Aalto University
School of Electrical Engineering

Sakari Bergen

Conductor Follower:

Controlling sample-based synthesis with expressive gestural input

Master's Thesis
Espoo, December 12, 2012

Supervisor: Professor Tapio Lokki
Instructor: Professor Tapio Lokki

Aalto University
School of Electrical Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Sakari Bergen

Title:
Conductor Follower: Controlling sample-based synthesis with expressive gestural input

Date: December 12, 2012 **Pages:** 14 + 63

Department of Signal Processing and Acoustics
Professorship: Acoustics and Audio Signal Processing **Code:** S-89

Supervisor: Professor Tapio Lokki
Instructor: Professor Tapio Lokki

Over the years, several systems that follow conductor movement and play back a score accordingly have been implemented. Since conducting is essentially communication between the conductor and the musicians – including more than only gestural communication – it is clear that the problem is very complex to master completely. Recent developments in the computer gaming industry have made consumer grade motion sensing devices commonly available and easy to use in software development, making such a device an intriguing choice for using as the input of a conductor follower system. A system using such a device together with a high quality sample based synthesis engine was designed and implemented. These building blocks provide a solid foundation for a natural user experience, and have not been previously used in a similar way.

The result is a VST plugin which uses MIDI scores as input, and may be configured to work with any sample based synthesis engine supporting patch switching via keyswitch events. The system was evaluated by a professional conductor, and while the expressive features of the system are somewhat limited, the tempo following functionality was concluded to resemble a real orchestra when adjusted correctly.

Keywords: conductor follower, motion capture, musical gestures, score following, expressive synthesis

Language: English

Aalto-yliopisto
Sähkötekniikan korkeakoulu

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Sakari Bergen		
Työn nimi:	Kapellimestariseuraaja: Samplepohjaisen synteesin ohjaus ekspressiivisillä eleillä		
Päiväys:	12. joulukuuta 2012	**Sivumäärä:**	14 + 63
Signaalinkäsittelyn ja akustiikan laitos			
Professuuri:	Akustiikka ja äänenkäsittelytekniikka	**Koodi:**	S-89
Valvoja:	Professori Tapio Lokki		
Ohjaaja:	Professori Tapio Lokki		

Vuosien mittaan on kehitetty useita järjestelmiä, jotka seuraavat kapellimestarin liikkeitä, ja soittavat musiikkia keräämänsä tiedon perusteella. Koska orkesterinjohto on pohjimmiltaan kommunikaatiota kapellimestarin ja muusikkojen välillä – sisältäen muutakin kuin eleiden kautta kommunikointia – on selvää, että ongelman täydellinen hallinta on hyvin monimutkaista. Tietokonepeliteollisuuden viimeaikaisen kehityksen myötä kuluttajatason liikesensoreita on helposti saatavilla, ja niille on helppo kehittää ohjelmistoja. Siksi ne ovat hyvin varteenotettava vaihtoehto kapellimestariseuraajan syötteen tuottamiseksi. Tässä tutkimuksessa suunniteltiin ja toteutettiin kyseisenlaista liikesensoria, sekä korkealaatuista samplepohjaista syntetisaattoria käyttävä kapellimestariseuraaja. Nämä komponentit antavat hyvän perustan luonnolliselle käyttäjäkokemukselle, eikä niitä ole aikaisemmin käytetty vastaavanlaisissa sovelluksissa.

Työn tulos on VST-liitännäinen, joka käyttää MIDI-tiedostoja syötteenään. Oikeilla asetuksilla liitännäistä on mahdollista käyttää lähes minkä tahansa samplepohjaisen syntetisaattorin kanssa. Järjestelmän toimintaa arvioitiin ammattikapellimestarin avustuksella, ja vaikka järjestelmän ekspressiiviset ominaisuudet ovat jokseenkin rajoittuneet, sopivilla asetuksilla sen temponseuraamisominaisuuksien todettiin muistuttavan oikeaa orkesteria.

Asiasanat:	kapellimestariseuraaja, liikeenseuranta, musiikilliset eleet, partituurin seuranta, ekspressiivinen synteesi
Kieli:	Englanti

Acknowledgements

The work for this thesis was done as a research assistant within the Virtual Acoustics research team at the Department of Media Technology in Aalto University School of Science, during May–December 2012. I would like to express my gratitude to Professor Tapio Lokki for making the project possible and supporting my work, Conductor Sasha Mäkilä for his invaluable feedback and ideas, and my fellow employees at the Department of Media Technology for contributing to a pleasant working environment. I would also like to thank the open source community for providing me with all the awesome tools that made this project possible. The research leading to these results has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no. [203636].

Espoo, December 12, 2012

Sakari Bergen

Contents

Abbreviations and Acronyms

API	Application Programming Interface
DSL	Domain Specific Language
HCI	Human-Computer Interaction
IEEE	Institute of Electrical and Electronics Engineers
MIDI	Musical Instrument Digital Interface
NI	Natural Interaction
NUI	Natural User Interface
RT	Real-Time (computing)
UI	User Interface
VST	Virtual Studio Technology: A virtual instrument and effect plugin architecture created by Steinberg GmbH
VSL	Vienna Symphonic Library

Notation

$\langle \alpha, \beta, \gamma \rangle$	Denotes a tuple containing the values α, β, and γ.
$T := \langle a, b, c \rangle$	Defines the tuple type T, which consists of the variables a, b, and c.
$\{\alpha, \beta, \gamma\}$	Denotes the set containing the values α, β, and γ.
$T \in \{\alpha, \beta, \gamma\}$	Defines the type T, which may only contain values from the set $\{\alpha, \beta, \gamma\}$.
$\mathbf{a}_i$	Denotes the i^{th} element in the array $\mathbf{a}$.
$\text{size}(\mathbf{a})$	Denotes the size of array $\mathbf{a}$.
$\mathcal{F} : D_1 \to D_2$	Denotes that the function $\mathcal{F}$ maps values from domain D_1 to domain D_2.
$\alpha \pmod{\beta}$	Denotes the floating point modulus operation. That is, the value $\alpha - N\beta$ for an integer N such that the magnitude of the result is less than β.
$\lfloor \alpha \rfloor$	Denotes the floor function of α. That is, the largest integer smaller than or equal to α.
$\text{sgn}(\alpha)$	Denotes the signum function of α.
$]\alpha, \beta[$	Denotes an open interval from α to β.
$[\alpha, \beta]$	Denotes a closed interval from α to β.
$]\alpha, \beta]$ and $[\alpha, \beta[$	Denote half-open intervals from α to β.

Chapter 1

Introduction

The interaction between conductors and musicians is a sophisticated form of non-verbal communication. While the expressions used in conducting have no strict rules, most gestures performed by experienced conductors are understood by adequately experienced musicians. Modeling this complex conductor-musician interaction requires understanding the fundamental principles behind the communication. Once the relevant features are identified, it is possible to implement a system that follows the movements of a conductor by capturing and interpreting relevant data. By combining this conductor following with a musical score and a sound synthesis system, an interactive sonic experience can be created.

1.1 Conducting Gestures

While some forms of musical conducting have been around for hundreds of years [1], the developments that lead to the modern form of conducting were driven by the increasing size and complexity of symphonic scores in the late nineteenth century [2]. The task of the modern conductor is to mold an interpretation by guiding the musicians to play a musical score according to his or her vision. This is done not only by non-verbal gestures using body postures, hand movements, eye contact and facial expressions, but also using verbal instructions during rehearsals.

Studying conducting from a scientific, data based approach, is a rather recent development. Sousa [3] was among the first, performing a study in 1988, which investigated the use of musical conducting emblems, and their interpretation by instrumental performers. Sousa used videotaped conducting gestures, which he presented to university, high school and junior high school students, to study which gestures could be classified as conducting emblems,

non-verbal acts with precise meaning and a common interpretation among instrumental performers. He concluded that 38 out of the 55 studied gestures were recognized by over 70% of the subjects, with the recognition rate having a strong correlation with the experience level of the musicians.

While some conducting gestures can be easily analyzed, and do have a commonly recognized meaning, to really understand how the communication between conductors and musicians works, one has to study the whole life cycle of a conducted performance – preparations made by the conductor before rehearsing with the musicians, rehearsal with musicians, and finally the actual performance. Konttinen [4] studied conducting as a practical and sociological activity in her dissertation *Conducting Gestures*. She came to the conclusion that the main purpose of all conducting gestures become apparent in a social situation – a rehearsal, performance or conducting class – where the communicational situation includes both the gesture and the social context in which it is used. Therefore the meaning of conducting gestures are heavily influenced by their social context – the musicians affecting the way the conductor performs, and the conductor affecting the way the musicians interpret the gestures. Especially the rehearsal situation, where the conductor needs to make the meaning of his gestures obvious, is crucial for founding the basis for successful gestural communication.

1.2 Gesture-Based Human-Computer Interaction

When using gestures for human-computer interaction (HCI), the device used for gathering data plays a big role in the capabilities and limitations of the system. Early gesture systems were based on input from a camera or a specially made input device, such as a light-pen and screen combination [5]. Recent developments in technology – such as the popularization of touchscreen smartphones – have made gestures as a form of HCI a part of everyday life for an increasing amount of people. Judging by the over 600 IEEE conference publications related to gestures in 2011, it is obvious that gesture based systems are an active field of research.

Depth sensor equipped motion sensing input devices – such as the Mictosoft Kinect [6] – represent a recent trend of incorporating gesture based interaction into consumer electronics. Interaction with applications using these devices does not require touching or wearing any equipment, and thus applications have the potential to provide a very natural gestural user interface. However, the complexity of the subject lies in extracting meaningful information out

of the raw motion data provided by these devices. For surveys on related literature, see [7] and [8].

User interfaces that do not require an artificial input device and are based on models that occur as natural to users, are sometimes referred to as natural user interfaces (NUI), and the act of using such an interface Natural Interaction (NI). Most NUIs are built on top of an application that presents the user with rather abstract data, with no established set of gestures for operating on it available. A conductor follower, however, is quite the opposite of the usual case, as it is based on an already defined, albeit somewhat varying, set of gestures.

1.3 Tempo Following

One of the most fundamental musical parameters communicated to the orchestra by the conductor, is tempo. Tempo is not, however, a simple parameter to follow in a musically pleasing way. To understand the concepts related to tempo following, one has to look at two separate problems: conductor-performer synchronization, and expressive timing. Furthermore, since tempo is essentially the first derivative of position, and both are communicated via the same means – by conducting beats – synchronizing both tempo and position is not trivial.

Beat synchronization has been studied in both laboratory [9, 10] and musical rehearsal situations [11]. These studies provide insight into beat induction, i.e. what features are used to derive beats from continuous motion. The studies conducted in rehearsal situations have shown that realistic conducting situations differ from laboratory beat induction tests. However, these studies have been rather limited, and do not provide any insight on how absolute tempo or tempo changes affect the synchronization.

Expressive timing refers to the mostly subtle differences in timing that occur over phrases. These changes are used by the musician as a means of expression. While the average tempo may be steady across a piece of music, the length of a transcribed note duration (e.g. quarter note) differs depending on its position in the score. This minute variance in timing has been studied, and many studies have concluded, that simple parabolic *tempo curves* are not sufficient for representing expressive timing [12, 13]. It has also been discovered, that expressive timing is dependent on the absolute tempo [12, 14], and is thus in no way trivial to model.

Several systems for producing expressive timing – based on rule sets, machine learning, or statistical models – have been developed. A comprehensive overview of such systems can be found in [13]. These systems aim at adding

tempo and loudness variation to raw notation data, in order to make it sound more human-like. Given that the tempo in the performance does not differ much from the notated tempo, these features can be added to a score file beforehand.

