (floating point values in the range $[-1, +1]$) in the first message's main array, but also all subsequent ones, get passed to SampleProcessor's method addSample, which adds them to the object's Vector of samples.

Processing

SampleProcessor detects, ImpulseProcessor analyses impulses

Whenever the timer runs processSample, SampleProcessor's main method of the same name is called. If it returns impulses (as a Vector of instances of DSImpulse), they are handed to ImpulseProcessor's method updateImpulseHistories. If nothing was returned, processing stops.

But even if impulses were returned, processing usally stops here: a static counter is started for DanSense.processSample to wait some more iterations (i.e., scheduled calls to itself). This is done, because after detection of a movement in one channel, it is likely that others will follow, soon. Waiting for a handful of samples makes the ensuing processing worthwhile.[11]

Once the counter has reached its target value (default values will be listed separately), the actual rhythmic analysis is started by calling SampleProcessor.freqAnalysis and ImpulseProcessor.deriveRhythm. Their results are then sent to the send method of the object implementing DSOutput that was given as one of the parameters to start. Besides actual results, raw and intermediate data is output for illustrating purposes, an example being the signal graph in Fig. 5.5 (see Section 5.3.8). For this, various simple output methods of the processing classes are called, as well.

Termination

DanSense also has a stop method, which closes the OSC receiver, cancels the timer and destroys the instances of SampleProcessor and ImpulseProcessor. After each call to stop, start can be used again.

[10] In case the flow of samples cannot keep up with the rate they are processed, then the last received sample is used again until a new one is available.

[11] In a setting with sensor samples that arrive far less than 200 ms after their creation, this may have to be changed to allow for a more real-time approach. Anyway, the processing only takes about 5 ms per iteration, despite the comparably inefficient FFT computation in Java.

5.3.3 Movement Detection: SampleProcessor

Each call to SampleProcessor's method processSample, a sample is taken from the object's Vector of samples. If no sample is available, this is interpreted as a temporal interruption of sample flow, and the previous sample is used again.

For this sample's processing, a loop through all channels is started, of whose number the object was informed at its creation. The sample, due to the loop now a scalar value, is offset, low-pass filtered and registered for later frequency analysis using several instances of SampleWindow.

Then, any previously stored starting times of bulges are reset in case they have become too old. Then, the sign of the sample is determined and its absolute value used to find the arithmetic average and median of the respectively signed recent sample values.

Record impulses at zero-crossings

Beginning the actual search for impulses, the current sample value is compared with the previous to detect zero-crossings. If such has happened, the maximal amplitude and area of the most recent bulges of both signs are checked for sufficient magnitude. If they suffice, then either one instance of DSImpulse averaging those bulges' properties gets entered in a Vector, or separate ones. After that, the information on bulges is reset. Even if the magnitudes do not suffice, the new bulge started by the zero-crossing is registered by storing the current time stamp as its beginning.

Independently from whether the zero-line was crossed, the current (or new) bulge's maximal amplitude gets registered (together with the current time stamp) in case it is significant. Also, the bulge's area gets updated and the current sample value stored as the previous to facilitate the zero-crossing check next time processSample is called.

Finally, an array of all channels' Vectors of DSImpulse instances gets returned.

5.3.4 Frequency Analyses: ImpulseProcessor

Once DanSense.processSample decides it is time to let ImpulseProcessor process the accumulated impulses, first a rhythmic analysis of the sensor signal takes place. As mentioned before, SampleProcessor.processSample register every incoming sample for this analysis. For the FFT or autocorrelation, a window of a certain number of past samples is required. This window is supplied, optionally as a down-sampled variant to, for example, not let the FFT deal with frequencies without rhythmic relevance.

Extract beat intervals by analysis of windowed signal

The actual work on this copy is done by freqAnalysis. For each channel, it creates a Hann-windowed copy of its windowed data and feeds it to FFT and auto-correlation algorithms. The beat intervals extracted by detecting the first significant peaks of the resulting arrays are then returned to the calling DanSense.processSample.

Choosing n to represent the number of the first significant peak's bucket, s as the sampling interval in ms, d as the size of the window (in samples) used for downsampling and w the size of the window used for the respective analysis method, the formulae for the beat intervals b are as follows:

$$b_{\mathrm{AC}} = s \cdot d \cdot n$$

$$b_{\mathrm{FFT}} = \frac{s \cdot d \cdot w}{n}$$

5.3.5 Rhythm Derivation: ImpulseProcessor

On ImpulseProcessor's instantiation, it creates as many instances of DSImpulseHistory as there are channels.[12] These instances are later handed their respective channel's impulses by ImpulseProcessor.updateImpulseHistories, which in turn got them from DanSense.processSample.

Update histograms with each impulse

When the latter calls ImpulseProcessor.deriveRhythm, all instances of DSImpulseHistory are made to update their instances of Histogram according to the impulses received since the last such update. Returned are those histograms' first significant and maximal peaks, as well as the sum of all their buckets, i.e., their weight. The first two are interpreted as beat interval and measure length, respectively.

Update metric quotient with histogram analysis

The metric quotient derived from them is used to create a Histogram instance of metric quotients, weighting the channels' contributions according to their histograms' weights. This histogram gets mixed into an inert one, whose highest-peaking bucket then determines this processing iteration's official metric quotient.

Derive beat interval by voting

Next, the channels' metric properties are then used to derive the largest group of channels that can agree on a beat interval, just as described in Section 4.2.2. Likewise, the update of the inert beat interval requires no structural explanation beyond the algorithmic description on the one hand and the documented code on the other.

Depending on a parameter choosing among the options described in Section 4.2.3, either all instances of DSImpulseHistory are asked to put their impulses in one instance of Vector, or this is done separately. Apart from the fact that the downbeat gets detected with yet more instances of

[12] This number of channels can be double that used in SampleProcessor, in case impulses are returned separately for the positive and negative sides of the acceleration graph.

Histogram, there are, again, no implementation-related specifics worth mentioning about this, or the subsequent clustering of impulses.

5.3.6 Helper classes

To sum of the description of DanSense's classes, descriptions of encapsulating and other helper classes follow:

DSImpulseHistory

Each instance of DSImpulseHistory keeps track of one channel's impulse history and III histogram. The former consists of a Vector of DSInterval instances, which are changed up by calls to the methods add and trim by ImpulseProcessor.updateImpulseHistories. The method add adds an impulse, whose intervals are then computed, trim discards too old impulses along with their respective intervals.

The histogram is kept current through calls to DSInterval.updateHistogram by ImpulseProcessor.deriveHistory. It is only changed if add has been called since the last update. Even if not, however, its weight is returned as a value decreased according to the histogram's age.

For the two variants of impulse folding, there are two respective methods: getFoldedImpulses gets passed a Vector and adds all its known DSImpulse instances to it, having applied the measure length's modulus to their time stamps. getFoldedImpulsesSeparately uses the method of looking for a sequence of downbeat impulses and folding the sequence at their positions, before adding the result to the passed Vector instance.

DSImpulse

More than being a mere container of impulses' magnitude, time stamp and spread, an instance of DSImpulse also manages references to the intervals it is constituent of. Apart from adding and removing such references, it offers the method removeIIIReferences as a precursor to its own destruction: its call serves to get all the impulses with which it shares intervals to remove their own references to those.

DSInterval

DSInterval does no more than constructing its own instance from two of DSImpulse and answering questions for its properties or those constituent impulses.

SampleWindow

An instance of SampleWindow is created by simply giving a length. For each call to replaceValue, the passed value is written to position specified by a positional counter, that counter incremented and the previously saved old value returned.

The current contents of the window are returned by getLinearized. The method circularFilter returns the result from computing a linear combination of the current window contents (starting where the positional counter points) whose coefficients are read from an array passed as parameter. The latter is used by SampleProcessor to smoothen data samples.

Histogram

Like a normal histogram, Histogram offers the possibility to define its dimensions, add (floating-point) values to buckets and read them. add also adds triangles of data, as required by the III histogramming in DSImpulseHistory. For use with downbeat detection after impulse folding, addModulo adds triangles in a way that they spill over from one end of the histogram to the other.

mix can mix in another histogram to treat its own instance as an inert one. For those cases, getWeight also gets the histogram's current weight, decayed according to the time passed since its last update.

Utils

General-purpose functions that do not require their own class are combined in Utils. It contains findMax to find maxima in arrays of type float and findFirstMax to find the first maximum above a specific threshold. Both allow the setting of an offset where to start searching (e.g., to avoid the 0 Hz bucket in the FFT's result) and the size of a window, the sum over which is used to determine a maximal position instead of the contents of just one bucket.

Besides minor helper functions, also the FFT implementation itself (c2c-FFT) is found in Utils.

5.3.7 Options and Defaults

The above descriptions of DanSense's classes have specifically not included every single method, let alone a line-by-line description of the code. As already mentioned, JavaDoc-style comments have been in the source and an HTML documentation generated from that.

Missing, however, is a comprehensive listing of all the decisions within the bounds of last chapter's algorithmic descriptions that, in the implementation, have been taken in one of the following forms:

A. fixed by the algorithmic source code,

B. fixed in the static constants' class DSDefault,

C. left open for run-time deviation from what is set in DSDefault (as parameter to DanSense.start, or

D. left open for processing-time deviation from what is set in DSDefault (having its own setter method in DanSense).

In the following overview of such decisions and parameters, each is prepended a letter corresponding to one of the above degrees of finalization. This should also be a helpful list for future work that is rather interested in improving DanSense than extending it.

Options for the Control Structure

C *How many iterations to wait until detected samples are processed?*
This is set to five, with respect to limiting additional delays. As already mentioned, it is computationally very conservative, anyway, to wait between processing iterations.
Constant: CHANNEL_SYNCHRO_WAIT [iterations, > 0]

Options for Movement Detection

C *How large to choose the window over which the offset for sample values is computed?*
At a sampling frequency of 160 Hz, offsetting over 200 samples seemed a good compromise between eliminating gravitational influence and capturing long movements.
Constant: OFFSET_WINDOW_SIZE [samples]

C *How large to choose the window over which the current activity level is computed?*
For lack of a reason to do it otherwise, the offsetting's constant is used.

A *Should the current activity level be determined using arithmetic averaging or using the median?*
Currently, the arithmetic averaging is taken, because apart from the median's lesser value, no advantage concerning detection quality could be seen. The median's code is still contained, but lines making it effective are commented out.

C *How large to demand bulges' amplitudes to be for them to count as impulses?*
By default they have to reach 80% of the current activity level and at least have an amplitude of 10% the maximum acceleration. For noisy signals, the former should be increased well above 100%.
Constants: MIN_AMPLITUDE_FACTOR, MIN_BULGE_AMPLITUDE

C *What age difference to accept between the starts of bulges for them to form a pair?*
Single movements of a length exceeding more than the one second appears unlikely, especially considering the lengh of the offset window. The default has thus been set to 1000 ms.
Constant: MAX_MOVEMENT_DURATION [ms]

C *Should one impulse per significant bulge be output?*
By default, one impulse represents both bulges, so the constant is set to *false*. This also makes the number of channels treated in ImpulseProcessor the same as for SampleProcessor.
Constant: SEPARATE_SIGNS [*true* or *false*]

A *Which properties of a bulge to use how for its impulse's?*
In the case of one impulse per bulge, the product of area and maximal amplitude is taken as the magnitude and the average between its starting and peaking time as time stamp. The bulge's width serves as spread value. In the case of one impulse per pair of bulges, their combined areas and combined amplitudes are multiplied to get the magnitude, the common zero crossing is used as time stamp and half their combined width as spread.