1.4 Expressive Sound Synthesis

Music is fundamentally a form of human expression, and thus expressiveness a fundamental property of music. Therefore one could argue that the ultimate goal of sound synthesis should be to support the expressive motives of the composer and performer of the music. However, most control parameters available in sound synthesis systems are not related to the expressive properties of the produced sound, but are rather based on the time-frequency properties of the signal. Various studies have been made on the relation between descriptive and emotional concepts, and the time-frequency properties of sound. The studies have covered both musical cues, such as tempo, loudness and articulation [15], and the timbre of individual notes [16]. While these studies do present clear results, the largest problem in applying them in a conductor follower system is deriving the emotional descriptors from the gestures of the conductor.

The majority of motion based expressive sound synthesis implementations have been novel approaches, which do not model any existing form of musical expression. Some have concentrated on using emotional data extracted from the motion of the user, to control music in a way that the emotional features of the music would match those of the user [17], while others have provided means to create mappings between arbitrary motion data and sound features [18]. Because bringing out expressive features of music is a crucial part of conducting, a conductor follower is a rather natural candidate for an expressive sound synthesis system.

1.5 Thesis Overview

This thesis is based on the development of a conductor follower system, which uses motion capture and sound synthesis techniques to provide an interactive experience of conducting a virtual orchestra. An example setup of the final result is displayed in figure 1.1 for reference. Previous conductor follower systems, motion tracking methods, score following methods, and expressive synthesis methods are studied and analyzed, and a set of methods for an implementation is devised based on the study. These methods are used to implement the system, and further details emerging from the implementation

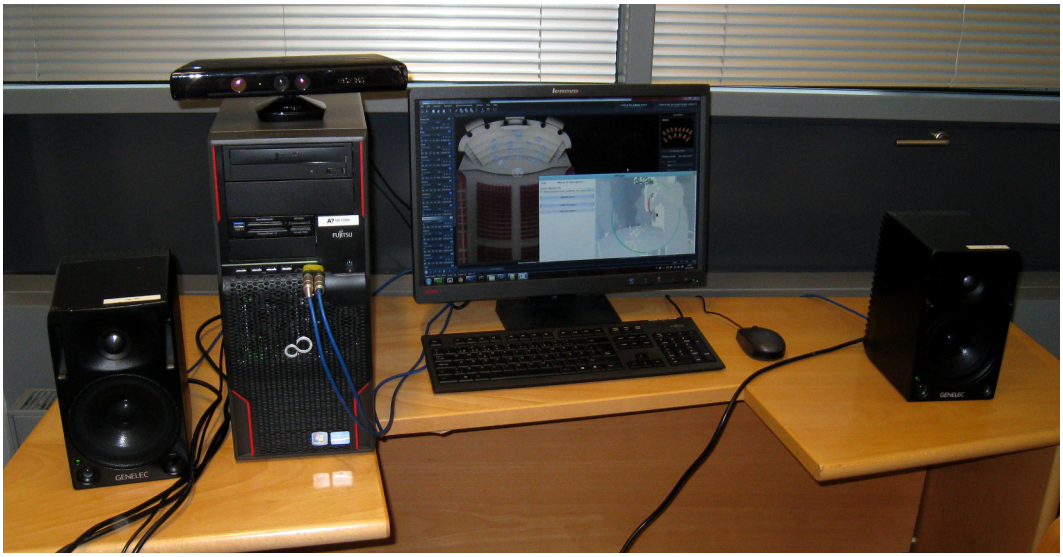

Figure 1.1: An example setup, including a set of speakers, a Microsoft Kinect, and a PC running Conductor Follower along with a synthesis environment, Vienna Ensemble [19].

are described and rationalized.

In this chapter the central concepts behind the work in this thesis are discussed. Chapter 2 discusses previous similar systems, and gives an overview on the current system. Chapter 3 describes the methods related to motion capture used for gesture and expressive feature detection. Chapter 4 formalizes the principles and defines the methods used for following the musical score. Chapter 5 discusses sound synthesis related design and implementation choices. Chapter 6 describes the visualization methods used in the system. Chapter 7 highlights the most relevant details in the implementation of the system. Finally, chapter 8 evaluates the results of the project, discusses recommended further work on the subject, and provides a conclusion of the thesis.

Chapter 2

Conductor Following

Over the years, several conductor follower systems for performance, educational and research use have been designed and implemented. These systems follow conductor gestures and produce or modify music based on the input. This chapter provides a brief overview of previous systems, and introduces the system to be implemented together with its design rationale.

2.1 Previous Work

Conductor following systems implemented in the past can be roughly classified based on four features: the types of sensors used for capturing gestures, the methods used for analyzing the gesture data, the level of control they provide, and the type of sound synthesis they use.

This section provides a brief overview on how these four features have been implemented in previous systems. As more comprehensive overviews can be found elsewhere – one of the latest and most detailed being [20] – the overview here is kept brief.

2.1.1 Sensors for Gesture Capture

The history of conductor following systems can be traced back to computer performance control systems, which allowed controlling score parameters during playback. While the earliest systems used control interfaces which do not resemble a conductor's baton [21], the first system that tracked a baton-like wand was developed in 1983 [22]. The system was able to analyze conducting gestures performed with the wand, and control the tempo and dynamics of a score. Since then, several similar systems have been developed, utilizing a multitude of tracking methods, including the following:

- systems that measure the capacitance between a baton and an "antenna",
- accelerometer based systems,
- magnetic sensors,
- infra red based systems,
- video analysis from a regular video stream, and
- direct measurement of the conductor's body movements with sensors attached to the conductor.

In addition to tracking the movements of the baton or hands, breath and gaze sensors have also been used. For examples of projects using each sensor type, the reader is asked to refer to [20].

Capturing the relevant data from the conductor's gestures provides the raw input to the system. While the quality and quantity of this data sets the baseline for what can be done in the later stages of processing, the method used for acquiring the data is very loosely coupled to the rest of the system.

2.1.2 Methods for Gesture Analysis

The methods used for analyzing the raw data provided by the sensors define the mode of interaction between the conductor and the system. The most fundamental differentiation between different systems can be made based on their mode of output: whether the method produces a set of continuously varying parameters, or a stream of discrete events based on detecting discrete gestures. Most systems combine both types of analysis methods. While it is possible to use heuristics to extract data [23, 24], also hidden Markov models (HMMs) [25–27] and artificial neural networks (ANNs) [23, 28] have been used. All these methods can be used to produce both discrete events and continuous values for parameters.

Since both hands have distinct roles in conducting, the methods used for gesture analysis also differ between the hands. Since the right hand is used for conducting the tempo and is in continuous motion, both continuous parameters – such as articulation and dynamics related features – and discrete events – such as beats, fermatas and beat pattern changes – can be extracted from its motion. The left hand, however, is usually used only for discrete gestures.

The features described above are all rather low level. However, it is possible to also extract higher level features from the conductor's movement. While the output of the analysis differs from the lower level feature extraction, the methods used for analysis are somewhat similar, using methods such as simple heuristics, HMMs or ANNs. The difference between these modes of control is discussed in more depth in the next section.

2.1.3 Performance Control

Fabiani et al. [20] define three levels of performance control: direct control, model-based performance control, and high-level control via semantic descriptors. The distinction between these modes is necessarily fuzzy; many systems use a combination of the three levels. In direct control systems, the user is in direct control of the performance parameters, such as tempo, dynamics, articulation, and instrument section balance. Model-based performance control combines the input from the conductor with a performance model. Performance models include e.g. changes in dynamics and tempo for achieving expressive phrasing. In model-based systems there is always a trade-off between stability and sensitivity – i.e. if much weight is put on the model, the system is not very sensitive to the input, while too much weight on the input can cause very large deviations from the model. Most of the previously implemented conductor following systems use a combination of direct and model based control.

The highest level of control utilizes semantic descriptors – such as emotional expressions – to control the performance. This requires mapping the descriptors to both motion features and performance features, which allows first translating motion features to descriptors and then applying relevant performance features based on the descriptors extracted. While studies have produced clear results for the relation between semantic descriptors and musical and tonal qualities [15, 16], the mapping of conducting gestures to semantic descriptors has not been studied as extensively.

2.1.4 Synthesis

Once the performance parameters have been generated based on the gestures of the user, they need to be applied to the score. The methods implemented in previous systems can be classified into two categories: synthesizing the output from a score file using a synthesizer, or applying modifications to a pre-recorded performance. While the former can obviously be implemented very flexibly using any synthesis method available, the number of options for implementing the latter is more restricted. The methods used for modifying pre-recorded performances include time-frequency manipulation and mixing different recordings [29].

2.2 Design Rationale

If the real life interaction between a conductor and musicians were to be modeled as closely as possible, it would require providing a feedback channel equiv-

alent to the verbal communication from conductor to musicians. This channel could then be used to teach the style of the specific conductor to the system. In other words, when the system misinterprets some gesture, the conductor could correct this interpretation by providing the correct interpretation via the feedback channel. This would make it possible for the system to learn the style of the conductor over time, using machine learning techniques. Implementing such a system would, however, be too large an effort for the scope of this project. Instead, a more static and general heuristics based system is considered and implemented.

The main goals of the project can be summarized with the following statements:

1. The system should be usable without wearing or holding external equipment.
2. The system should be usable by anyone familiar with conducting, without having to provide extensive instructions on usage.
3. The system should react to conducting like a real orchestra.
4. The system should produce sound that resembles a real orchestra.

Out of these goals, only the first one is easily evaluated, while the rest are more complex, at least point 3 requiring expert evaluation.

2.3 System Overview

The system can be broken down to four functional components: motion tracking, score following, sound synthesis and visualization. Since the synthesis is not implemented in this project, we need to communicate with the synthesis component via a third party application programming interface (API). For this reason, a plugin component is required. Since the plugin standard used in the project (discussed in more detail in section 5.1.1) provides a method of creating user interfaces (UIs), the visualization is part of the plugin. This leaves us with three modules to be implemented: the *Motion Tracker*, *Score Follower* and *Plugin*. A data flow oriented overview of the architecture is presented in figure 2.1.

The Motion Tracker takes depth sensor input, analyzes the body movements of the conductor, and produces a stream of events and a set of continuously varying motion parameters. These events and parameters describe those features of the conductor's movement, which are relevant to conductor following.

The output of the Motion Tracker is then analyzed for both tempo following and expressive synthesis purposes by the Score Follower. To do this, the

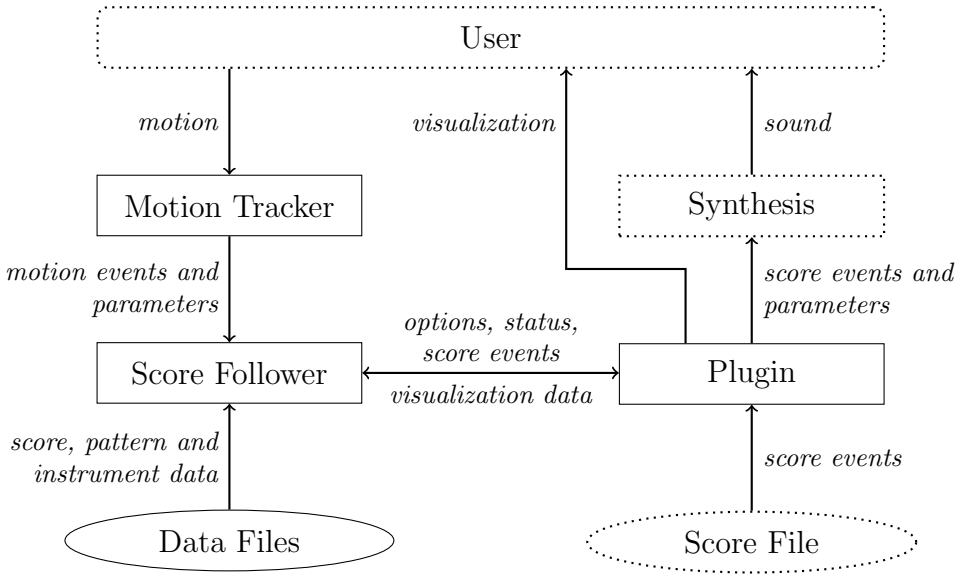

Figure 2.1: Data flow oriented architecture overview. Sharp-cornered rectangles depict software modules and ellipses depict data sources. The round-cornered rectangle is reserved for the user. A dotted outline implies that the item was not implemented or specified in this project.

Score Follower needs additional data, provided by the user via data files. This data includes score meta data, as well as information on the beat patterns and instruments used in synthesis. Since the score data format is closely related to the plugin standard used, the score data is provided by the Plugin, using an API provided by the Score Follower for abstraction. Once all the necessary data is available to the Score Follower, it can perform its actual task, which consists of three parts: following tempo, producing expressive synthesis parameters, and producing visualization data. The tempo following and expressive synthesis together produce a stream of score events, which are passed on to the Plugin, together with the visualization data. The Score Follower also exposes its status, and a set of options for the Plugin to use.

Finally, the Plugin passes on the score events and parameters to the synthesis component in the format required by the plugin API. It also produces the visualization and UI for adjusting the options and observing the status of the Score Follower.

Chapter 3

Motion Capture

Motion capture in the context of the conductor follower system, consists of tracking the right hand of the conductor. The methods used for this are fairly mature, and not in the main focus of this thesis. Since sufficient methods and implementations for motion capture using depth sensor devices are commonly available, there is no need to cover the details of those here.

This chapter introduces the methods used for extracting all relevant features from the motion data. This includes tracking the hands, and detecting beats and expressive features from the hand motion data.

3.1 OpenNI and Hand Tracking

OpenNI (for Open Natural Interaction), is a library with an open source API, which provides a ready implementation for tracking hands using depth sensor equipped motion tracking devices. While the API is open, the hand tracking method is proprietary. It provides a solid implementation with adjustable smoothing. OpenNI is discussed on an implementation level in Section 7.1.2.

3.2 Supporting Methods

Extracting meaningful information out of the motion data requires using many time related analysis methods. Features such as velocity and acceleration require inspecting the movement at a single point in time, while the cyclic nature of conducting requires averaging over time, in order to extract features that apply at the beat or bar level.

For the analysis at a single point, a polynomial regression filter is used. For averaging over time, methods such as exponential moving averages and peak holders are used. These averaging methods require selecting parameters for

the amount of averaging applied. When selecting these parameters, there is always a trade-off between the ability to react to fast changes and the amount of averaging provided.

3.2.1 The Polynomial Regression Filter

Luck and Toiviainen have studied the gestures of conductors both from a beat synchronization [11] and expressive [30] point of view. Both studies used polynomial regression filtering for data smoothing and differentiation. As these studies form the basis for a lot of the analysis used in this project, it is logical to use the same methods.

The filter uses polynomial fitting over a sliding window to approximate the movement as a polynomial at each point, using a least squares method. This approximation is applied to each spatial axis separately, using the points at times $t_{i-n} \dots t_{i+n}$, for analyzing the movement at time t_i, giving a filter length of $2n + 1$. Once the polynomial is approximated, it can be used for solving a smoothed value and all derivatives up to $2n$ at t_i.

The Polynomial Regression Peak Detector

As the polynomial regression filter provides the first derivative, it can be easily used to detect the presence of local minima and maxima

$$M \in \{\text{None, Dip, Peak}\}\,.$$

A change in direction can be detected as a change of sign in the first derivative:

$$m_i = \begin{cases} \text{None} & \text{if } \operatorname{sgn}\left(f'_{i-1}\right) = \operatorname{sgn}\left(f'_i\right) \\ \text{Dip} & \text{if } \operatorname{sgn}\left(f'_{i-1}\right) < \operatorname{sgn}\left(f'_i\right) \\ \text{Peak} & \text{if } \operatorname{sgn}\left(f'_{i-1}\right) > \operatorname{sgn}\left(f'_i\right), \end{cases} \tag{3.1}$$

where f'_i is the first derivative of the observed function at time i.

3.2.2 Exponential Moving Average

Compared to a simple moving average (SMA), the exponential moving average (EMA) gives two benefits in our use case:

- The EMA reacts to sudden changes faster than a SMA, while still offering good averaging.
- An EMA filter with a large amount smoothing is computationally much lighter compared to a SMA filter providing a similar amount of smoothing, as seen below.

The EMA can be effectively calculated recursively:

$$S_i = \begin{cases} y_i & \text{if } i = 1 \\ \alpha y_i + (1 - \alpha) S_{i-1} & \text{if } i > 1, \end{cases} \tag{3.2}$$

where $\alpha \in]0, 1[$ is the smoothing coefficient, and y_i the i^{th} observation. High values of α provide less averaging with faster reaction to changes.

3.2.3 Peak Holder

A peak holder simply holds the peak value

$$P_i = \max\left(y_i \dots y_{i-n}\right), \tag{3.3}$$

where y_i is the i^{th} observation, and n the length of the peak holder. High values of n provide a smoother output, but do not react to sudden decreases in magnitude as fast.

3.2.4 Standard Deviation of the Sample

The windowed standard deviation of the sample

$$S_N = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \overline{x})^2}, \tag{3.4}$$

where $x_1 \dots x_N$ are the N latest observed values, and $\overline{x}$ is the mean value of these observations. The output of the operation expresses the variability of the input values over a window of N samples.

3.3 Beat Detection

The aim of the beat detection system is to work with as many conducting styles as possible. The approach taken here is thus not based on specific beat patterns or gestures. Many previous systems have been based on using machine learning techniques with predefined beat patterns [20], but such systems would easily get confused with the large variance present in the style of many orchestral conductors. Instead, the approach is based on beat induction from the motion features of the conductor's hands.

Luck and Toiviainen [11] conducted a beat synchronization study, which was novel in the sense that it studied the subject in a real orchestral rehearsal

situation. Previous studies have been made primarily in laboratory settings, and thus do not present a good foundation for emulating the behavior of an orchestra. The study analyzed the motion of the baton in four excerpts of conducting, collected from one rehearsal situation. Since the study represented the conducting style of only one conductor, and the response of only one orchestra, it does not present universally valid results.

Studies have shown that the feature with the largest correlation with beats, is acceleration along the trajectory (a_t) [9, 10]. However, a good correlation between features and events does not necessarily mean the features can be used for detecting events trivially. Features with high correlation with beats – including a_t – were evaluated for their suitability for beat detection in the conductor follower system via simple experimentation. Initially the most promising feature was vertical velocity (v_y), which was noted as one of the most correlated features in [11]. a_t on the other hand, was found to have local minima around the beat, but also elsewhere. The timing of the local minima also changes along with the conducting style. If a system based on machine learning were to be made, the conducted experimentation would suggest that v_y and a_t would most likely be good input for the system.

Based on experiences with a trained conductor – both motion capture data analysis and discussions – a very simple solution for beat detection using only the vertical position of the hand was finally chosen. The method combines the theoretical position of the beat in any beat pattern, the vertical bottom, with the fact that musicians react to conducted beats with a lag. Once a vertical minimum is found in the hand movement (based on change of sign in v_y), its timestamp and location are recorded. Each new position that is not a new local minimum is checked for two criteria, using two different thresholds for the vertical position – the reset threshold and beat threshold:

1. If the vertical position has not increased more than the reset threshold from the minimum position, the timestamp for the minimum is reset. This is done to accommodate situations, where the conductor's hand stays relatively still after its bottom position, but has not yet indicated a beat.
2. If the vertical position has increased more than the beat threshold, and the time since the bottom position is larger than a tempo dependent value, a beat is detected.

3.4 Start Gesture Detection

The gesture for beginning playing the score is detected using basic motion features. First of all, a momentary immobile state of the right hand is required

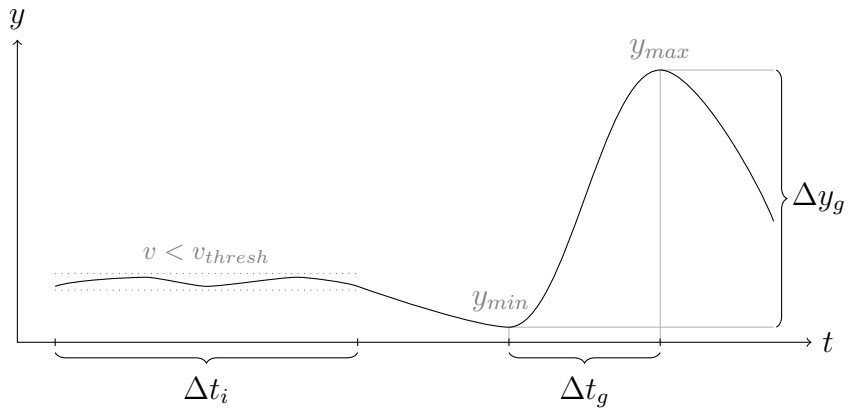

Figure 3.1: Start gesture parameters: immobile time (Δt_i), gesture amplitude (Δy_g), and gesture time (Δt_g).

before performing the actual start gesture. The immobile state is detected from the absolute speed (v), which must be below a threshold (v_{thresh}) for the hand to be considered as immobile. To qualify as sufficient, the immobile state has to be held for a while, and the entire start gesture has to be performed within a limited time after the immobile period. After the immobile period, the hand is required to make a vertical move, indicating the starting tempo with the vertical minimum and maximum of that move. The criteria for start gesture detection are concluded in figure 3.1 and in the list below:

- A long enough immobile state (Δt_i greater than threshold) has been detected not too long ago.
- A local minimum for the vertical position (y_{min}) has been detected after the immobile state.
- A local maximum for the vertical position (y_{max}) has been detected after the local minimum.
- The distance between the minimum and maximum (Δy_g) is large enough.
- The time between the minimum and maximum (Δt_g) is within a suitable range. This range is based on the start tempo estimation described in section 4.4.

3.5 Expressive Feature Extraction

Luck and Toiviainen [30] studied the correlation between the kinematic features of conductors' hands and the perceived magnitude of different expressions

in their movement. The studied expressions were *Expression*, *Valence*, *Activity* and *Power*. A point-light representation of two conductors' movements was presented to subjects, who rated the expressiveness on a continuous scale in both time and magnitude. The study showed correlations between kinematic features such as position, velocity, acceleration, and jerk[1] and the aforementioned expressions. However, the final conclusions of the study stated that the findings should probably not be applied as such, but that the most essential findings were that observers are indeed sensitive to more fine-grained kinematic features, and that increased amplitude, greater variance and higher speed of movement convey higher levels of expressivity.

Based on the results by Luck and Toiviainen [30], Sousa [3], and experimentation, a set of kinematic features was selected for extracting expressive features out of the motion data. These features are described in this section. When analyzing the motion features, the dimensionality of all variables (position, speed, acceleration, and jerk) is reduced to two dimensions, ignoring backward and forward movements. The reason for this can be found from the hand tracking algorithm, which often interprets changes in the shape or angle of the hand as forward or backward movements. These "false" movements produce unexpected results if included in the analysis.