Options for Frequency Analyses

B *By what factor and how to downsample sensor values for frequency analysis?*
The sixth line in Pascal's triangle is chosen as smoothing FIR[13] filter (coefficients 1, 5, 10, 10, 5, 1), so the factor is six.
Constant (for the filter): SMOOTH [array of floating-point values]

[13] *Finite Impulse Response* filters are such whose output is influenced only on a finite range by individual impulses. The given smoothing filter is one, because for each target sample, only the neighbouring six of its corresponding source position are combined: the given coefficients are those for their linear combination and subsequent division by the sum of coefficients.

C *How large to choose the window over which the frequency analyses are computed?*
The focus being on the FFT, a downsampling factor of six makes a window size of 256 extend over a comfortably sufficient interval of 9.6 s. Due to the chosen FFT algorithm, this must be a power of two.
Constant: ANALYSIS_WINDOW_SIZE [samples]

A *Which windowing function to use for frequency analysis?*
For this, the Hann function (see Appendix A.1) is chosen.

C *How to decay the inert frequency analyses?*
A variant of Seppänen's decay formula has been used (cf. Section 4.2.2), whose parameter c is, for several inert contexts, represented by various constants that will be discussed in Chapter 6 for their effect. That for frequency analyses is set to 1.[14]
Constant: FREQ_ANALYSES_INERTIA

A *How to combine the frequency analyses' (FFT and auto-correlation) beat intervals to a common value?*
Currently, the auto-correlation results are discarded.

Options for Rhythm Derivation

A *How to combine two impulses' properties to an interval's?*
The magnitude is the minimum of the impulses', the length is set to their distance in time, and the spread is their spreads' average.

C *What age to allow for impulses to remain in the window of observation, i.e., how large to choose that window?*
A suitable value depends very much on the style of movement. In the current implementation, though, this parameter should be chosen identical to the following.
Constant: MAX_IMPULSE_AGE [ms]

C *How large to choose the histograms' width?*
This value should be well below that of the allowed impulse age, as it is intended to capture recurring features within that period. By default, it is set to 2 s.
Constant: HISTOGRAM_WIDTH [ms]

B *How large to choose the width of the histograms' individual buckets?*
The bucket width defaults to 20 ms, limiting deviations from the actual value (within the precision bounds of the method as such) to 10 ms.
Constant: HISTOGRAM_BUCKET_WIDTH [ms]

[14] Actually, parameters c and d in the formula are preset to some useful dimension, leaving the factor to be tuned in the one- or two-digit range.

C *How large to choose the decay factor for the inert histograms of IIIs and those of metric quotients?*
Like in the case of frequency analyses, these are used in Seppänen's decay formula. They are set to 1 and 10, respectively.
Constants: MQ_HISTOGRAM_INERTIA, III_HISTOGRAM_INERTIA

A *How large to choose the shifting window over which histograms' peaks are determined?*
Five buckets are used. This is more than sufficient when expecting the histograms' other parameters to be roughly set in a way that using single buckets would already be possible.

A *What amplitude to demand from histogram peaks to be considered significant?*
The first peak, used for the determination of the beat interval, must be at least double the average bucket value. As it is computed over a window, that window's buckets' sum must be higher by a factor of the window's size in buckets.

A *When to discard channels' metric properties as not trustworthy?*
When beat intervals or measure lengths less than 10 ms or above 2 s result, the respective channels are not considered in the remainder of the current processing iteration. This also applies when the histogram's weight is below 10, but sensible values for that parameter are very much dependent on the absolute size of the values added to the histogram, which, in turn, is primarily determined by the way impulses' magnitudes are computed.

B *How many metric quotients to consider in their inert histogram?*
Considering that metric quotients above 12 involve very unusual factors (like the mixing of rhythms with metric quotients of 3 and 5), the constant has been set to that value.
Constant: MAX_METRIC_QUOTIENT [ms]

A *What deviations to allow from exact integer multiplicity between channels' beat intervals in order to let them support each other in a vote?*
They are discarded when the longer beat interval is more than 1.2 times as long as the shorter one.

A *How to combine beat intervals determined by histogramming and by frequency analyses to a common value?*
Currently, the FFT value is preferred if it is in proper bounds, i.e., between 10 ms and 2 s.

D *Should impulses be folded separately for each channel?*
In order not to lose rhythmic information by being oblivious to phase differences (cf. Fig. 4.16), this question is answered negatively (set to *false*). In case bulges are detected separately for both signs of a sensor signal, the inherent phase difference between the two makes it advisable to activate this synchronization. Constant: SYNCHRONIZE_CHANNELS [*true* or *false*]

A *How large a deviation to allow from the observation window's length when selecting candidates for downbeat impulse sequences?*
Allowed are chains that exceed the supposedly maximal length by half a measure and do not fall short of it by more than 1.5 measures. This guarantees that sequences of, for example, only two large impulses cannot alone influence the measure length by which their channel is folded.

As anounced before, those values most decisive for the processing of different kinds of movements will be revisited in Chapter 6, *Evaluation*.

5.3.8 Front-End Wrapper Programs

Instead of calling DanSense from one's own program, a way to try the algorithm are the two supplied wrappers: one for the command-line shell, one for Max/MSP.

DanSense4Shell

DanSense4Shell calls DanSense from the command line

That for the shell is a runnable program called DanSense4Shell. Its main method evaluates any arguments, creates an instance of DanSense, sets that instance's parameter for the synchronization of channels and calls its start method with the remaining parameters.

It also hands it a reference to a new instance of its internal class ConsoleOutput, which implements DSOutput and thus offers a range of overloaded functions named send for DanSense to call for output. To semantically distinguish outputs with the same data type, DSOutput defines a number of constants that are passed to the send functions by DanSense.

It also sends the values Personal Orchestra requires to conduct music using the acceleration sensors (see Section 5.4). This sending was, arbitrarily, chosen to take place whenever the common beat interval is received from DanSense.

Command Usage

As for the other command-line programs, the parameter codes given must directly precede the respective parameter:

```
java -classpath ".:.." DanSense4Shell c127.0.0.1 w1
```

The following summarizes the parameters that can be given to DanSense4Shell, as well as their default values. Most of the latter are given as constants defined in DSDefaults and discussed in the previous section.

	code	unit	default value
OSC server port	p	port no.	OSC_SERVER_PORT
PO host	c	IP no.	(none, no PO output)
PO sending port	d	port no.	1233
PO receiving port	e	port no.	1234
offset window size	o	samples	OFFSET_WINDOW_SIZE
freq.-analysis window size	a	samples (2^n)	ANALYSIS_WINDOW_SIZE
fold channels separately?	y	0 or 1	SYNCHRONIZE_CHANNELS
one impulse per bulge?	s	0 or 1	SEPARATE_SIGNS
max. movement duration	v	ms	MAX_MOVEMENT_DURATION
histogram width	h	ms	HISTOGRAM_WIDTH
impulse processing freq.	w	iterations	CHANNEL_SYNCHRO_WAIT
maximal impulse age	m	ms	MAX_IMPULSE_AGE
minimal magnitude factor	f	(float)	MIN_AMPLITUDE_FACTOR
III histogram inertia	i	(float)	III_HISTOGRAM_INERTIA
metric-quot. histogr. inertia	q	(float)	MQ_HISTOGRAM_INERTIA
frequency analysis inertia	r	(float)	FREQ_ANALYSES_INERTIA

All this information is also given by the program itself when specifying no arguments.

Output

The send methods output some figures that are rather interesting for debugging than to use it. The actual output of the rhythm takes place textually:

```
830 850   0   0   0   0   0   0 [measure lengths]
430 430   0   0   0   0   0   0 [beat intervals]
  2   2   0   0   0   0   0   0 [metric quotients]
AutoCorr beat interval (first ch.):  468
FFT beat interval (first ch.):  418
 12 105 137 340 365 378 435 498 515 520 525 809 820 831 [times]
  7   8   7  14  29  19  15  54  43  39  38   8   8   8 [mag.]
#             0             |
common measure length:  860
common beat interval:  430
```

The first three labelled rows of numbers show the respective value for all channels, in this case eight. Channels whose values were discarded by the algorithm are represented by zeroes.

Additionally, the beat intervals determined for the first channel by auto-correlation and FFT are output. The metric quotient appears to be two, which fits the fact that this is also the one derived by the two significant channels.

The next lines give the times and magnitudes of the folded impulses before clustering to a rhythm. As an interpretative example, one can

see that the impulses with large magnitude are centered at a position around about 500 ms into the measure. This would, for the clustered rhythm, then be moved to the beginning of the measure.

This rhythm, with a metric quotient of two, is then displayed by the line with the characters "`#`" and "`O`". They indicate the conceptual impulses, the second impulse being weaker than the first. Even lower levels of magnitude would be indicated by "`o`", "`.`" and "` `" (the space character). "`|`" shows the position of the measure's end.

The last lines show the measure length and beat interval detected, thus explaining the scale used in the pseudo-graphical line just above.

With the exception of the magnitudes of the rhythm's conceptual impulses, which needs to be stored until their time stamps are known, too, all received values are output immediately. Also required to be stored are the measure length (to be used for the rhythm display) and the sensor tilt transmitted to Personal Orchestra.

DanSense4Max

DanSense4Max graphically outputs results

DanSense4Max is an external object for use in Max/MSP[15] As such, it extends MaxObject from Max/MSP's Java library.

To get input, it declares inlets to whose markers values can then be fed in a patch. Max/MSP instantiates DanSense4Max on loading a patch containing it. Besides the various numerical inputs to change parameters, messages *start* and *stop* can be sent to the object, calling the respective functions of DanSense.

In contrast to DanSense4Shell, which could not do this due to its static nature as a runnable problem, DanSense4Max implements DSOutput without a helper class. Its send functions immediately hand over data to Max/MSP's outlet function, so outputs occur at those outlets that correspond to the values of the constants in DSOutput that define the various kinds of output.

While the outlets as such are also defined automatically from the definitions in DSOutput, inlets have to be changed in DanSense4Max's code. This is because their listing in a general DanSense class would not be of use to other implementations.[16]

[15] Those unfamiliar with Max/MSP should have a look at Appendix B first.

[16] The use of constants to semantically distinguish outputs could also be replaced by defining a separate function for each kind of output. However, the use of one send function per data type (integer, float, array of integer, array of float) very well fits Max/MSP's equally structured range of outlet functions.

With the exception of the one controlling whether impulses get folded separately per channel, all parameters are stored internally in order to be able to supply them to start next time it is called. They, as well as an example use of the outputs can be seen in Fig. 5.5, which shows the Max/MSP patch supplied with DanSense.

For reasons of clarity, not all objects and connectors are shown, especially not the separate patches *RhythmGraph* and *OnsetGraph*. These are referenced in the boxes with the prepended command *p* and are responsible for displaying a graphical representation of the rhythm (in the window "rhythm") and one of the folded, but not yet clustered impulses (in the window "impulses"). Whereas the latter are shown without any borders, the rhythm appears with two vertical lines that indicate the measure boundaries.[17]

The parameters shown use the names of the respective variables in DanSense or, capitalized and with interspersed underscores, occur as their constants' names in DSDefaults. With the exception of the one explained in the text of Fig. 5.5, those constants can all be looked up in Section 5.3.8.

Throughout the development of DanSense, this Max/MSP patch's graphical output served as the benchmark. The displayed example is that of a simple movement of two sensors on a circular path, alternating between two diameters. The intended pattern "strong–weak" is indeed detected, but at least in the seconds before the screenshot was taken, one strong impulse did not coincide with the others.

Chapter 6 will give more examples of the patch's output.

5.4 Personal Orchestra

Personal Orchestra (PO) is a software developed at the Media Computing Group, allowing the conducting of recorded orchestral performances using an optically tracked baton. The influence a user has over the playback of the recording can be summarized in three points:

- The audio and video tempo adapts to the frequency of the baton movements, without changing the audio's pitch.
- A series of positions in the recording identified as the downbeat using BeatTapper is made to coincide with the bottom positions of the conducting gesture.

[17] The colours used for the impulses chosen along a rainbow scale. If all were coloured identically, overlapping circles would be hard to see.

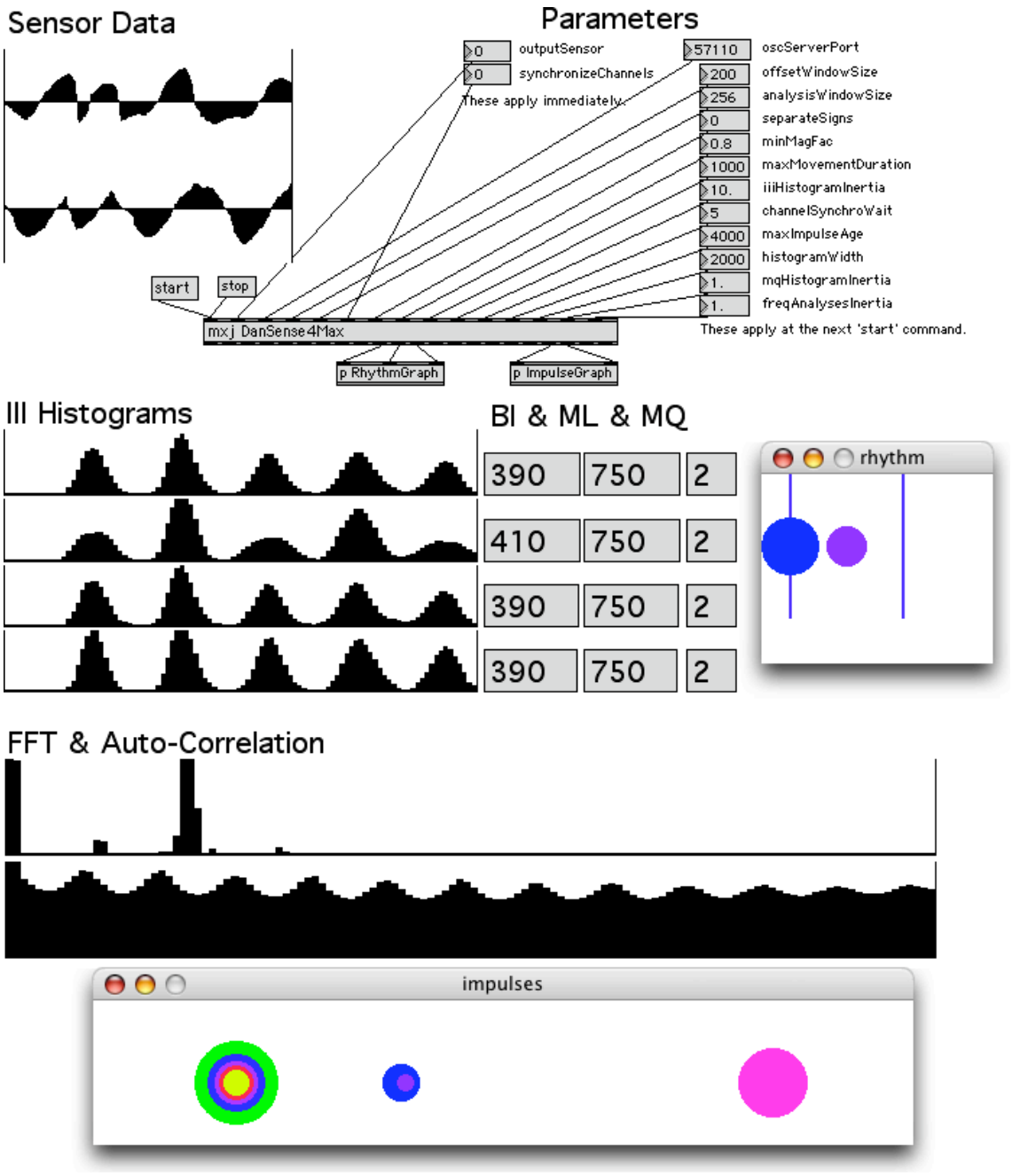

Figure 5.5: A Max/MSP patch for DanSense. Shown are the patch with the DanSense4Max object, its parameters and outputs, but also two separate output windows on top. The parameter *outputSensor* chooses the sensor for some outputs that are not given to all due to space constraints: the raw data on the top right, the frequency analyses' graphs at the bottom and the III histograms in the middle, alongside their extracted beat intervals ("BI"), measure lengths ("ML") and metric quotients ("MQ"). In case there is only one histogram per sensor axes, those of a neighbouring sensor are shown as well. File-based communication is indicated by dashed lines, objects are dashed when static, i.e., when not instantiated. For more information on parameters and the patch's structure, please refer to the main text.

- The direction the baton is held into while conducting emphasises the instrument group in the respective part of the orchestra, assuming a calibrated position of detector and screen.

PO has been adapted to accept accelerometer input

To demonstrate the theoretical use of acceleration sensors as input devices for conducting, a recent version of PO has been adapted to receive information from DanSense. Assuming that one sensor is used for the conducting, the current beat interval is used to determine the playback's tempo. One axis' tilt is interpreted as an indication of which part of the orchestra to emphasise. In order to avoid abrupt tilt readings due to the permanent oscillation of the sensor, an average over a certain number of past samples is used.[18]

While the common beat interval of all sensors is that also sent to PO, only one sensor's axis can sensibly be used to derive the tilt value. For this, the sensor with the most regular movement (determined by the III histogram weighing most, cf. Section 4.2.2) is determined and its 'second' axis taken. In the case of the used sensors, this makes sense as their form intuitively suggests to use the other one for producing the beat interval readings.

As the delay of 200 ms incurred by the used version of Sense4Motion does not allow the sensor movement's phase to be used as real-time input for beat synchronization, that functionality has been left out in favour of an interpolation of guessed downbeat positions. Experience shows that, once the system has adapted to a change in tempo, the user instinctively gets his movement in sync with the music's downbeat positions, thus giving the illusion of real-time behaviour. The temporary changes in movement frequency necessary for this synchronization have no noticeable effect on the music's tempo, as they get blurred in the processing algorithms' window of observation.

5.4.1 Data Transmission

The sending of tempo and tilt information is initiated by DanSense and performed by DanSense4Shell via UDP datagrams. The information is sent each time a rhythmic analysis is performed, but the UDP socket is only opened once, when DanSense4Shell starts.

The data is sent as a simple concatenation of two four-byte IEEE float values. The used Java class ByteArrayOutputStream converts the floats to to big-endian byte representation independently of the machine DanSense is run on. This fits the big-endian architecture of Macintosh computers

[18] This is responsible for one of the required instances of SampleWindow in SampleProcessor.

(on which the used Objective-C is dependent), which is why on the receiving end, no further conversion beyond a mere cast is necessary.

5.4.2 Changes to the Program

As part of the adaptation, the class POCP_ServerDanSense has been created and those named StateMachine and UserInput have been changed. Their relationship is sketched in Fig. 5.6 and described below.

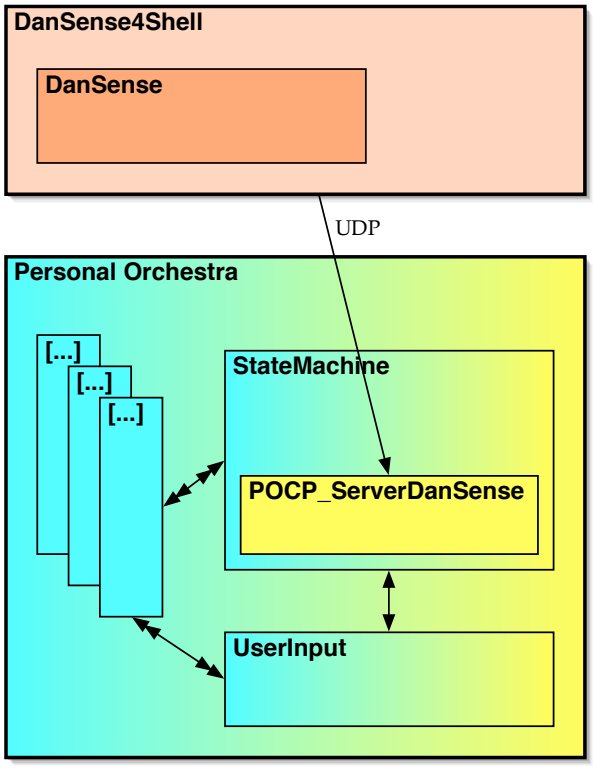

Figure 5.6: The DanSense adaptation of Personal Orchestra. As before, the subboxes represent classes, red indicating DanSense, blue software created by others (in this case at the Media Computing Group) and yellow auxiliary code, here the adaptation of Personal Orchestra to use accelerometer data as input.

POCP_ServerDanSense is a changed version of the existing POCP_Server. It contains methods to accept a UDP connection and receive data.

An instance of POCP_ServerDanSense is created by StateMachine, which hands the former a pointer to its own instance, thus enabling POCP_ServerDanSense to call methods of StateMachine. The latter, in its changed version, contains state variables on the current tempo, sensor tilt, current *beat* (in PO terms this means the downbeat phase, i.e., the position

relative to the bounds of the current measure) as well as the time of those values' most recent setting. Apart from a setter method, it also features such to get the set values.

In the case of the tempo, the average value between the previous and the newly received one is used to provide a smoother input for the audio stretching system.[19] Also, the mentioned interpolation of downbeat positions takes place here.

The values are read by the instance of UserInput, the class containing the methods used by the playback system and user interface for deriving conducting information. In two of its methods, code has been changed so that now, the values sent by DanSense are used to determine tempo, instrument emphasis and their respective control displays.

Only minor are the changes made to allow tempi in excess of four times the standard. One obvious reason for this is the increased fun involved in getting the orchestra to play at 600 bpm. A more serious one is that very low readings of the beat interval set such a high tempo that many interpolated downbeats quickly accumulate. Even if this is only in effect for a short while, normally-paced downbeats will later be added to the 'pile', forcing PO to spend a long time catching up with all those individual downbeats that have meanwhile accumulated. In order not to have the tempo set to a low maximal tempo for the rest of the piece, short-lived and seemingly unreasonable tempi should be allowed.

Such changes have been applied to STF_AudioTimePitch, STF_AudioTimeStretch and STF_Synchronize by setting respective factors to 10 or 0.1. As other changes to the code, they can be found by searching for upper-case "DANSENSE" comments.

[19] The tilt, as mentioned, is already sent by DanSense as an interpolated value.

6 Evaluation

On the one hand, the screenshot of Max/MSP's graphical output already showed the used algorithms to give a clear indication of a rhythm when fed with simply structured movements' data. On the other, empirically justified skepticism concerning the complex nature of dance movements was voiced in Section 4.1.2. Such a mixed message is going to be this chapter's.

First, a few tests that come to mind are constructed and evaluated. Later, the appropriate setting of parameters for their most effective analysis in a dance context is discussed, and results of DanSense's application to recordings of such movements are described. While this chapter concludes with a final assessment of this thesis' success, a treatment of possible changes beyond those of mere parameters are deferred to Section 7.2, *Future Work*.