3.5.1 Windowed Trajectory Length

The windowed trajectory length measures the overall amplitude of the hand movement over a given time window. The length of the window is set to one second, in order to give sufficient averaging over beat gestures. The trajectory length over a given time can also be interpreted as the average absolute speed over the same time window. However, using the positional information to derive this metric produces more accurate results, as the results of the polynomial regression analysis are more sensitive to noise in the data. In addition to the size of the conducting pattern being shown to be a good indicator of dynamics in [3], the velocity of the right hand showed a high correlation with *power* in [30]. Therefore, using the windowed trajectory length as an expressive feature is well justified.

3.5.2 Windowed Standard Deviation of Velocity

The windowed standard deviation of the absolute velocity ($S_N(v)$) measures how continuous the movement of the hand is. The value for N defines the measurement period, and is derived from an analysis time that provides sufficient

[1] Jerk is the third derivative of position, i.e. the derivative of acceleration.

averaging over beat gestures: about 700 milliseconds. While [30] doesn't include any results to back up using $S_N(v)$ as an expressive feature, [3] describes several conducting emblems related to the playing style on the staccato–legato axis, which include the variability of velocity as an integral component.

3.5.3 Filtered Peak Jerk

The values for jerk produced by the polynomial regression filter are rather sensitive to noise, and thus need more filtering than the other features. The absolute value of the two dimensional jerk is first filtered with an exponential moving average filter and subsequently fed to a peak holder. Due to the large variability in the absolute jerk, the hold time of the peak holder has to be large: around 1300 milliseconds. Right hand jerk was shown to have a strong correlation with *power* in [30]. Also, wen making *forceful* movements with the hands, as some conducting styles were described in [3], the filtered peak jerk has large values.

Chapter 4

Score Following

The task of score following in the context of a conductor follower system, means scheduling *score events* (notes) based on the *beat events* extracted from motion data. In essence, the task consists of establishing a mapping between wall clock time and the relative timestamps of score events. This mapping is based on the conducted tempo of the piece, which is formed based on score data and beat events.

The aim of the score following system is to behave like a real orchestra would, when reacting to a conductor. This implies that

1. The orchestra should not be confused by changes in the conducting pattern.
2. Extra beats in "unexpected" positions are not acted upon.
3. Sudden changes in tempo are not immediately followed.

Items 1 and 2 require that the tempo follower must have knowledge of all the possible conducting patterns for each time signature, and that it must be able to relate each beat to some beat in the pattern. In this chapter, we first formalize the different time bases used, and then deconstruct score following into three different tasks: beat classification, tempo adjustment and start tempo estimation.

4.1 Physical Time and Score Positions

When modifying the playback speed of a musical score that has tempo variations already build into it, it is important to precisely define the concepts of time used. Wall-clock time is the monotonically progressing concept of time most familiar to humans. It is the time counted by a regular clock, and is referred to as physical time, its dimension denoted by T and quantities with t.

Score positions, on the other hand, are slightly more complex. Musical events in a fully defined score have two time bases:

1. Musical time has the base unit of *beats*, but it also includes a time varying derived unit of *bars*. Any position in the score can thus be expressed as either a beat offset, or a combination of bars and beats. Its dimension is denoted with B and quantities with b.
2. Score time has the base unit of *seconds*. This is an offset from the beginning of the score if the original tempo is followed. Its dimension is denoted with Φ and quantities with φ.

A score position is thus defined as $P := \langle b, \varphi \rangle$, and variables are denoted with a p.

Mapping between physical time and score positions is denoted with the mapping operator $\mathcal{M} : T \to P$, and it's inverse $\mathcal{M}^{-1} : P \to T$.

The mapping operator (as well as its inverse) requires knowledge of the realized playback speed, and can thus be applied reliably only to past events. It is also possible to use the mapping operator for estimating future events. These estimates, however, may produce results that differ from future mapping operations, and may only be used in special cases.

The relation between physical time and score time is embedded in the mapping operator. To formalize the relation, we can represent the relation with a warp factor

$$w = \frac{\Delta\phi}{\Delta t}, \tag{4.1}$$

which is piecewise constant over time.

The model here is a simplified model suitable for the problem at hand. For a more musically oriented overview on the subject of score time and physical time, along with a formalization of tempo, see [31].

4.2 Beat Classification

Once the beats are detected as described in section 3.3, the beats have to be classified. A beat classification is defined as the tuple

$$C := \langle t, b, \Delta b \rangle ,$$

where t is the time of the beat, b the (uncorrected) position of the beat, and Δb the offset to the classified position. Each beat pattern has a related classification operation $\mathcal{C} : B \to C$, which uses the relative positions of the detected beats within the current bar, to produce a classification. Out of the set of classifications produced by the beat patterns, the classification with the best quality is selected as the final classification.

4.2.1 Beat Patterns

To interpret the beats of a conductor properly, knowledge of beat patterns has to be applied. Each time signature has a set of common conducting patterns, which may be varied within the piece being conducted. For example the 6/8 signature may be conducted placing beats on all six eighth notes, on the first and fourth eighth, or even only on the first.

The beat classification operation presented in section 4.2.2 attempts to not get confused by changes in beat patterns. As an example, a change from beating all six eighths in 6/8 to beating only the first and fourth eighth, the second beat in a bar could easily be mistakenly classified as a beat on the third or fifth eighth by a naïve method. The presented method should, however, be able to detect the pattern change as a more probable choice, producing a correct estimate.

4.2.2 The Beat Classification Operation

The classification operation $\mathcal{C}$, uses the positions of beats in a beat *pattern* ($\mathbf{p}$), and the *history* of the latest beats ($\mathbf{h}$), to classify the detected beats. To keep all the relevant beats in $\mathbf{h}$, the current bar is tracked continuously. When a new beat is to be classified, it is added to the end of $\mathbf{h}$, and all beats that have been classified as being in prior bars are removed. Next, the match quality for each beat pattern is assessed:

1. Let the quality of a match between a detected beat and a beat in a pattern be
$$q(i,j) = -|\mathbf{h}_i - \mathbf{p}_j|. \tag{4.2}$$
2. Let α be such that $q(1,\alpha)$ is maximized, i.e. the best match for the first beat.
3. For all other beats $\mathbf{h}_k$, $q(k,\alpha)$ and $q(k,\alpha+1)$ are evaluated, affecting the score in the following way:
   (a) If α exceeds the size $(\mathbf{p})$ in any quality evaluation
      - The wrapped value $\alpha' = \alpha \pmod{\text{size}(\mathbf{p})}$ is used instead of α.
      - The compensated value $\mathbf{p}'_j = \mathbf{p}_j + \left\lfloor \frac{\alpha}{\text{size}(\mathbf{p})} \right\rfloor l$, where l is the length of the bar, is used instead of $\mathbf{p}_j$.
   (b) If $q(k,\alpha) \leq q(k,\alpha+1)$
      - α is incremented, and the evaluation is repeated.
      - If this was not the first evaluation for the current value of k, a penalty for a missing beat is added to the total quality of the

match.

(c) If $q(k, \alpha) > q(k, \alpha + 1)$

- The best match has been found, and $q(k, \alpha)$ is added to the total quality of the match.
- If α has not yet been incremented, a penalty for duplicate beats is added to the total quality.

To achieve a better response to tempo changes, the evaluation of beat patterns is repeated using a range of stretch factors. The positions in $\mathbf{h}$ are adjusted with respect to the first beat in the array:

$$\mathbf{h}'_i = \mathbf{h}_i + s(\mathbf{h}_i - \mathbf{h}_1), \tag{4.3}$$

where $\mathbf{h}'$ is the corrected array, and s the stretch factor. To reduce the possibility for false positive tempo changes, the range of s is restricted to be close to unity, and a penalty proportional to the difference from unity is applied to the scores produced with stretch factors.

Once each beat pattern and stretch factor combination has been evaluated, the best match is selected for classifying previously unclassified beats. The classification method depends on the amount of beats detected for the current bar:

- The first beat in any bar is assumed to always be on the first beat. Thus, its position does not depend on the beat pattern of the current bar, and it can be classified based on the beats in the previous bar. This maximizes the number of beats used to estimate the beginning of each bar, and should thus also maximize the probability of a correct estimate.
- The first and second beats in a bar do not usually provide enough information about the beat pattern used in that bar. Therefore the second beat in a bar is only given an estimate if it scores better than its neighboring beats in the winning beat pattern by a given factor, i.e.

$$q(\beta, \gamma) > a\, q(\beta, \gamma \pm 1), \tag{4.4}$$

  where β is the beat being classified, γ the best estimate, and a the factor determining how much better the estimate has to be compared to its neighbors.
- Considering the number of beats in most beat patterns, three or more beats should give a good enough estimate (or else there is no hope of ever getting one). Therefore all beats after the second are always classified.

4.3 Tempo Adjustment

After a successful beat classification, the current position and tempo offsets are estimated. Based on these estimates, changes to the playback tempo are made, which attempt to correct the offsets so that the playback matches the conducting as well as possible. The change in tempo is realized by adjusting the warp factor w for each audio block (see section 7.3 for details).

4.3.1 Tempo Estimation

The instantaneous tempo estimate is calculated from the beat classifications as

$$v_i = \frac{(b - \Delta b) - (b' - \Delta b')}{t - t'}, \tag{4.5}$$

where b and Δb are the position and offset of the latest conducted beat, t the time of the latest conducted beat, and b', $\Delta b'$ and t' those of the previous conducted beat, respectively.

Since the beats are not uniformly spaced in time, a conventional moving average can not be used for filtering the instantaneous tempi. Instead, a linearly decaying time dependent filter is used. The filter is defined by its cut-off time t_{co}, which forms the limit for how old tempo estimates are taken into account. Each tempo estimate (v_i) not older than t_{co} is given a weight coefficient

$$c_i(\Delta t) = 1 - \frac{\Delta t}{t_{co}}, \tag{4.6}$$

where Δt is the absolute offset to the current time. The target tempo is then the weighted and normalized sum of all the estimates:

$$v_t = \frac{\sum c_i v_i}{\sum c_i}. \tag{4.7}$$

4.3.2 Tempo Function

The parameters Δb and v_t form the basis for tempo adjustment. The target of the adjustment is to be at the tempo v_t, and having made a position adjustment of Δb after a catchup time t_c. As can be seen, the operation requires adjusting two interlinked parameters, tempo and position, which is not possible using a linear function. Instead, the tempo change is built as the sum of two functions

$$v(t) = v_i + v_t(t) + v_b(t), \tag{4.8}$$

where v_i is the initial tempo, $v_t(t)$ is a linear function contributing to the total tempo change, and $v_b(t)$ a non-linear function contributing to the position

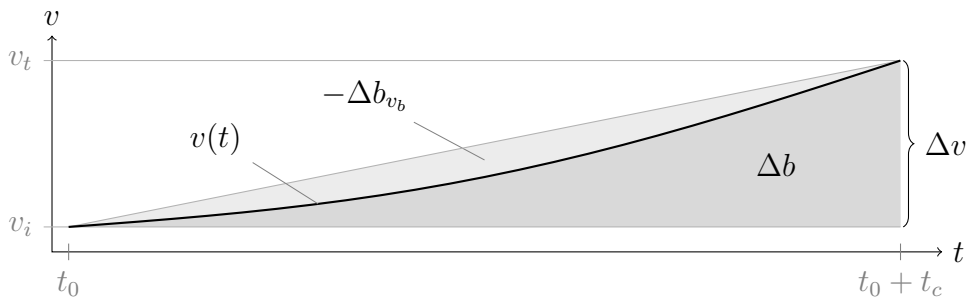

Figure 4.1: Tempo function $v(t)$, showing the total tempo change Δv, total position change Δb, and position change caused by the non-linear part of the tempo function, Δb_{v_b}. Also visible are the initial and target tempi, v_i and v_t respectively.

change without affecting the final tempo. A visualization of the tempo function is presented in figure 4.1. The values of t in the following functions are relative to the beat classification time, and are always larger or equal to zero.

Correcting tempo differences

To make a tempo change over t_c equal to Δv, we define the tempo changing part of our tempo function

$$v_t(t) = \begin{cases} \frac{\Delta v}{t_c}\, t & \text{if } t < t_c \\ \Delta v & \text{otherwise.} \end{cases} \tag{4.9}$$

Since the tempo is changed gradually, it does not immediately match the target tempo. The difference to the target tempo causes a cumulative difference in position, which can be simply solved by integrating over time:

$$\int v_t - \Delta v \,\mathrm{d}t = \Delta v \left(\frac{t^2}{2\, t_c} - t \right) + C. \tag{4.10}$$

The difference in position caused by the difference between $v_t(t)$ and Δv accumulates over the catchup time t_c, causing a gross difference of

$$\Delta b_{v_t} = \int_t^{t+t_c} v_t - \Delta v \,\mathrm{d}t = -\frac{\Delta v\, t_c}{2}. \tag{4.11}$$

This difference in position needs to be taken into account when defining $v_b(t)$, which should compensate for the position difference.

Correcting position differences

To have an effect on position, but zero effect on the final tempo, we define the position changing part of our tempo function

$$v_b(t) = \begin{cases} a \sin\left(\frac{\pi t}{t_c}\right) & \text{if } t < t_c \\ 0 & \text{otherwise,} \end{cases} \tag{4.12}$$

where a is the correction coefficient contributing to the amount of change in position. To solve a, we need to look at the cumulative position difference caused by $v_b(t)$, which can be solved from the integral

$$\int v_b \, \mathrm{d}t = \frac{at}{\pi}\left(1 - \cos\left(\frac{\pi t}{t_c}\right)\right) + C. \tag{4.13}$$

Similarly to Δb_{v_t}, the gross difference in position caused by v_b over t_c

$$\Delta b_{v_b} = \int_t^{t+t_c} v_b \, \mathrm{d}t = \frac{2\,a\,t_c}{\pi}. \tag{4.14}$$

To completely correct the position after t_c, v_b needs to both compensate for Δb and counter the positional change caused by v_t, giving the equation

$$\Delta b_{v_b} = -\left(\Delta b + \Delta b_{v_t}\right). \tag{4.15}$$

Thus, applying equation (4.14), we can solve a as

$$a = -\frac{\pi\left(\Delta b + \Delta b_{v_t}\right)}{2t_c}. \tag{4.16}$$

4.3.3 Offset Compensation

Given the previous beat offset $\Delta b'$ (at time t'), and current v_t and v_b, the current offset can be estimated as

$$\Delta b(t) = \Delta b' - \int_{t'}^{t} v_t + v_b \, \mathrm{d}t. \tag{4.17}$$

To get the best beat classifications, $\mathcal{C}$ is used with the relative beat position

$$b' = \mathcal{M}(t) - p_B - \Delta b(t), \tag{4.18}$$

where t is the time of the beat, and p_B the beginning of the current bar. The offset approaches zero as t approaches $t' + t_c$.

4.3.4 Tempo Changes

Tempo changes in the score are taken into account by changing the tempo function accordingly. When the tempo changes by a factor a, new values for Δv and Δb are calculated, and the tempo function is reset. The new value for the required change in tempo

$$\Delta v = a\left(v_i' + \Delta v'\right) - v'(t), \tag{4.19}$$

where v_i' is the previous initial tempo, $\Delta v'$ the previous tempo change, and $v'(t)$ the current tempo. The new value for the offset

$$\Delta b = \Delta b'(t), \tag{4.20}$$

where $\Delta b'(t)$ is the previous offset function as defined in equation (4.17), is expressed in beats, and is thus not affected by the tempo change.

When changing a tempo function that has not yet finished applying its changes, the most intuitive course of action would be to set

$$t_c = t_c' - t, \tag{4.21}$$

where t_c' is the previous value of t_c, and t the time since the function was last reset. However, since Δv is finite in most cases, we can see that the coefficient $\frac{\Delta v}{t_c}$ in equation (4.9) approaches infinity, as t approaches t_c':

$$\lim_{t \to t_c'} \frac{\Delta v}{t_c' - t} = \pm\infty,\ \Delta v \neq 0. \tag{4.22}$$

Thus it is not feasible to use equation (4.21), but we instead define $t_c = t_c'$.

In some situations tempo changes can be very difficult for the conductor to follow, causing unexpected results when the tempo in the score changes. An example of this could be a gradual decrease in tempo followed by an instant return to the original tempo. If the conductor were to conduct the decelerando without changing conducting tempo, the return to the original tempo would cause a sudden increase in tempo. Since this issue is highly dependent on the score used, a user adjustable parameter is provided, which scales the change factor a in equation (4.19):

$$a' = 1 + \alpha\left(a - 1\right), \tag{4.23}$$

where a' is the new value for a, and $\alpha \in [0, 1]$ the desired fraction of tempo changes to follow.

4.3.5 Relaxed Tempo Following

Since the tempo adjustment function defined in section 4.3.2 is non-linear, selecting the catchup time t_c properly is critical. To have a consistent state for each beat estimation, the adjustment should be fully completed or very near completion at the time the next beat occurs. A very short time, however, would cause very abrupt changes in tempo. Thus t_c is selected as the time between the current and previous beat, which should predict the time to the next beat fairly well.

The tempo adjustment method described above is very strict, as it attempts to do a full tempo and position correction between each beat. In a real life situation the timing of the detected beats often have some fluctuation over time, causing large beat offsets. When such an offset is present, the non-linear part of the tempo function, trying to correct the offset, causes a large temporary change in tempo.

It is not viable to filter the beat offset values, as is done with the instantaneous tempi, for the offset is directly affected by the tempo adjustment function. Doing so would cause feedback in the system, with hard to predict results. Instead, only part of the beat offset is compensated for. This adjustment does not apply to the offset caused by the tempo change (equation (4.10)), but only the offset of the latest beat. To relax the tempo following, only a fraction of the beat offset Δb is corrected, modifying the coefficient for the non-linear part of the tempo function, defined in equation (4.16).

The fraction discussed above, together with the tempo filter cutoff time presented in equation (4.6), mostly define the sensitivity of the tempo following method. These two are made user-adjustable parameters, so they can be adjusted based on both the score and the user's preferences.

4.4 Start Tempo Estimation

The start tempo is estimated based on the duration of the start gesture. As there are several beat patterns for a given time signature, the length of the start gesture may also differ. The duration from bottom to top position is assumed to be half the length of the first *conducted* beat. The starting tempo is thus

$$v_s = \frac{\Delta t_{b_{1,2}}}{2}(t_t - t_b), \tag{4.24}$$

where $\Delta t_{b1,2}$ is the time between the first and second conducted beat, t_t the time at the top of the gesture, and t_b the time at the bottom of the gesture. From the different values of v_s derived from the different conducting patterns, the one closest to the transcribed tempo is selected as the final start tempo.

Chapter 5

Sound Synthesis

Since the objective of the system is to emulate an orchestra, also the sound synthesis system should be able to provide realistic synthesis of an entire orchestra. It should be able to reproduce a musical composition at varying tempi and articulations. The most straightforward approach to supporting these requirements is feeding score events – notes, tempo events, etc. – to a synthesis engine, which produces the final output in real time. The score event format needs to be in (or convertible to) a format that is understood by the synthesis engine. This means the score format is tightly coupled to the synthesis engine, while the coupling to the rest of the system is rather loose. The synthesis engine and score format selected for the system are presented in this chapter.

5.1 Vienna Symphonic Library

There are several commercial orchestral synthesizers available, the majority of which use some form of sampling or concatenative synthesis. Based on the feature set provided and subjective evaluation of sound quality, Vienna Symphonic Library (VSL) [19] was chosen for the task. VSL uses a form of concatenative synthesis, using a vast library of samples played at various velocities and articulations [32].

Each instrument in VSL contains a set of *patches*, which represent a certain articulation and/or playing style. Each patch is capable of synthesizing the whole scale of the given instrument at several velocities. Some patches (such as *staccato* articulations) have a limited note length, while others allow stretching out the note infinitely. Switching between patches can be accomplished with *keyswitch* events.

VSL also contains a *Multi Impulse Response Mixing and Reverberation* engine (MIR). MIR allows simulating an acoustic environment with the possi-

bility of placing virtual instruments and microphones rather freely in a room. MIR includes impulse response data for the Vienna Konzerthaus, and allows placing the virtual microphones at the conductor's podium.

5.1.1 Virtual Studio Technology

Virtual Studio Technology (VST) [33] is a virtual instrument and effect plugin architecture created by Steinberg GmbH. It is one of the plugin formats that VSL is available in. While the plugin API should be considered an implementation detail, it can not be completely ignored in the design phase, because of the vast effect design choices can have on implementation complexity.

5.2 Score Event Format

In order to play a score, the score events need to be provided to the system in some format. The objective was to be able to use existing material as much as possible. The format used in the system combines a MIDI score, discussed in section 5.2.1 and a terse domain specific language (DSL) based score description format, discussed in section 7.6.1.

5.2.1 The Musical Instrument Digital Interface

Regardless of its limitations and age, the Musical Instrument Digital Interface (MIDI) is still one of the most used standards for the digital representation of musical events. MIDI is also the built-in way to communicate musical events in the VST plugin format. Thus using MIDI as the score format is a natural choice.

A MIDI score contains note, time signature and tempo information. It may also contain *program change* events, as specified in the General Midi specification [34], to indicate the instrument to be used for each track. However, practical experience with MIDI files showed, that using program changes to deduce the instrument to use, was not sufficient. Often scores would use the *string ensemble* patch for all strings, even though the string instruments were separated to individual tracks. In cases like this it is necessary to be able to manually define the instrument to be used with each track.

5.3 Patch Switching

In order to control the switching of patches, the synthesis system must have some knowledge of the available instruments and their patches. The parame-

ters used for describing the patches are:

Length The maximum note length allowed by the patch.

Attack The sharpness of the attack portion of the notes.

Weight The *musical weight* of the note, i.e. an accented note would have a large weight.

In addition to the instrument context described above, the patch selection uses a note and conductor context to select the most appropriate patch for each note. The note context includes the length of the note, the current relative tempo, the time to the next note, and the velocity of the note. Furthermore, the conductor context includes all the motion based expressive features, as described in section 3.5.

When a new note is to be played back, the best patch is selected in three stages: First a set of instrument patch parameters is synthesized from the note and conductor contexts. Second, a distance between the synthesized parameters and each instrument patch is calculated. Finally, the closest match is selected, causing the instrument patch to be changed if it differs from the patch currently in use. The patch parameter synthesis and distance function is described in the following sections.

5.3.1 Patch Parameter Synthesis

Synthesizing instrument patch parameters from the note and conductor contexts, consists of applying mapping functions, one for each patch parameter, to the contexts. The length of the note is scaled by the relative tempo, and used as the *length* parameter. Also, the *weight* is a direct mapping from the normalized *jerk peak* in the conductor context.

Attack, however, is synthesized from a combination of the time to the next note, and the *standard deviation of velocity* and *windowed trajectory length* in the conductor context. Experimentation showed that the standard deviation of velocity strongly correlated with the windowed trajectory length, and to achieve more control over the system, this correlation was taken into account: The basis for attack

$$a = \frac{S_N(v)}{(1-\alpha) + \alpha\, l_t}, \tag{5.1}$$

where $S_N(v)$ is the standard deviation of velocity, α the level of correlation desired, and l_t the windowed trajectory length. All variables except a have values within $[0, 1]$, a larger α causing l_t to have greater effect on a.