6.1 Test of Intended Features

+ Simple patterns are detected correctly

The theoretical soundness of a good portion of the proposed ideas is proven by the fact that moving the sensors in a well-defined way yields a result mirroring the intent. Supplying a continuous repetition of the sequence *strong–weak–weak* along one axis, Max/MSP displays

- a roughly correct measure length[1] and
- three equidistant circles, the largest at the beginning, the others at relative positions $\frac{1}{3}$ and $\frac{2}{3}$ of that interval.

Figs. 6.1 to 6.4 are to illustrate the above claim with the described and other manually induced rhythms. They each show a screenshot of about five seconds' sensor data on top and corresponding histograms below. Their extracted values (beat interval, measure length and metric quotient) appear on the right and the resulting rhythm at the bottom. Except for a longer setting for allowed impulse ages (5 s), standard parameters have been used. The diagrams' sensor graph extends over about 5 s, the histograms over 2 s and the full width of the output window (except in the first example, where it is smaller) represents about 1.6 s.

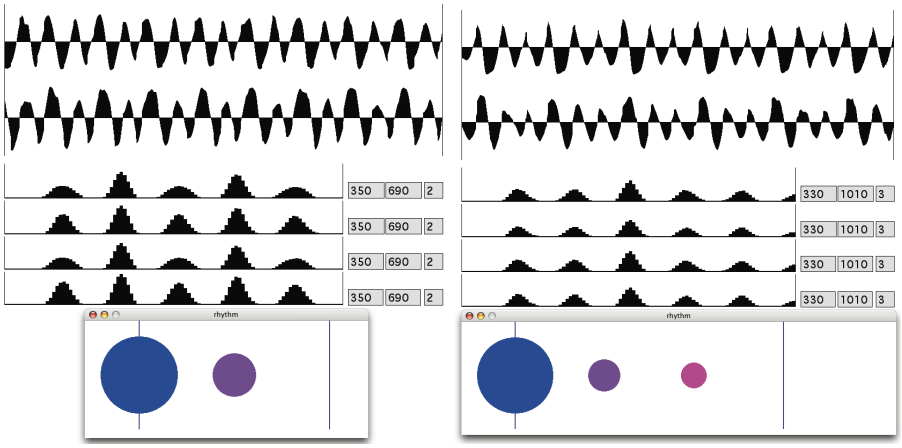

Figure 6.1: Circular sensor movements I. Shown are a circular *strong–weak* movement (left) and a *strong–weak–weak* version (right) with two acceleration sensors. The resulting rhythms were first obtained after four and six seconds, respectively. On the right, one can see that the bulges appear to have a decreasing magnitude. This is either due to human imperfection by failing to produce movements of equal strength or equal distance, or to the fact that bulges might be paired up by the algorithm in a way that combines the latter strong bulge (negative) with his weaker successor on the positive side.

– Downbeat is not always output at measure's beginning

Taking care to keep the movement uniform, it is also possible to effect a size ordering on the beat-level positions (*strong–medium–weak–weakest*). Attempts at this, however, show the same fault that could also be seen in the diagrams of Fig. 6.3: although histogramming is used to determine

[1] The precision of the only "roughly correct" measure length could be improved by making the buckets of the III histogram smaller, extending the analysis window (for the FFT), decreasing the downsampling factor or using auto-correlation. But, with respect to the resulting rhythm, imprecise readings of beat intervals or measure lengths appeared not to be the source of problems, especially considering that the finally used value is computed as a weighted average.

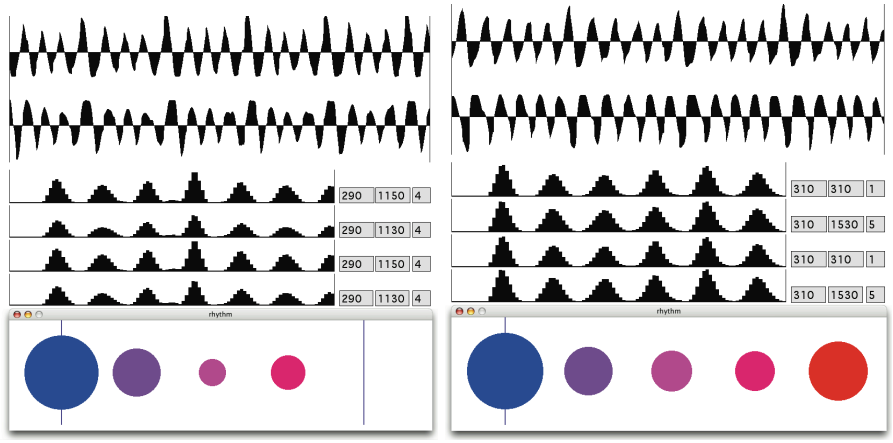

Figure 6.2: Circular sensor movements II. Again, circular movements of two sensors are shown, this time a *strong–3×weak* version (left) and a *strong–4×weak* one (right). The resulting rhythm was first obtained after nine or 14 seconds, respectively. A reduction of the III histograms' intertia to 7, that of the metric quotient's histogram to 0.1 and that of frequency analyses (here: FFT) to 0.9 let the latter delay shrink to 5 s. Again, magnitudes are uneven in both cases. The increased strength of the last one on the right may, again, be due to a mixing with the strongest movement. However, the second would then seem to get some share of that magnitude, as well, so there must be other factors. Those are, in both cases, human in nature: either there is an inclination to move stronger at near the intended strong impulse, or imprecision in the impulses' spacing leads to a clustering that mixes old downbeat impulses into others.

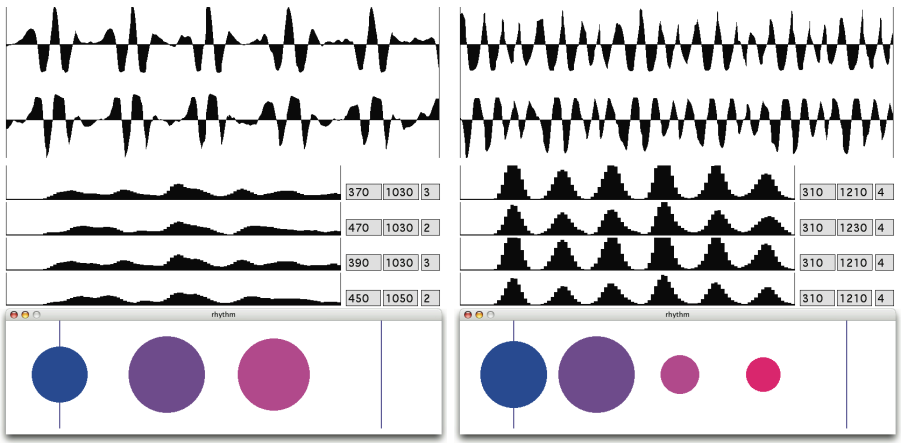

Figure 6.3: Circular sensor movements III. The movements shown are both intended to be of metric quotient four: the left one of the kind *strong–strong–pause–pause*, the right one of *strong–strong–weak–weak*. Nevertheless, the derived metric quotients are different. Apparently, creating the pauses at the right distance did not succeed completely, also the aftershocks contributed to form a third beat.

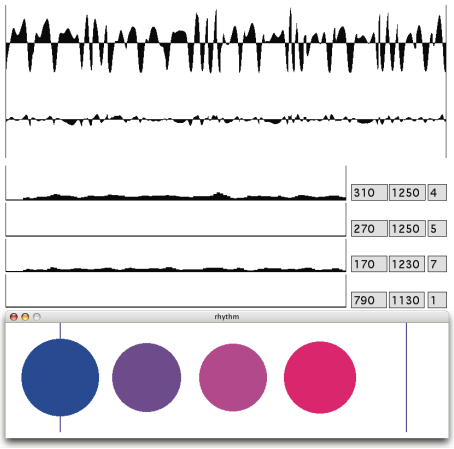

Figure 6.4: Circular sensor movements IV. Here, a sequence of five equally strong movements was intended, but the latter two should only have half the others' duration, like three quarter notes being followed by two eights. DanSense infers a metric quotient of four, which can be called correct in the sense that the counting would probably not occur in the interval of "eigth notes".

the position (in the measure of folded impulses) that should become the measure's beginning, the conceptual impulse on count 1 is not always the largest. Apparently, the dynamics of clustering are not completely done justice by the histogramming technique.

High metric quotients are problematic

Detection-wise, problems occur once rhythms with a metric quotient of above five are tested: the first and third histograms shown in Fig. 6.5 have their maximum at their first bulge, so only the other two vote for six as the metric quotient. At sensors one and three, the difference in magnitude between the intended downbeat impulses and one measure's remaining five seems to be too small to let the pairs of downbeat impulses outweigh the sheer number of pairs that are only one beat interval apart.

– Common metric quotient prevents detection of combined rhythms

One might instead try to create a complex rhythm by combining two sensors' information. The pattern given in Fig. 4.16 may serve as a simple example: with each one of two sensors, an equidistant sequence of impulses is created, but with a phase difference putting one sensor's impulse exactly between to of the other. If one sensor is moved more vigorously than the other, a *strong–weak* pattern should emerge, the figure conveys. It turns out that this only works in case there is some channel supporting the hypothesis that the metric quotient is two, or if that belief is still existent by the metric quotient's inertness. In a 'clean' environment, however, moving the sensors precisely as described will make all channels report a metric quotient of one. This makes it impossible for them to agree on *strong–weak*, even if their impulses are

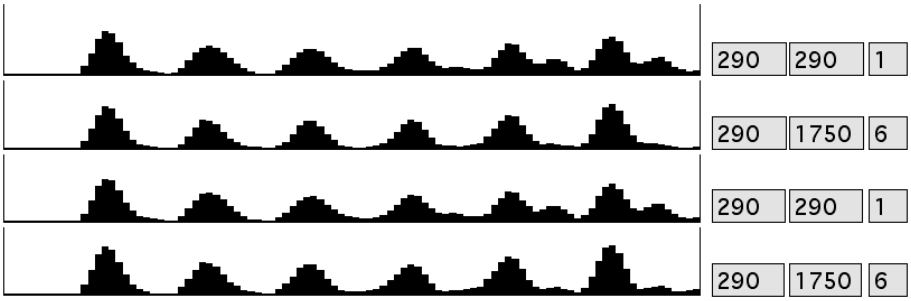

Figure 6.5: III histograms of a movement with metric quotient six. Two of the four sensors slightly favour the determination of one as the metric quotient, because their first bulges are, although by only a tiny amount, their respective largest.

folded together and thus keep their phase information. Fig. 6.6 shows DanSense's analyses of attempts at employing a phase difference.