Additionally, if the time to the next note is below a given threshold, a reduction to the attack is applied. This tries to prevent playing in a staccato

style in fast passages, if the notes are short, but not separated from each other in the score. This relies on the fact that most MIDI scores are written with some separation between notes in staccato passages.

An additional parameter, which does not affect the patch selection, but is related to the conductor context, is *velocity*. The reason this is an exception, is that some patches are *layered*, which means that the synthesis engine uses different samples, based on the velocity of the notes played. The *windowed trajectory length* in the conductor context is used to calculate the final velocity

$$v = \alpha\, v_c + (1 - \alpha)\, v_s, \tag{5.2}$$

where α is the weight given to the conductor context, v_c the velocity in the conductor context, and v_s the velocity in the score. Finally, v is limited to be in the range $[v_{min}, v_{max}]$, where $v_{min} > 0$, in order to neither mute notes nor exceed the maximum velocity.

It is worth noting, that the values for all the parameters in the conductor context, have already been scaled and clamped to be in the range $[0, 1]$. The sensitivity of the system can be adjusted by adjusting all parameters except velocity using the equation

$$p' = \frac{1}{2} + \alpha \left(p - \frac{1}{2} \right), \tag{5.3}$$

where p' is the scaled parameter, $\alpha \in [0, 1]$ the sensitivity, and p the original value of the parameter. The sensitivity of velocity can be adjusted by changing the value of α in equation (5.2) instead. All values used in the implementation were adjusted based on informal testing.

5.3.2 Patch Distance Function

Initially a simple euclidean distance function was used to measure the patch distance. This did not, however, produce satisfactory results, so a more sophisticated method was produced from a mostly empirical basis. The basis of the distance function is applying different weights to the parameters, to emphasize the perceptually more important parameters:

$$d = \sum w_i \left| p_{i_t} - p_i \right|, \tag{5.4}$$

where d is the distance, w_i the weight of the i^{th} parameter, p_{i_t} the target for the i^{th} parameter, and p_i the i^{th} parameter. Ultimately, *weight* and *attack* are given equal weight, while *length* is emphasized. As the length parameter

describes the longest possible note, the distance function for the weight also has to be adjusted to give a larger penalty for too short notes:

$$d_l = \begin{cases} w_l \, |l_t - l| & \text{if } l_t - l > 0 \\ \alpha \, w_l \, |l_t - l| & \text{otherwise,} \end{cases} \tag{5.5}$$

where d_l is the contributing factor of length to the sum in equation (5.4), $w_l > 1$ the weight given to length, l_t the target length, l the length, and $\alpha < 1$ the distance reduction factor for too long notes.

Chapter 6

Visualization

During the development of the conductor follower, a good mechanism for observing the state of the system in real time was lacking, which eventually lead to developing a real time visualization. While the visualization provided its greatest benefits during the development of the system, it can also provide valuable feedback to users of the system. This chapter discusses the functionality and motivation behind each visualization feature separately. All the features are presented in figure 6.1.

6.1 Movement Tracing

When conducting a pattern, it is very difficult for the conductor himself to perceive the actual shape of the hand movement. These days it is common practice to videotape the rehearsals of conducting students [4] for the students to better see how they perform. However, this method does not provide an immediate form of feedback during the rehearsal.

By visualizing both the current position of the hand, and a trace of the movement history, immediate visual feedback can be provided. An intuitive way of presenting the movement history is to draw a colored line segment between each sampled position, with the opacity of the segment decreasing the older the samples are.

6.2 Spatial Beat Visualization

The visualization can show the spatial position of the detected beats, which occur along the traced hand trajectory. Due to the nature of the beat detection algorithm – which does not represent an absolute truth about how beats

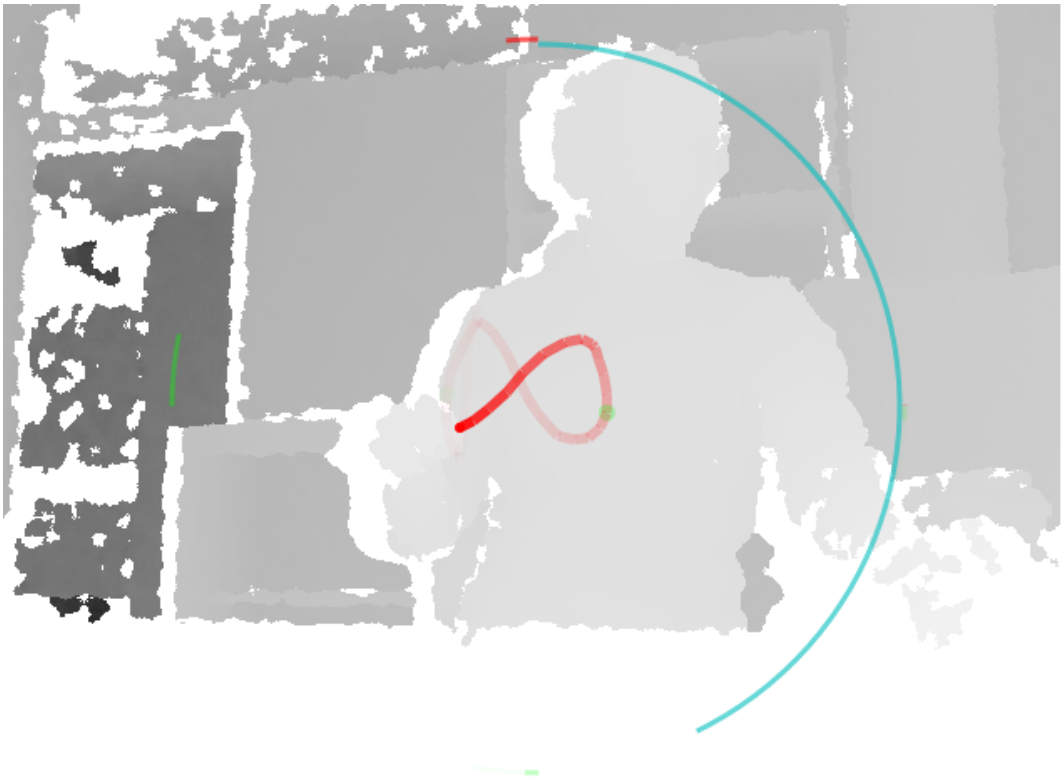

Figure 6.1: Visualization screenshot, showing movement trace (red line), spatial beat position (green dot), current bar phase (turquoise segment), beat offsets (red and greed segments), and depth sensor data (grayscale).

should be detected – the usefulness of this visualization for pedagogical purposes is questionable. However, the visualization was a good aid during the development of the system, and could be a good aid if the system is to be developed further.

6.3 Beat Offset Visualization

To indicate the positions of the detected beats relative to their classifications, a circular visualization is used. The progress of the current bar is visualized with a segment of a circle, alternately growing to a full circle, and shrinking to zero length, as the bars progress to their ends. The detected beats are visualized outside this circle, as a segment from the detected position to the position of the respective classification. Beats that happened early are visualized in green, and beats that happened late in red. Similarly to the movement tracing, the opacity of the old beats decreases and they eventually disappear as time progresses.

6.4 Depth Sensor Output

In addition to visualizations that describe the state of the conductor follower, the raw output of the depth sensor can be displayed as a gryscale video. This is mainly useful for solving possible issues related to the way the depth sensor works: In some situations infra red sources, such as the sun, or certain hand positions can cause problems with the hand tracking algorithm. The depth sensor output can in some situations help in identifying and possibly correcting these problems.

Chapter 7

Implementation Details

This chapter describes details and implementation choices that have been left out from the previous parts of this thesis, as they are not at the core of solving the problems at hand: The architecture is discussed from an implementation centric view, the timing and threading constraints of the implementation are discussed, time handling and motion capture implementation details are described, and the data file formats used by the system are specified. Further details regarding the source code availability and compatibility can be found in appendix A.

7.1 Architectural Overview

Since the synthesis environment is a VST plugin, implementing the conductor follower as a VST plugin as well, was a logical choice. The input for the plugin is read from a MIDI file and the output is MIDI events via the VST interface. These implementation details are, however, hidden behind carefully designed interfaces, separating the core functionality from the input/output functionality. The module division that allows this abstraction, along with the libraries used to implement the features, are described in this section.

7.1.1 Modules

To achieve good abstraction and loose coupling between the different parts of the system, the implementation is divided into six well defined and loosely coupled modules:

1. Common utilities
2. Data file parsers
3. Motion capture

4. Expression to synthesis parameter mapper
5. Score Follower (the core functionality)
6. The VST plugin

The common utilities module contains components that are not specific to conductor following. These include utilities for lock-free programming, mathematical operations and algorithms, time and geometry handling, logging, and debugging.

The data file parser module contains all the parser definitions and supporting data structures for all the input files used by the system. Its main purpose is to hide the implementation of parsing behind a concise API. The format of the data files is discussed in more detail in section 7.6.

The motion capture module provides a simple event-based API for motion related data. It includes the logic for extracting relevant events from motion data. The methods used are described in chapter 3, and the most important implementation details in section 7.5.

The expression to synthesis parameter mapper maps expression parameters to synthesis parameters. The methods used are discussed in section 5.3.

The score follower module provides most of the core functionality, not including motion capture. This includes beat classification, tempo and score following, jitter correction, and instrument patch switching among other things. It also provides abstract interfaces for hiding the implementation details of the score format. The methods used in this module are discussed in chapter 4, and some of the implementation details in section 7.3.

The VST plugin module's main purpose is to separate as much plugin implementation detail from the rest of the system. It implements the VST interface, provides the plugin UI, and the visualization. It also implements the score related interfaces, defined in the score follower module.

In addition to these six modules, the project contains four unit test modules. These modules are not significant regarding the functionality, but merely had a supporting role during system development.

7.1.2 Supporting Libraries

The implementation makes heavy use of several of the Boost C++ libraries [35]. The most noteworthy ones are described in the following list. Libraries included in the C++11 standard [36] are denoted with *, while libraries not yet in the official boost distribution are marked with †.

Chrono* Time library for timestamping and jitter correction.

Lockfree† Various lock-free constructs.

Spirit A parsing library, used for configuration file parsing.
Thread* Threading utilities.
uBLAS Linear algebra, used for polynomial fitting.
Units Compile-time dimensional analysis.

The OpenNI Framework [37] is an open source cross-platform framework for Natural Interaction (NI) devices. PrimeSense Ltd [38] provides a proprietary middleware package called NITE, which works with the OpenNI API, providing higher level functionality such as skeletal and hand tracking. NITE is used via the OpenNI APIs for hand tracking. The motion tracking module contains all the code that uses the OpenNI APIs, so that the rest of the system has no dependencies on OpenNI.

Juce [39] is an open source, multimedia oriented cross-platform C++ library. It was used for its VST plugin wrapper, MIDI file I/O, and user interface (UI) functionality. Juce is only used in the plugin, and the rest of the system has no dependencies on it.

7.2 Real-time Computing and Threading

A real-time audio plugin, which uses motion tracking hardware and has a user interface, imposes certain restraints on how it may be implemented to achieve acceptable performance. These restraints and the patterns used for solving them are discussed in this section.

7.2.1 Real-time Computing

A real-time (RT) system is defined as a system where calculations need to be completed before a given deadline, and can be classified into three categories:

Hard Missing a deadline is a total system failure.
Firm The usefulness of a result is zero after its deadline.
Soft The usefulness of a result degrades after its deadline.

An audio plugin has firm real-time constraints, since the result for each block of audio needs to be delivered in time. Missing a deadline means that the missed block is not played back, causing a glitch in the sound output.

It is important to understand the difference between deadline based real-time constraints and throughput based constraints. Sometimes a calculation is called real-time if the throughput is high enough to complete a large calculation in a given time. For example, if one minute of audio is processed in (or under) one minute, it might be said that the calculation happens in real time. This

does not, however, mean that the same implementation is capable of repeatedly processing smaller blocks of the audio and produce the output for each block given a deadline equal to the block length. Throughput based real-time constraints have more to do with the algorithmic complexity of the methods used, while deadline based constraints require special programming techniques on the implementation level. All further references to RT constraints in this thesis refer to the deadline based definition.

7.2.2 Thread Model

In addition to having RT constraints, the 30Hz frame rate of the motion capture system makes the application a multirate, multithreaded system. Working in such an environment requires using lock-free programming techniques, such as atomic variables and lock-free ringbuffers. The multirate nature also requires having robust timestamping and synchronization mechanisms.

The VST architecture implies using at least two threads: one for audio and/or MIDI processing, and another for the UI. In addition to these two threads, motion capture needs it's own thread, and one additional thread is needed for some lock-free programming techniques. To summarize, the threads are:

1. VST MIDI (audio)
    - Provided by plugin host.
    - Produces the MIDI events.
    - Has firm RT requirements.
    - Low latency required.
2. VST UI
    - Provided by plugin host.
    - Renders the plugin UI.
3. Motion Capture
    - Runs the motion capture device.
    - Has soft RT requirements.
    - Higher latencies allowed compared to the MIDI thread (30 Hz).
4. Butler
    - Runs asynchronous tasks for the RT threads.

7.3 Time Handling

Handling time in a multirate, multithreaded system, is not a trivial task. The motion capture thread is in a blocking state most of the time, waking up at a 30 Hz frequency to process data provided by the motion capture system. The plugin, however, needs to provide MIDI data based on the host's audio settings, a typical configuration running the plugin at around 100 Hz (an audio block length of 10ms). Additionally, the thread in which the plugin runs is controlled by the host, and the code has to be completely lock-free.

Considering the implications of needing to run lock-free code on a modern multi-core system, running a general purpose operating system (OS), no assumptions about causality or parallelism between motion capture and plugin code execution can be made – scheduling latencies and parallel execution forces the use of a reliable timestamping mechanism for synchronization between the audio and motion capture threads. The Boost Chrono library provides a steady clock, which measures time since the latest system boot-up. Acquiring the current timestamp was verified to be lock-free at least on Windows and OS X as of Boost version 1.49.0. These timestamps are used for measuring the interval between events happening in the motion capture and plugin threads. Timestamping events in the motion capture thread is straightforward, each event simply being timestamped with the current timestamp returned by the system. The plugin thread, however, requires jitter correction to prevent timing problems.

7.3.1 Jitter Correction

The plugin thread needs to know not only the timestamp corresponding to the beginning of the current audio block, but also an estimate for the end of the block. Let the audio block be defined by its beginning and end:

$$\boldsymbol{\tau} := [\tau_b, \tau_e[.$$

Using the information provided to the plugin, the theoretical block length

$$\Delta\tau' = \frac{n_b}{f_s}, \tag{7.1}$$

where n_b is the block size in samples and f_s the samplerate. This length is then used for estimating the end of each block

$$\tau_e = t_c + \Delta\tau', \tag{7.2}$$

where t_c is the current timestamp at the beginning of the block. Due to scheduling latencies, the value of t_c might differ from the previous block end

estimate $\tau_{e_{\text{prev}}}$. In order to prevent gaps and overlaps between the time estimates of consecutive audio blocks, the beginning of each block is taken from the end of the previous block

$$\tau_b = \tau_{e_{\text{prev}}}. \tag{7.3}$$

The VST specification requires each MIDI event to have its position defined as a sample offset from the beginning of the current block. As the actual block length

$$\Delta\tau = \tau_e - \tau_b \tag{7.4}$$

differs from the theoretical block length $\Delta\tau'$ most of the time, this difference needs to be taken into consideration when calculating the event sample offset

$$\Delta n_e = \frac{\Delta\tau'}{\Delta\tau}(t_e - \tau_b)f_s, \tag{7.5}$$

where t_e is the event time. Using equation (7.1), this can be simplified to

$$\Delta n_e = n_b \frac{t_e - \tau_b}{\Delta\tau}. \tag{7.6}$$

7.3.2 Dimensional Analysis

Because of the various time bases and units used (see section 4.1), using compile time dimensional analysis using Boost.Units [40] eased the development process. Boost.Units uses zero runtime overhead C++ template metaprogramming techniques [41] to check during compile time, that all calculations have the proper dimensions. This does not guarantee dimensionally proper calculations all the time (e.g. dimensionless units may cause errors), but does catch many errors made during development. For handling time, a custom unit system was declared. This system contains base dimensions for beats, bars, samples and physical time, and derived dimensions for bar durations, tempi, sample rates, and speed changes. The unit system is described in detail in Table 7.1. The difference between physical time and score time (both in seconds), comes inherently from the libraries used: Boost.Units for score time and Boost.Chrono for physical time. This implies that an explicit conversion function is always needed to convert between these two dimensions.

7.4 Essential Common Utilities

A number of utilities are widely used across the application, and reserve a mention. These are mostly related to lock-free programming techniques.

Dimension	Unit	Notes
beats	beat	Base for musical time
bars	bar	See bar duration.
samples	sample	See sample rate.
time	second	Base for physical time.
sample rate	samples per second	Derived dimension.
bar duration	beats per bar	Derived dimension.
tempo	beats per second	Derived dimension.
tempo change	beats per second squared	Derived dimension.

Table 7.1: Overview of custom unit system for time.

7.4.1 Event Buffer

The event buffer is probably the most used utility in the implementation. It is a container for storing timestamp-event-pairs. It allows specifying the internal storage depending on the use case. For data that is collected when loading the score, a dynamically resizable container (such as std::vector) can be used, while buffers used during playback mostly use a statically sized ringbuffer to guarantee memory allocation free operation. Utilities for accessing timestamp-based ranges, with all valid combinations of bounded, left-bounded, left-open and left-closed are available.

7.4.2 The Butler Thread

The butler thread is used for asynchronously running tasks that can not be run in an RT thread. The butler thread is configured with a maximum execution interval, and checks for new tasks and executes them repeatedly, until terminated. The thread blocks as necessary to not exceed the maximum execution interval.

7.4.3 Lock-Free Ringbuffer

The lock-free ringbuffer provided by the Boost.Lockfree library [42] is used for most of the lock-free inter-thread communication. This includes passing events from the motion capture thread to the MIDI thread, and logging from RT threads using the Butler thread. It is an essential utility for lock-free multithreaded programming.

7.4.4 Chen & Burns Buffer

While the lock-free ringbuffer is a good solution for event-like data, better mechanisms exist for sharing a single instance of data, e.g. a state. For this use, a fully asynchronous reader-writer mechanism as described by Chen and Burns [43] was implemented. The mechanism allows lock-free access for a single writer and a fixed number of readers. It uses a number of copies of the data, and a set of atomic variables to track the reading and writing states.

7.5 Motion Capture

Since most of the hard work related to motion capture is done by NITE – the implementation of which is proprietary – there is not much to discuss in this section. However, a quick overview on the implementation of the polynomial regression filter is provided.

7.5.1 The Polynomial Regression Filter

The polynomial regression filter is used for three purposes in the system: smoothing, differentiation and interpolation. The polynomial fitting part for the filter is implemented using matrix operations and the uBLAS library. Estimating the polynomial coefficients is done by the equation

$$\widehat{\mathbf{a}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}, \tag{7.7}$$

where $\widehat{\mathbf{a}}$ are the estimated coefficients, $\mathbf{y}$ the observations, and $\mathbf{X}$ the design matrix

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ 1 & x_3 & x_3^2 & \dots & x_3^m \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix}, \tag{7.8}$$

where $x_1 \dots x_n$ are the observation times, and m the order of the resulting polynomial. It is required that $n > m$. Smoothing and differentiation is implemented by selecting an odd value for n and evaluating the polynomial at the center value $x_{\lfloor \frac{n}{2} \rfloor}$. An optimization that is easy to implement for equally sampled data, is to pre-calculate $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T$, with $x_{\lfloor \frac{n}{2} \rfloor} = 0$. The value and derivatives at the center point can thus be calculated from the polynomial

$$\widehat{y}(x) = \widehat{\mathbf{a}}_m\, x^m + \widehat{\mathbf{a}}_{m-1}\, x^{m-1} + \ldots + \widehat{\mathbf{a}}_2\, x + \widehat{\mathbf{a}}_1. \tag{7.9}$$

The simplified equations being

$$\widehat{y}(0) = \widehat{\mathbf{a}}_1 \tag{7.10}$$

for the value, and

$$\widehat{y}^{(n)}(0) = n!\,\widehat{\mathbf{a}}_{n+1} \tag{7.11}$$

for the n^{th}derivative.

To make $\mathbf{X}^T\mathbf{X}$ invertible, it is also necessary to reverse the values $x_1 \dots x_n$ used in equation (7.8). This change requires a correction to equation (7.7): the observation values $\mathbf{y}$ need to be reversed also.

7.6 Data File Formats

For additional data needed by the system, a number of JSON-like [44] domain specific languages (DSLs) were developed. The syntax in all of them is similar, only the contained data varying between different formats. The formats are described here briefly.

7.6.1 Score Description Files

Because the data present in a MIDI score was not sufficient alone, a format for describing additional data related to the score was developed. A score definition file includes the following information:

Name	A human readable name for the score.
MIDI file	The path of the related midi file.
Instrument file	The path of the related instrument definition file.
Beat pattern file	The path of the related beat pattern file.
Track information	A list of track to instrument mappings.
Score events	A list of score events.

An example of a score description file can be found from Appendix B. For more information on instrument definitions, see section 7.6.2, and for beat patterns section 7.6.3. While the format supports having different types of score events, only one type of score event was implemented. This event type is described in detail below.

Tempo Sensitivity Events

A tempo sensitivity event contains the following information:

Position	The score position to which this sensitivity change applies.

Sensitivity A sensitivity value in the range $0 \ldots 1$.

The event causes the catchup time t_c, as described in section 4.3.2, to be adjusted to provide a different level of sensitivity in tempo following. A low sensitivity leads to a long t_c, whereas a high sensitivity translates to a short t_c.

7.6.2 Instrument Definition Files

The purpose of instrument definition files is to provide the necessary data for mapping expressive parameters to different instrument patches. It also defines the MIDI channel configuration used in the synthesis environment. The format allows several instruments of the same type to be used at the same time, whose count is limited to the number of MIDI channels reserved to it. The file is a list of instrument definitions, which include patch definitions. A patch definition includes the following information:

Name	A human readable name for the patch.
Keyswitch	The keyswitch for selecting this patch.
Length	Length of patch, as described in section 5.3. This is a relative value in the range $]0, 1]$.
Attack	Attack of patch, as described in section 5.3.
Weight	Weight of patch, as described in section 5.3.

Each patch is unique to an instrument, each instrument also including the following information:

Name	A human readable name for the instrument.
Shortest note	The length for the relative note length of 0.
Longest note	The length for the relative note length of 1.
Channels	A list of MIDI channels to use for instances of this instrument.
Patches	List of patches.

An example of an instrument definition file can be found from Appendix B.

7.6.3 Beat Pattern Definition Files

The beat pattern definition files include the information for all the conducting beat patterns that the system recognizes. Each beat pattern includes the following information:

Meter	The time signature to which this pattern applies.

Beats	A list of beat times as an offset into the bar in quarter notes.

An example of a beat pattern definition file can be found from Appendix B.

Chapter 8

Discussion and Conclusions

While evaluating systems similar to the conductor follower has been discussed in e.g. [45], no formal evaluation was performed for the system implemented. The content in this chapter is mostly based on testing during development, and a few informal testing sessions with a professional conductor.