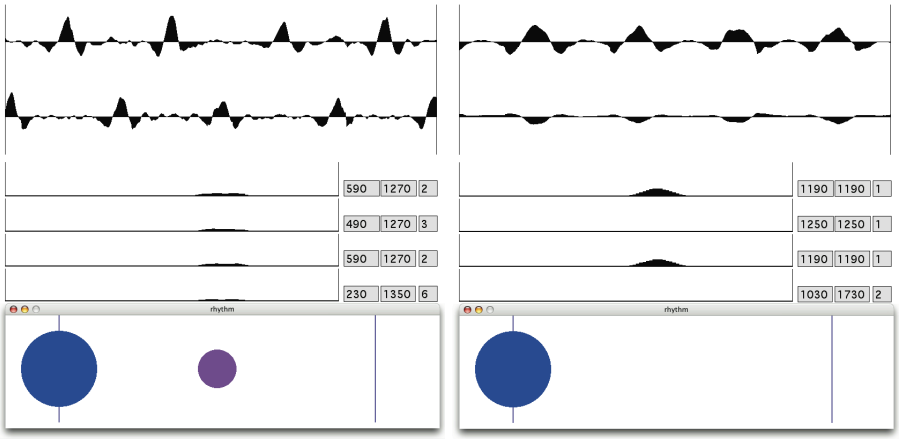

Figure 6.6: Attempts at creating rhythms through phase differences. The left diagram shows the data of one sensor that is alternately moved along its two axes, at different strengths. Although two channels read a metric quotient of three, that of two prevails and yields the intended rhythm. As the vote on the metric quotients came from the channels themselves, the movement along the axes was, apparently, not exclusive, but also influenced the other, leading to *strong–weak* rhythms on both channels and thus also in the combination. In the right-hand diagram, however, alternation takes place between sensors (this can't be seen, as always only one sensor's data graph is output). The moved axes' channels each report a metric quotient of one, and as these feature the histograms with the highest weight, this value is used for the combined rhythm, which therefore only consists of a single conceptual impulse. (Note: Histograms are not scaled up in order to give a proper comparison to the other diagrams.)

+ Histograms' weight properly indicates relevance

Anyway, the use of histograms' weight as a factor in decisions proves useful, as is already shown when one person moves one sensor regularly and another moves a different one chaotically: the regular rhythm is clearly detected.

Clipping appears to have little effect

A doubt raised in Section 2.1.1 was that the sensors' limited acceleration range and the fact that they appear not to be centered at $0\,g$ might lead to excessive clipping and thus hamper the detection of movements. Deliberately turning a sensor in Earth's gravitation so that even at rest, the signal's graph is near the maximally possible amplitude, constitutes a worst-case scenario, in this respect. But, even then, the acceleration toward the hardly measurable direction suffices to produce bulges in the graph that register as movements. It should be kept in mind, though, that in case positive and negative bulges are treated separately, those representing the 'disadvantaged' side will hardly show differences in magnitude.

+ Parameters allow precise adaptation to known demands

Stability could be another criterion for judging the algorithm's quality, but with the three parameters influencing the inertia of III histograms, frequency analyses and the histogram deciding on the metric quotient, stability as such is easy to achieve.

The main challenge is to calibrate the aforementioned parameters to fit the expected performance: a rhythm expected to be constant throughout the analysis period will allow for the conservative addition of new values, while in other cases, the output metric quotient should rather not remain constant at one value for half a minute if all channels have consistently voted for another throughout that period.

6.2 Suitable Parameters

Precisely this trade-off is a major problem with dance movements. While the useful ranges of the parameters described in the previous chapter were quite large, they have to be calibrated more carefully in the case of dance performances appear. Sensible settings (whose de-facto effectiveness will be discussed in the next section) appear to be the following:

- Set the maximal duration allowed for a single movement (MAX_MOVEMENT_DURATION) according to the expected abruptness. In a disco dance, 500 ms should be more than enough.
- Set the maximal impulse age (MAX_IMPULSE_AGE) to at least three times the expected pattern length, probably the underlying music's measure length. This should minimize problems like those shown in Fig. 6.5 and allow a search for sequences of downbeat impulses: in the example of a piece at 120 bpm, a measure extends

over 2 s, so in a time frame of 7 s, at least three downbeat impulses should be found.

- Set the inertia for the III histograms (III_HISTOGRAM_INERTIA) to a (double-digit) value that makes peaks persist for a few measures, allowing the respective channel's vote on the metric quotient to remain constant in the face of a temporary lack of supportive intervals.
- Similarly, set the inertia for the frequency analyses (FREQ_ANALYSIS_INERTIA) to one that makes it just robust enough. In the current form of the chosen algorithm, the FFT is the prime determinant of the beat interval, thus this factor should perhaps be chosen highest in unstable circumstances.[2] The histogram's metric long-term influence, in contrast, is once more secured by the...
- inertia of the histogram that keeps track of past 'sentiments' on the metric quotient (MQ_HISTOGRAM_INERTIA). One is tempted to set this very high, but once there is a change of metric quotient in the piece, the emerging discrepancy may take long to correct.

It turns out that a mismatch of the inertial parameters can lead to effects like a consistent metric quotient, but a varying beat interval. But then again, the parameters that lead to this effect might otherwise produce good results on the signal in question.

6.3 Tests with Dance Movements

Among the about thirty recordings made with different dancers, none had a stable reading with parameters that just about proved sufficiently stable to show a consistent reading when moving a sensor by hand. Even adherence to all the above rules did not yield a convincing reading using the dance recordings.

One could argue that reading a metric quotient of two for more than half the duration of a performance along *Rhythm Is A Dancer* (cf. the signal graphs in Fig. 4.8) is not bad considering the underlying music's metric quotient of four. But this should rather be called a step in the right direction, as deriving that two is a factor of the metric quotient is surely easier than deriving the correct one.

Figs. 6.7 and 6.8 uses the mentioned example to show the volatility of DanSense's output.[3]

[2] Its default value has nevertheless been chosen comparably low to yield an acceptable adaptability to intended tempo changes when DanSense is used in connection with Personal Orchestra.

[3] This section's parameters were, besides default values, a minimum magnitude factor of two and inertial factors for III histograms (ten), metric quotients (0.7) and

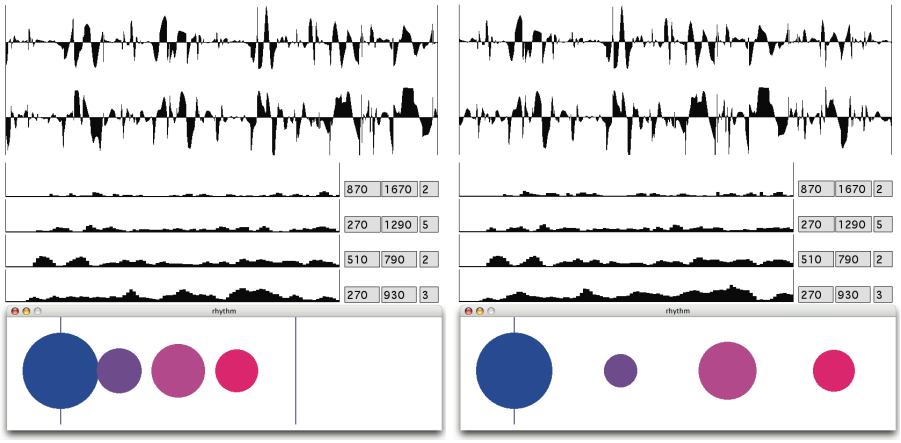

Figure 6.7: Rhythm output for *Rhythm Is A Dancer*. The left diagram shows a proper detection of the danced metric quotient, at least the underlying music's, which is four. The measure length, however, should be 1.9 s, but is less than half that figure. The diagram on the right shows the situation about one second later, as can be seen from the shifted sensor graph. The beat interval has doubled, approaching its correct value.

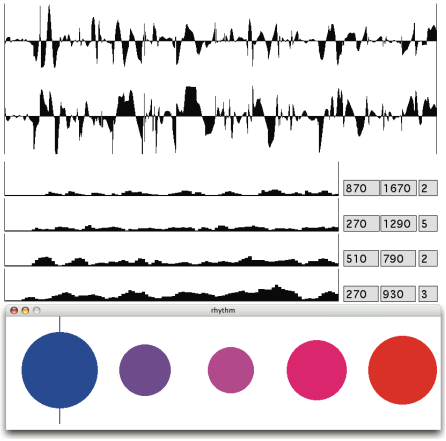

Figure 6.8: Faulty rhythm output for *Rhythm Is A Dancer*. In addition to the previous diagram's doubled beat interval, now the metric quotient has increased to a wrong value, five.

The main problem appears to be the movement detection, considering the comparably high quality of the results when moving the sensor in a controlled fashion. But also in the case of, for example, the *Cha Cha* moves performed by a professional dancer and shown in Fig. 4.6, the algorithm requires a quite high value of histogram inertia for the metric

frequency analyses (1.3).

quotient (around two) to stick to its most prominent output of a metric quotient of two. A look at the data excerpt of the mentioned figure, this should rather be a four. Trying this *strong–strong–pause–pause* by hand shows that such a sequence can theoretically be detected. Apparently, little phase differences between the channels prevent the correct metric quotient from 'locking in'. Fig. 6.9 shows one of the better moments of the analysis.

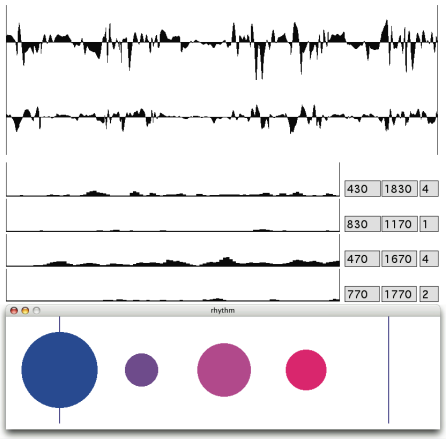

Figure 6.9: Rhythm output for a *Cha Cha* dance. Despite not capturing the intricacies of the rhythm as could be seen in 4.6, a correct metric quotient is used for the output.

Had the results of the algorithm's application to dance recordings be more promising, the originally envisaged benchmark, could have played a role in this evaluation. That was letting the dancer judge in how far the output matches his (or her) intuitive idea of the rhythmic pattern he produced. But, as long as DanSense only produces acceptable results for movement of the sensors by hand, such a study appears baseless.

In any way, the taken sensor recordings can, just like the programs' source codes, be downloaded from:
`media.informatik.rwth-aachen.de/enke.html`.

There, it is also indicated at which body positions the various sensors were located during the recordings. A hint on the best location for the sensors could not be derived due to the wide variety of styles tried, ranging from standard dance steps (*Rumba*) over popular choreographed dances (*Macarena* by *Los del Rio*, 1993) to music not associated with any specific movements (*Moondance* by *Nightwish*, 1998).

6.4 Conclusions

Considering the lack of precision the task description bore due to the lack of a specific target application, a conclusion as to how far the aim was reached can only be equally vague. The following statements can be made:

- The movement detection works well with simple movements like when waving with one's hand, but has difficulties discerning important from unimportant acceleration onsets when applied to complex dance movements.
- Given rather regular sequences of impulses as input, the chosen algorithms successfully infer metric properties.
- The frequency analysis using the Fourier transformation helps to stabilize the common beat interval derived from several sensor channels' inter-impulse-interval histograms.

The main problem the evaluation showed is that the erratic movement extraction from complex movements can only with considerable limitations regarding responsiveness and flexibility of the system be converted into a clean and steady output. Lester [Les86, p. 77] writes about musical rhythm: "Once a metric hierarchy has been established, we, as listeners, will maintain that organization as long as minimal evidence is present." In DanSense as well, the various inert data structures (III histogram, histogram of metric quotients, beat interval and magnitudes of the rhythm's conceptual impulses), lead to a quite conservative behaviour when registering information that contradicts the stored values.

Before suggestions of how this can be changed are subject of the next chapter, some concluding 'should-haves' are in order:

- The question of how to derive rhythmic structures may be academically more interesting, but in retrospect, a better movement detection appears more important. It is unclear, though, how much room for improvement there is.
- Instead of applying algorithmic changes to the chosen histogram approach, the focus might better have been on doing a comparative study of simple versions of various methods.
- Had the topics "movement detection" and "rhythmic analysis" been declared two separate and ranked aims, their separate development and testing could have benefited both by clarifying modules' individual deficiencies.

What cannot be seen from this thesis at all are the different development stages the algorithms and implementations lived through. DanSense started as a Max/MSP patch, changed into a number of external Max/

MSP objects, then found its way encapsulated as a single Java object before being split again so as to make the mentioned modularity possible. Although these are no conclusions regarding academic content, there are some lessons to be learnt regarding the planning of projects and the deliberation of when to rather write code and when to rather rely on rapid prototyping tools.

The resulting code, at least, should be quite legible and thus make it possible for a continuation of the project not to rely on the ideas, diagrams and pseudo-codes described in this printed (or PDF) version.

7 Summary & Future Work

7.1 Summary

Recalling the aim of this thesis, an algorithm should be developed and implemented that derives a rhythmic representation from accelerometer data using a specifically supplied prototype of a sensor package. The kind of movements to analyse was not specified, nor was the term 'rhythm' or its desired representation formally defined. (Chapter 1)

Contemplation of what might be suitable led to the notion of rhythm as a set of conceptual impulses, characterized by their mutual intervals and relative magnitude. Defining measure length and beat interval as the determinants of rhythm's recurrence and internal structure, it was suggested that conceptual impulses appear on positions specified relative to the bounds of one rhythmic pattern. (Chapter 2)

Previous work studied for inspirational input included such employing movements for musical purposes, as well as such analysing music for rhythmic aspects. Whereas none of the former had done a survey of how to detect movements in sensor data, the latter research provided

several ideas for rhythmic analysis, like interval histograms and types of frequency analyses. (Chapter 3)

Adapting the ideas to this thesis' setting, a pipeline of algorithms was developed to detect movements from the sensors' signals and derive a rhythm from them, specifically determining measure length and beat interval, but also the downbeat as the most prominent conceptual impulse. (Chapter 4)

These algorithms were implemented in Java, as a library called DanSense, together with example front-ends and some helper applications that turned out helpful during the parallel development and evaluation of the algorithms. Also, as a proof of concept, conducting software was changed to process input from the given accelerometers. (Chapter 5)

Comparing the implementation's detection results for various parameter settings, the usefulness of the developed algorithms and implementations was compared with the original aim and found to be on the right track, but not yet suitable for use in an end-user application. (Chapter 6)

The following, final section will now discuss possible ways to improving DanSense.

7.2 Future Work

The ways that this thesis could lead to further activities can be classified in five categories:

- alterations of minor algorithmic decisions taken,
- extensions to the described algorithms,
- complementary use of wholly different approaches,
- changes to fundamental concepts and
- extensions of the aim originally set.

7.2.1 Alterations to Algorithmic Decisions

Apart from changing the constants that determine parameters like window sizes, the decay of histograms or the minimal magnitude for movement detection, various choices in algorithmic design have been taken that require actual code changes to alter. Among possible questions to be asked are:

- *How to prepare the signal for proper analysis?*
  The low-pass filtering could be extended considerably, e.g., removing all frequencies below those one wants to detect.

- *How to derive impulses' properties from their respective bulges'?*
  For example, peak times instead of attack times could be used as time stamp. In the case of clipped sensor graphs, the middle of the clipped area could be considered the peak and a higher magnitude extrapolated.
- *What to consider a significant movement?*
  Is it possible to specifically detect unintentional aftershocks by some other way than counting bulges alternately on both sides? Perhaps the relation to the last detected bulge's size can play a role. Also, the current level of activity could be determined by the frequency of detected bulges instead of the average sample value.
- *How to define intervals' properties using their constituent impulses'?*
  Some function may be better suited to yield a common magnitude than the minimum function.
- *How to combine metric structures derived using Fourier transformation with those yielded by the histogram approach?*
  There may be some ingenious way of discarding chaotic channels' input, but in any case a comparison of all channels' votes with previous values appears more promising than using a potentially wildly varying Fourier reading.
- *How to fold impulse sequences?*
  Reliance on the downbeat alone may no longer appear to be a good idea if one considers that the similarity of whole measures could be compared. If some kind of auto-correlation were to be applied, downbeat sequences might get some confidence rating, or be derived where this previously was impossible.
- *How to find the correct downbeat position and cluster beginning with the downbeat?*
  The derivation of the downbeat and the maximal downbeat sequence might be improved by executing them in combination. Currently, the downbeat is determined via histogramming, a technique whose problems have been described in Section 6.1. It might instead be found by the sequence detection, that is anyway being done. An alternative for the clustering along the metric grid would be an iterative version, i.e., clustering at an increasing coarseness. This would, in a way, do histogramming and clustering at the same time.

7.2.2 Extensions to Algorithms

The following ideas represent considerable deviations from the current algorithm:

- The question of how intensively to filter the signal might not be answered at all, letting the algorithm run on various versions of the signal at different levels of smoothness, then choosing the most promising information on a per-sensor basis. Different signal versions might be differently suitable for movement detection and frequency analyses.
- Movement detection could take place across sensors, or at least across all of one sensors' axes. This could take place by either calculating the sum of squares, or using a combined rule-base scheme that uses one axis' information to improve detection quality on the other, like mutual support of bulges.
- The detection of the metric quotient might make use of the knowledge of the beat interval, which, in the case of the FFT, is derived independently, anyway. As mentioned in Section 6.2, beat interval and metric quotient sometimes start to mismatch abruptly. In a combined approach, the algorithm might conclude that a sudden doubling of the beat interval should either not occur, or be matched by a doubling of the metric quotient. This would also counter problems with III histograms' readings for the measure length, as shown in Fig. 6.5.
- Determining the beat interval using III histograms' peaks could be changed to some form of Seppänen's greatest common divisor, as described in 3.2.1.
- Should the III histograms' new vote on a metric quotient contratict the latter's inert histogram, the III histograms could be searched for evidence of the inert information. This way, unreasonably long persistence of outdated metric quotients could be prevented, but still the case avoided that little histogram changes have a decisive impact.
- Instead of folding a whole sequence of impulses into one measure, they could be searched for multiple such sequences. In certain cases, this may give hints to an ongoing tempo change.
- The algorithm could be told which sensor is attached where and change parameters or even methods according to the kind of data that can be expected from there. This could even be extended by specifying sensor positions in combination with kinds of dances.
- Rather than the current alternatives of either first folding all channels' impulse sequences and then combining them, or folding their combined sequence, the latter could be changed to detect phase differences between limbs. Currently, the folding is oblivious to whether the impulses came from eight sensors or from a single one.

7.2.3 Completely Different Approaches

Other possible changes rather involve a change *of* than *in* the algorithm. Methods listed in Section 3 could be checked again for their applicability.

- Smith's [Smi99] Wavelet spectrograms, may, in combination with adapted image processing, also be applicable in a real-time context and fare better in the face of noise than expected.
- Similarly to Cemgil's and Kappen's work [CK03], setting up an grid of expected bulge positions, or a probability map of such, might clear up some noise in the movement detection.
- Gouyon's and Herrera's high-level auto-correlation [GH03] may help finding from one metric level to the next. Considering that DanSense currently only considers one metric subdivision, the beat level, this is almost an entry for the next group. Also, implausible impulses could be dismissed and equidistant groups of them supported if their cardinality equals the metric quotient.
- Consistent phase differences between limbs may even give hints on the underlying rhythm, rather than making its detection difficult.

7.2.4 Concept Changes

Apart from the already mentioned increase of metric hierarchies, other fundamental changes could include:

- Deriving rhythms for separate channels and later combine them either by addition or by building some kind of product, depending on their phase difference.
- Not demanding a rhythm's conceptual impulses to occur on the metric grid, but impartially look for arbitrary clusters and represent each using one conceptual impulse.
- Deriving several rhythms whenever a single one seems not to fit well to channels otherwise appearing to not be dominated by noise. If a leg is moving with a different metric quotient or measure length than an arm, they would currently agree on one of them, if they are multiples of each other, or consistently sabotage each other's detection.

7.2.5 Extensions to the Original Aim

As already mentioned in the introduction, the rhythmic detection around which this thesis is centered was intended as an intermediate step to derive music from rhythm. Another one was the classification of rhythms according to a database of more or less famous named rhythms and their features. As noted by Smith [Smi99, p. 86], such databases already exist, but using automatic clustering and similar machine learning techniques, they could also be automatically generated by supplying an initial set of pieces from which to learn.

Other applications can be sought once Sense4Motion's delay of 200 ms is substantially reduced: real-time applications would become possible, like supplying phase information to Personal Orchestra in order to have direct control over the downbeat.

Furthermore, instead of relying on accelerometers, the input data could be replaced by any other input transformable into a series of impulses. Theoretically, any of the sensors mentioned in Section 3.1 could be used, provided the data reception and sample processing parts of DanSense are adapted. In this context, it would also help if the sending side (or the respective proxy program) sent some semantic information on their structure: for example, the used acceleration sensors could specify their axes as such, instead of merely providing a number of signals that might or might not be related to each other.

A Methods of Frequency Analysis

A.1 Fourier Transformation and the Frequency Domain

Any real-valued, continuous and periodic function can, as stated by the French mathematician Joseph Fourier, be represented as a constant term plus a linear combination of simple sine and cosine waves of various periodicities. Given such a function, knowing the factors of this linear combination means knowing about the relative prominence of frequencies in the given function. Instead of, for each frequency, specifying the sine's and the cosine's coefficients, the same information can also be given as a single sine's (or cosine's) magnitude and a phase.

In the example of accelerometer signals, one would hope to find the beat interval and measure length of the rhythm underlying the movement among the, magnitude-wise, most prominent frequencies. Such signals differ from the mentioned conditions in two ways: they are not continuous and not periodic.

Periodicity is conceptually assumed and only a finite part of the signal considered. In order to prevent frequency content from unintentionally emerging due to a certain misfit of the signal's end and its beginning, various windowing functions are used that dampen these ends, making the transition smooth. In DanSense, the *Hann* function (see Fig. A.1) is used.

Discretely sampled and thus non-continuous signals can be analysed for frequency content as well, using the *Discrete Fourier Transformation*

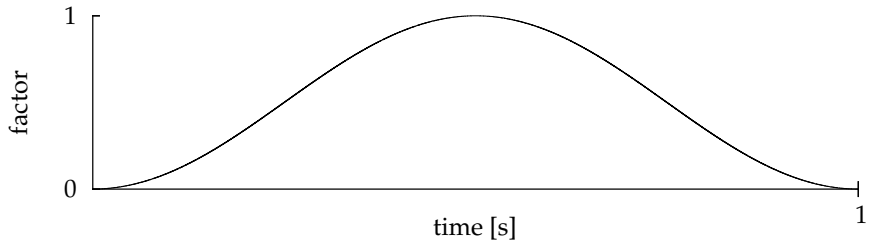

Figure A.1: The Hann windowing function. In this plot, the horizontal axis extends over the interval $[0, 1]$, but when applied to a specific signal window, it needs to be stretched to cover all samples, e.g., from 1 to n. The function, the plotted equation being $y(x) = \frac{1}{2}(1 - \cos(2\pi x))$, is named after Julius von Hann, an Austrian meteorologist.

(DFT)[1]. While the name correctly suggests the existence of a continuous variant of this transformation, it is not relevant for this thesis and thus not further treated.

Given a finite signal with real-valued samples $x_0, \ldots, x_{n-1}$, the DFT yields the sine and cosine coefficients as the real and imaginary parts of complex numbers[2] $X_0, \ldots, X_{n-1}$ by the following formula:

$$X_k = \sum_{t=0}^{n-1} x_t e^{-2\pi i k \frac{t}{n}}, \quad \forall k = 0, \ldots, n-1.$$

The desired information about frequencies' prominence in a signal can be obtained by applying the DFT to it and then computing the absolute value of each X_k.

Which frequency components are represented by X_k depends on the sampling frequency. Be that f, then X_0 represents 0 Hz and $X_{0.5(n-1)}$ represents $\frac{1}{2}f$, the frequences in-between being equidistantly distributed among those X_k with $0 < k < \frac{1}{2}(n-1)$. The remaining half of X mirrors the first.

There is an inverse to the DFT, the *iDFT*:

$$x_t = \frac{1}{n} \sum_{k=0}^{n-1} X_k e^{2\pi i k \frac{k}{n}}, \quad \forall t = 0, \ldots, n-1.$$

[1] DFT is often used as an abbreviation for *Discrete Fourier Transform*, but according to the Oxford English Dictionary, a "transform" is not a transformation, but the result of its application.

[2] Generally, the input signal can be complex itself, but this case is not relevant in the given context.

Its existence makes it possible to convert a signal into its frequency representation using the DFT, changing that (in the *frequency domain*) and finally converting the result back using the iDFT. One application for this is the dampening of certain frequencies by multiplying the frequency-representation coefficients by which they are represented with small factors.

A.1.1 Fast Fourier Transformation

The *Fast Fourier Transformation* (FFT) is a quick method of computing the DFT for signals of a length (or period) that is, in the best case, a power of two.[3] The central idea is to divide the DFT sum into two:

$$\begin{aligned} X_k &= \sum_{t=0}^{n-1} x_t e^{-2\pi i k \frac{t}{n}} \\ &= \sum_{t=0}^{\frac{n}{2}-1} x_{2t} e^{-2\pi i k \frac{2t}{n}} + \sum_{t=0}^{\frac{n}{2}-1} x_{2t+1} e^{-2\pi i k \frac{2t+1}{n}}. \end{aligned}$$

Those two sums, again, each compute a signal's DFT: of the even and odd positions in the original signal, respectively. So the procedure can be applied again and again, as long as the resulting halves are of even length. As opposed to the plain-DFT complexity of roughly $O(n^2)$, the FFT is computed in $O(n \log n)$.

A.1.2 Convolution

In Section 3.2.2 the term *convolution* was mentioned in connection with the smoothing of a rectified signal using the Hann windowing function. The convoluted signal $f * g$ of two signals f and g is defined as

$$(f * g)_t = \sum_{k=-\infty}^{\infty} f_k g_{t-k}.$$

Here, the index t specifies the (time) position within the signal. The sum can actually be computed within finite bounds in case one of the signals is non-zero over a finite interval.

Be f now a discrete realization of the Hann windowing function, re-centered to peak at $f_0 = 1$ and being $f_t = 0$ for all $t \notin [-5, 5]$. Then, for each sample g_t of the signal g, the samples $g_{t-5}, \ldots, g_{t+5}$ would be multiplied in pairs by $f_{-5}, \ldots, f_5$ and summed to yield $(f * g)_t$. Due

[3] To be precise, this is the simplest among a whole range of FFT algorithms.

to the hill-like nature of the Hann function, the resulting value is an average of f around f_t. Doing this for every t, a smoothed version of f as a whole results.

Convolving two signals is, with the exception of a constant factor, equivalent to applying the iDFT to the component-wise product of their individual DFT transforms. Using the FFT, the computational effort for convolution can thus be considerably reduced from $O(n^2)$ to $O(n \log n)$.

A.1.3 Comb Filters

In the processing of discrete signals, a filter is a function transforming one signal to another. A comb filter can be imagined as adding echoes of a signal to itself. Generally, such a filter's output signal $y_1, \ldots, y_n$ results from the input signal $x_1, \ldots, x_n$ as follows:

$$y_i = ax_i + bx_{i-t} + cy_{i-t},$$

where a, b and c are parameters and t is, put simply, the echo interval.

Fig. A.2 shows a graphical example and thus motivates the filter's name. Mathematically, the application of a comb filter is a convolution of the signal with the spiky comb signal. Like any other convolution, its application can be sped up by making use of temporary conversion into the time domain using the FFT.

A.2 Auto-Correlation

Formally, for a signal with samples $s_1, \ldots, s_n$, the auto-correlation function A is defined as

$$A_j(s) = \sum_{i=1,\ldots,\frac{n}{2}} (s_i \cdot s_{i+j})$$

for all values $j = 1, \ldots, \frac{n}{2}$. In the resulting array A, all positions j represent a time lag of $j \cdot T$, T being the sampling interval. If the array's value at a position is high, it means that the signal is similar to itself when shifted by $j \cdot T$. For example, for a signal sampled at $T = 5\,\text{ms}$ of a piece with a rhythm that repeats every second, the value at $j = 200$ should stand out.

If the signal is considered infinite (i.e., at its end to loop back to its beginning) then one can check even larger lag candidates $j = 0, \ldots, n-1$ using

$$A_j(s) = \sum_{i=0,\ldots,n-1} (s_i \cdot s_{(i+j) \bmod n}).$$

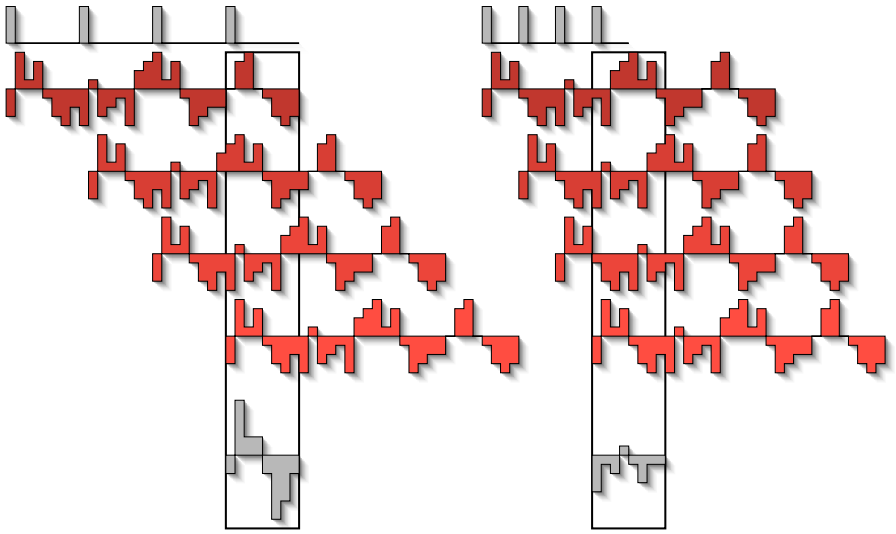

Figure A.2: Application of a comb filter. In the top row, two comb filters' representations are displayed: one with an interval of eight, one with an interval of four samples. Below, the given signal is shown, copied and shifted for each of the (exemplarily, four) spikes. Assuming the signals repeat infinitely often, the framed section is representative for the filter's application. At its bottom, the summed signal of the framed components is given, showing that, judging by the summed signal's energy (meaning the sum of its values' squares), the interval of eight samples is more prominent than that of four.

If the signal is not really infinite, a windowing function might have to be applied beforehand in order not to have artifacts from the differences between the signal's start and end, as discussed at the beginning of this chapter.

The measure of self-similarity is relative across the auto-correlation function's values. As samples are multiplied, a constantly zero-valued signal, for example, yields an auto-correlation of zero for all lags.

B Introduction to Max/MSP

Max/MSP can be described as a rapid prototyping environment for the processing of

- symbolic (i.e., MIDI) music,
- audio signals via its package *MSP* and
- video signals via its package *Jitter*.

In the environment's main user interface, data sources, processing elements and output interfaces, all called *objects*, are graphically represented by boxes. Each object can have multiple connectors called *inlets* and *outlets*, through which it can be connected to others. An example graph of objects and their connections, called a *patch*, can be seen in Fig. B.1. It is created by choosing objects from a palette or entering their name in empty boxes.

Besides those objects natively available, new ones can be added as newly programmed *externals*, using Java or C. Because of this, the above list of Max/MSP's fields of application is actually obsolete, because with the right objects, it could do anything. Of course, the rapid-prototyping aspect is lost once changes are primarily applied by changing and recompiling code instead of drawing connections between boxes with a mouse. Meanwhile, though, numerous third-party suppliers offer externals with functionality that is not natively supported by Max/MSP, making the feature as such an advantage even without ever intending to program a single line of code.

Also, the environment is fast enough to make patches useful beyond the design phase. By its possibility to define inlets and outlets for whole patches, they can be used like any other object in other patches. This modularity makes the design of highly complex processing graphs possible without getting lost in visual chaos.

Besides the mentioned extensibility through external objects, data can, for example, be input using manual controls on the patch, by reading

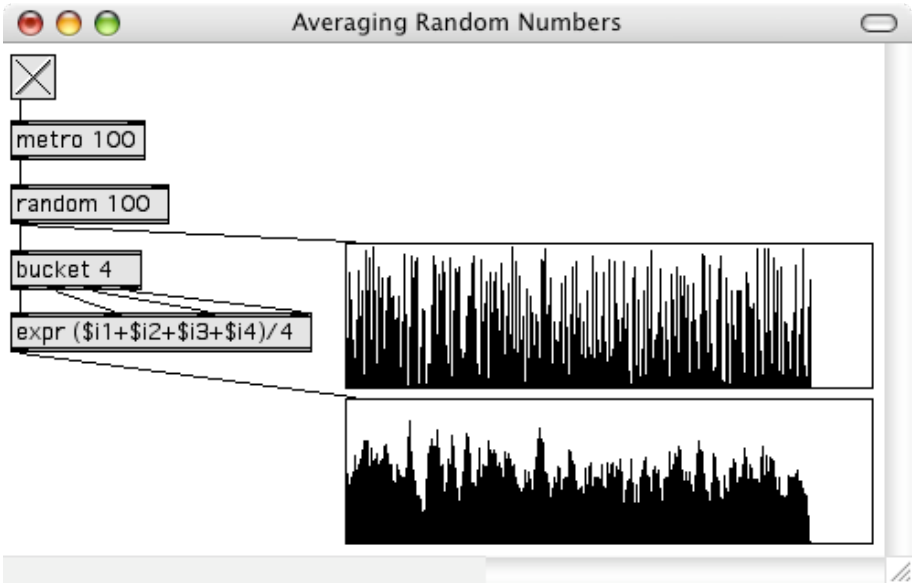

Figure B.1: Example of a Max/MSP patch. This patch displays a sequence of random numbers as well as a moving average graphically. The small check box in the top left sends out a the number one when being activated and zero on deactivation. The *metro* object is also activated and deactivated by receiving these numbers. In its active state, it sends out Max/MSP's standard signal, a *bang*, every 100 ms as specified by its parameter. Each time this signal is sent, it reaches *random* and causes it to send out a random number from zero to just below that given as parameter, in this case in the range to 99. This number is first sent out to the right, to be graphed alongside old values by the unlabelled *multislider* object. On the left, it reaches the *bucket* object, which remembers as many numbers as its parameter says, here four. On reception of a new value, it sends out the stored values (oldest on the right and first), forgets the oldest and remembers the received one as the newest. Thus, in each step, four successive random numbers reach the *expr* object, which stores them and, on receiving a value in its leftmost inlet, evaluates the mathematical expression given as parameter and sends out its result. The expression's variables resemble the inlets, from left to right, and "i" indicates integer inputs. The computed value (the numbers' average) is graphed by another multislider for visual comparison.

files or from system devices like a microphone or video camera. Output can occur to number boxes, dynamic graphs, geometrical arrangements, as sound, or also to files. Processing objects natively available include such for mathematical functions, list manipulation, control structures, MIDI note generation, audio filtering and image processing.

Glossary

auto-correlation	The notion of comparing a signal with lagged versions of itself.
beat	The dominant regular base pulse in an impulse series.
beat interval	The inverse of a beat's frequency.
bpm	Beats per minute, a measure for a musical piece's tempo that usually equals the number of quarter notes played in one minute.
comb filter	A signal filter that, put plainly, adds echoes of the signal to itself.
downbeat	The position on the metric grid's beat level that starts a measure.
FFT	Fast Fourier Transformation; a method of converting a discretely sampled signal into its frequency representation.
III	Inter-impulse interval; the time passing between two, not necessarily successive, impulses.
impulse	An accented event in time.
measure	One among a rhythm's repeating instances.
measure length	The duration of each of a rhythm's repeating instances.
metric quotient	The quotient of a rhythm's measure length and its beat interval.
MIDI	Musical Instruments Digital Interface; a standardized protocol defining the representation of notes and other musical events for digital transmission, as well as the physical interface over which this takes place.
rhythm	A repeating series of accentuations of and intervals between impulses.

Bibliography

[Aba96] Frank R. Abate (ed.), *The Oxford Dictionary and Thesaurus*, Oxford University Press, USA, 1996.

[BGK+04] Michael Barry, Jürg Gutknecht, Irena Kulka, Paul Lukowicz and Thomas Stricker, *Multimedial Enhancement of a Butoh Dance Performance*, 2nd International Conference on Advances in Mobile Multimedia, Bali, 2004.
csn.umit.at/publications/MOMM04.pdf

[Bon00] Bert Bongers, *Physical Interfaces in the Electronic Arts*, in *Trends in Gestural Control of Music*, IRCAM, Paris, 2000.
www.mat.ucsb.edu/594O/PhysicalInteractionBongers.pdf

[Bro93] Judith C. Brown, *Determination of the Meter of Musical Scores by Autocorrelation*, Journal of the Acoustical Society of America **94** (1993), no. 4, 1953–1957.
www.wellesley.edu/Physics/brown/pubs/meterACv94P1953-P1957.pdf

[CK03] Ali Taylan Cemgil and Bert Kappen, *Monte Carlo Methods for Tempo Tracking and Rhythm Quantization*, Journal of Artificial Intelligence Research **18** (2003), 45–81.
www.cs.cmu.edu/afs/cs/project/jair/pub/volume18/cemgil03a.pdf

[Cla85] E. F. Clarke, *Structure and Expression in Rhythmic Performance*, ch. 9, pp. 209–237, Academic Press, London, 1985.

[CMS01] Patricio de la Cuadra, Aaron Master and Craig Sapp, *Efficient Pitch Detection Techniques for Interactive Music*, International Computer Music Conference, La Habana, 2001.
ccrma.stanford.edu/~pdelac/research/MyPublishedPapers/icmc_2001-pitch_best.pdf

[Col04] Nick Collins, *Beat Induction and Rhythm Analysis for Live Audio Processing*, 1st-year Ph.D. report, 2004.
www.cus.cam.ac.uk/~nc272/papers/pdfs/report1.pdf

[CS89] Mark Coniglio and Dawn Stoppielo, *MidiDancer*, 1989.
www.troikaranch.org/mididancer.html

[DB03] Christopher Dobrian and Frédéric Bevilacqua, *Gestural Control of Music Using the Vicon 8 Motion Capture System*, Conference on New Interfaces for Musical Expression, Montréal, 2003.
music.arts.uci.edu/dobrian/motioncapture/NIME03DobrianBevilacqua.pdf

[DH86] W. J. Dowling and D. L. Harwood, *Music Cognition*, Academic Press, Orlando, 1986.

[Dix01] Simon Dixon, *Automatic Extraction of Tempo and Beat from Expressive Performances*, Journal of New Music Research **30** (2001), no. 1, 39–58.
www.ofai.at/~simon.dixon/pub/2001/jnmr.pdf

[dJ04] Leo de Jong, *Sensor Music Project*, 2004.
www.multipro.demon.nl/html/sensor_music_project.html

[Eck01] Douglas Eck, *A Positive-Evidence Model for Rhythmical Beat Induction*, Journal of New Music Research **30** (2001), no. 2, 187–200.
www.iro.umontreal.ca/~eckdoug/papers/2001_jnmr.pdf

[Fel02] Mark Feldmeier, *Large Group Musical Interaction using Disposable Wireless Motion Sensors*, Master's thesis, Massachusetts Institute of Technology, 2002.
www.media.mit.edu/resenv/pubs/theses/Feldmeier-SM.pdf

[GF98] Niall Griffith and Mikael Fernström, *Litefoot – A Floor Space for Recording Dance and Controlling Media*, International Computer Music Conference, San Francisco, 1998.
www.idc.ul.ie/data/publications/4_litefoot.pdf

[GH03] Fabien Gouyon and Perfecto Herrera, *Determination of the Meter of Musical Audio Signals*, 114th Convention of the Audio Engineering Society, Audio Engineering Society, 2003.
www.iua.upf.es/mtg/publications/AES114-GouyonEtAl.pdf

[Gue05] Carlos Guedes, *Mapping Movement to Musical Rhythm: A Study in Interactive Dance*, Ph.D. thesis, New York University, 2005.
homepage.mac.com/WebObjects/FileSharing.woa/wa/MappingMovement.pdf.pdf-zip.zip?a=downloadFile&user=carlosguedes&path=.Public/MappingMovement.pdf

[HCV+03] Dennis Hromin, Michael Chladil, Natalie Vanatta, David Naumann, Susanne Wetzel, Farooq Anjum and Ravi Jain, *CodeBlue: A Bluetooth Interactive Dance Club System*, IEEE Globecom, San Francisco, 2003.
www.cs.stevens.edu/~swetzel/publications/codeblue.pdf

[JA03] Kristoffer Jensen and Tue Haste Andersen, *Real-Time Beat Estimation using Feature Extraction*, Computer Music Modeling and Retrieval Symposium, Montpellier, 2003.

haste.dk/tue/articles/cmmr03-beat.pdf

[Joh91] Eric Johnstone, *A MIDI Foot Controller – The PodoBoard*, International Computer Music Conference, San Francisco, 1991, pp. 123–126.

[KG05] Doo Young Kwon and Markus Gross, *Combining Body Sensors and Visual Sensors for Motion Tracking*, International Conference on Advances in Computer Entertainment Technology, Valencia, ETH Zürich, 2005.
graphics.ethz.ch/~dkwon/downloads/publications/ace05/kwo05_ace.pdf

[Kop98] Naomi Koppel, *Translating Dance into Music: Developing an Abstract Representation of the Human Form*, 1998.
zoo.cs.yale.edu/classes/cs490/98-99a/koppel.naomi.naomi

[Les86] Joel Lester, *The Rhythms of Tonal Music*, Southern Illinois University Press, Carbondale, 1986.

[LKD+06] Eric Lee, Henning Kiel, Saskia Dedenbach, Ingo Grüll, Thorsten Karrer, Marius Wolf and Jan Borchers, *isymphony: An Adaptive Interactive Orchestral Conducting System for Digital Audio and Video Streams*, Conference on Human Factors in Computing Systems, Montréal, 2006.
media.informatik.rwth-aachen.de/materials/publications/lee2006b.pdf

[LMB+96] Geraldo Lima, Marcelo Maes, Márlio Bonfim, Marcus Lamar and Marcelo Wanderley, *Dance-Music Interface based on Ultrasound Sensors and Computers*, 3rd Brazilian Symposium of Computation and Music, Recife, 1996.
recherche.ircam.fr/equipes/analyse-synthese/wanderle/Gestes/Externe/sbcm96.pdf

[Mic05] Ulrich Michels, *dtv-Atlas Musik*, 2nd ed., Deutscher Taschenbuch-Verlag, 2005.

[MJ03] Ramia Mazé and Margot Jacobs, *Sonic City: Prototyping a Wearable Experience*, Proceedings of the 7th IEEE International Symposium on Wearable Computing, 2003.
www.tii.se/reform/results/publications_2003/2003_iswc.pdf

[Ng04] Kia C. Ng, *Music via Motion*, Proceedings of the IEEE **92** (2004), no. 4, 645–655.
www.media.aau.dk/people/sts/mapping/kiang.pdf

[Par94] Richard Parncutt, *A Perceptual Model of Pulse Salience and Metrical Accent in Musical Rhythm*, Music Perception **11** (1994), 453.

[PCS06] Chulsung Park, Pai H. Chou and Yicun Sun, *A Wearable Wireless Sensor Platform for Interactive Art Performances*, IEEE Conference on Pervasive Computing and Communications, Pisa, University of Californie, Irvine, 2006.
www.ece.uci.edu/~chou/eco-percom06.pdf

[PHBT00] Joseph Paradiso, Kai-Yuh Hsiao, Ari Benbasat and Zoe Teegarden, *Design and Implementation of Expressive Footwear*, IBM Systems Journal **39** (2000), no. 3/4.
researchweb.watson.ibm.com/journal/sj/393/part1/paradiso.pdf

[PHS+00] Joseph Paradiso, Kai-Yuh Hsiao, Joshua Strickon, Joshua Lifton and Ari Adler, *Sensor Systems for Interactive Surfaces*, IBM Systems Journal **39** (2000), no. 3/4, 892–914.
researchweb.watson.ibm.com/journal/sj/393/part3/paradiso.pdf

[PKM95] Russell Pinkston, Jim Kerkhoff and Mark McQuilken, *A Touch-Sensitive Dance Floor / MIDI Controller*, International Computer Music Conference, San Francisco, 1995, pp. 224–224.

[SBQK05] Prashant Srinivasan, David Birchfield, Gang Qian and Assegid Kidané, *A Pressure-Sensing Floor for Interactive Media Applications*, International Conference on Advances in Computer Entertainment Technology, Valencia, Arizona State University, 2005.
ame2.asu.edu/projects/floor/papers/srinivasan_ace122.pdf

[Sch98] Eric D. Scheirer, *Tempo and Beat Analysis of Acoustic Musical Signals*, Journal of the Acoustical Society of America **103** (1998), no. 1, 588–601.
web.media.mit.edu/~eds/beat.pdf

[Sep01] Jarno Seppänen, *Computational Models of Musical Meter Recognition*, Master's thesis, Tampere University of Technology, 2001.
www.cs.tut.fi/sgn/arg/music/jams/waspaa2001.pdf

[Smi99] Leigh M. Smith, *A Multiresolution Time–Frequency Analysis and Interpretation of Musical Rhythm*, Ph.D. thesis, University of Western Australia, 1999.
www.leighsmith.com/Research/Papers/MultiresRhythm.pdf

[Woo51] Herbert Woodrow, *Time Perception*, ch. 32, pp. 1224–1236, Wiley and Sons, New York, 1951.

Index

[This version incorporates error corrections and was compiled on September 15, 2007.]

Neben den im vorangehenden Text genannten Hilfsmitteln fanden selbstverständlich Hard- und Anwendungssoftware sowie Recherchen Anwendung, sonst keine. Zitate und sonstige Entnahmen fremder Quellen sind als solche kenntlich gemacht; mit ihrer Ausnahme habe ich die vorliegende Diplomarbeit selbständig verfasst.

Aachen, am 31. August 2006

(Urs Enke)