8.1 Evaluation of Results

The conductor follower system implemented tracks the motions of the user, analyses them, and plays back a modified score via a synthesis engine. The system is evaluated from three different perspectives:

1. The overall architecture,
2. the implemented feature set, and
3. the quality of the implemented features.

8.1.1 Architecture

The architecture was designed largely based on the real-time computing and threading restrictions discussed in section 7.2. Also the selected synthesis environment (chapter 5) and supporting libraries (section 7.1.2) had some influence on the architecture. These implementation specific design motives lead to designing an architecture with a rather large amount of clearly defined, minimal interfaces. While having loose coupling between implementation modules is usually good, inserting an abstraction layer in the wrong place can be detrimental to the clarity of the related interface.

While the data file interfaces and the main score follower interface (depicted in figure 2.1) worked out very well, the motion tracker interface was more implementation detail driven, and thus not quite as elegant. This is due to the

fact that the motion data processing is done in the motion tracking thread, in order to keep the MIDI thread load as low as possible. This design choice can be justified by taking into account the timing constraints for each thread. The coupling between the motion tracker and score follower is tighter than it should, because it is only feasible to extract the motion features needed by the score follower from the motion data. This means that the score follower needs to be aware of exactly which events the motion tracker is producing and the motion tracker aware of what events the score follower expects to receive. Only passing the raw motion data and the polynomial regression filter results (speed, acceleration, and jerk, see section 3.2.1) would loosen the coupling between the modules.

8.1.2 Implemented Features

Since the communication modeled in a conductor follower system happens between people – the conductor and the musicians – having a "complete" solution to it would essentially require passing the *Turing test* [46] in a conducting situation. Examined from this perspective, the most significant feature lacking from the system is a communication channel outside of conducting gestures, as discussed in section 2.2. A practical viewpoint will, however, reveal that this feature is only complementary to other features, as it can only be used to adjust features that are already present in the system. Implementing a system that could actually learn new features based on conductor feedback is unfeasible with current technology.

Sousa [3] lists eight different conducting emblem categories: beat patterns, dynamics, styles, preparations, releases, fermatas/holds, tempo changes, and phrasings. Out of these, beat patterns, dynamics, styles, tempo changes, and phrasings are supported at some level, and are discussed in more detail in the next section. Looking at the remaining features, the biggest shortcoming of the system is the lack of support for holds and grand pauses (fermatas). Missing a fermata will cause the system to drift off badly, and is a much more severe problem than missing almost any other expressive feature. Also, depending on the style and timing of a release gesture, it might disrupt tempo following. Supporting preparations, on the other hand, would only serve a pedagogical or entertainment value, as the system does not miss notes unless such behavior is explicitly implemented.

8.1.3 Feature Quality

Utilizing the category division by Sousa [3], the following conducting emblem categories are supported at some level: beat patterns, dynamics, styles, tempo

changes, and phrasings. Dynamics, styles, and phrasings are not directly interpreted, but are indirectly supported by the expressive feature extraction (section 3.5) and synthesis patch switching mechanisms (section 5.3). The mechanism is somewhat limited, but gives some level of freedom to modifying the expressive features of the score.

Beat patterns are supported on the timing level, i.e. the system understands that the beat timing can vary between different conducting patterns in the same time signatures. It does not, however, try to use the directional information of the pattern to make the following more robust. This is partly a design choice, as conductors often do not follow the "textbook" patterns very precisely, but instead use more freedom in their movements to convey expressive features, as can be seen from recorded conductor movements [11]. Thus, the directional information can not be relied on too much in beat classification, but should rather be used as a complementary data source.

Tempo change support is the most tested feature in the system. It was evaluated by a professional conductor, and based on this very informal evaluation, it was concluded to perform very well with moderate tempo changes. Abrupt changes did, however, sometimes cause classification problems, causing the follower to lose synchronization with the conductor. Also, if the score file contained tempo changes, which the conductor did not follow, it made the situation even worse, as the expectations of the system differed largely from the conducted tempo.

8.2 Future Work

As discussed in section 8.1.2, creating the "perfect" conductor follower is something that is not going to be achieved in the foreseeable future. This means that there will always be something to improve in the system. However, some features and improvements will be more beneficial than others, and this section attempts to address the most relevant ones. Potential modifications based on alternative use cases are also considered.

8.2.1 Machine Learning

Machine learning was not used in this system, mostly because it would have implied having to train the system for each user separately, or having to collect a large amount of training data. However, if this approach would be taken, it would definitely be possible to improve many of the features implemented. Due to the modular approach taken in designing the overall architecture of

the system, incorporating machine learning methods into the system should be feasible without introducing too much technical debt.

Based on previous work, one of the most likely parts of the system to use machine learning in, would be gesture detection [20]. Also, expressive score manipulation is a subject where machine learning has been used earlier [13].

8.2.2 Motion Tracking

Improvements in the motion tracking methods could both improve existing features, and introduce new features to the system. As all of the continuous user input to the system is provided via the motion tracking interface, limited capabilities in motion tracking cause a bottleneck to the capabilities of the whole system.

From existing features, the beat tracking method is one of the most simplified ones with regards to motion tracking. This is due to the fact that it only uses the times of detected beats, and dismisses all other information related to the beat pattern trajectory. However, using the directional information for beat classification is not a very simple problem, as discussed in section 8.1.3. While other methods could most probably provide improvements to the system, using machine learning and conductor specific training is probably the most feasible way to improve beat tracking.

As for supporting very specific gestures, such as holds or preparations, there is almost no support present in the system. While implementing such features requires having methods for detecting the gestures from the motion input, they also require work in other parts of the system. As the system does not include anything closely related to the problem at hand, future implementations of such features are free to use any applicable methods available.

8.2.3 Score Analysis

Several methods for applying expressive features to musical scores and performances have been developed over the years [13]. They all analyze the score, and produce *expressive* modifications to it, based on the results of the analysis. Using such analysis in tandem with the input from the conductor, could possibly lead to much improved and more human like expressiveness.

The current system largely uses expressive features built into the MIDI score, while the aforementioned systems analyze "raw" scores, with no dynamic or timing variations built in. While the expressive feature in the MIDI score work well when the tempo is constant – the quality of the MIDI file setting the baseline for the quality – changes in tempo can produce unsatisfactory results. The problem is mostly caused by the non-linear part of the tempo

function (see section 4.3.2), which does not present a musically satisfactory result [12, 13]. Also, the absolute tempo of the piece has an effect on expressive timing [14], which is not taken into account in the current system. Using a more "intelligent" system to make tempo adjustments, would definitely be an improvement.

8.2.4 Potential Modifications

In its current form, the use cases for the system are somewhat limited, due to the purely academic nature of the implementation. However, there are other potential use cases for the system, which would require some modifications or extensions to the system. Some of them are discussed in this section. Others, such as use in conducting classes, would require more research into what features would actually be useful.

Conduct your own recordings One use case for the system is being able to record the result of your conducting, for the purpose of listening to it later on. Given a well made score and high quality synthesis environment, the recordings can reach a high quality. Listening to the resulting recordings could be beneficial for conductors or composers evaluating different interpretations, or for plain entertainment purposes. Extending the idea to the internet, these recordings could also be shared among friends, or one could even hold virtual conducting competitions where the recordings are shared and judged online. Since currently the audio is output from a VST plugin, recording it with a third party software is possible. However, it would be possible to make this task a lot easier by building the functionality into the system. Especially if the recording is to be shared, having one application for the whole task would be a crucial improvement to the user experience.

"Conductor Hero" Given the popularity of games like *Guitar Hero*, where the player uses an instrument like controller to simulate playing along with a band, modifying the system to behave like a similar game for conducting is a use case easily come up with. However, the setting in such games is quite the opposite of the current functionality of the conductor follower system – where in the game, the user has to keep up with the system, in the conductor follower, the system has to keep up with the user. Nevertheless, there are several components in the system, which could be reused in a game; the required development effort would be very large though.

8.3 Conclusions

The main purpose of the work was to implement a conductor follower system using consumer grade motion tracking equipment and a high quality sample based synthesis environment. Also among the goals was keeping the mode of operation on a generalized level, so that the system works without the need for training. Even though there is room for improvement – and always will be for systems of such complexity – the end result met the main goals.

While the methods applied do not contain anything especially novel, the system does have a well documented and extensible architecture, and should be usable as an implementation basis for future projects. Also, compared to previous similar projects, lots of effort was put into thoroughly documenting the underlying methods – especially ones related to tempo following. All in all, the system is able to model an essential subset of conductor-musician communication, and produce a satisfying interactive experience resembling conducting a real orchestra.

Bibliography

[1] Elizabeth A. Green. *The Modern Conductor*. Englewood Cliffs, NJ: Prentice Hall, 1981.

[2] Ronald Wayne Gallops. "The effect of conducting gesture on expressive-interpretive performance of college music majors". PhD thesis. University of South Florida, 2005.

[3] Gary Donn Sousa. "Musical conducting emblems: an investigation of the use of specific conducting gestures by instrumental conductors and their interpretation by instrumental performers". PhD thesis. The Ohio State University, 1988.

[4] Anu Konttinen. "Conducting gestures: institutional and educational construction of conductorship in Finland, 1973–1993". PhD thesis. Helsinki University, 2008.

[5] Matthew Turk. "Gesture recognition". In: *Handbook of Virtual Environments: Design, Implementation, and Applications*. Mahwah, NJ: Lawrence Erlbaum Associates, 2002.

[6] Zhengyou Zhang. "Microsoft Kinect sensor and its effect". In: *Multimedia, IEEE* 19.2 (Feb. 2012), pp. 4–10. ISSN: 1070-986X. DOI: 10.1109/MMUL.2012.24.

[7] Thomas B. Moeslund, Adrian Hilton, and Volker Krüger. "A survey of advances in vision-based human motion capture and analysis". In: *Computer Vision and Image Understanding* 104.2 (Nov. 2006), pp. 90–126. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2006.08.002.

[8] Daniel Weinland, Remi Ronfard, and Edmond Boyer. "A survey of vision-based methods for action representation, segmentation and recognition". In: *Computer Vision and Image Understanding* 115.2 (2011), pp. 224–241. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2010.10.002.

[9] G. Luck and J. Sloboda. "Exploring the spatio-temporal properties of simple conducting gestures using a synchronization task". In: *Music Perception* 25.3 (2008), pp. 225–239.

[10] G. Luck and J.A. Sloboda. "Spatio-temporal cues for visually mediated synchronization". In: *Music Perception* 26.5 (2009), pp. 465–473.

[11] Geoff Luck and Petri Toiviainen. "Ensemble musicians' synchronization with conductors' gestures: an automated feature-extraction analysis". In: *Music Perception* 24.2 (2006), pp. 189–200.

[12] P. Desain and H. Honing. "Tempo curves considered harmful". In: *Contemporary Music Review* 7.2 (1993), pp. 123–138.

[13] Alexis Kirke and Eduardo R. Miranda. "An overview of computer systems for expressive music performance". In: *Guide to Computing for Expressive Music Performance*. Ed. by Alexis Kirke and Eduardo R. Miranda. Springer London, 2013, pp. 1–47. ISBN: 978-1-4471-4122-8. DOI: 10.1007/978-1-4471-4123-5_1.

[14] Peter Desain and Henkjan Honing. "Does expressive timing in music performance scale proportionally with tempo?" In: *Psychological Research* 56.4 (1994), pp. 285–292. ISSN: 0340-0727. DOI: 10.1007/BF00419658.

[15] Patrik N. Juslin. "Cue utilization in communication of emotion in music performance: relating performance to perception." In: *Journal of Experimental Psychology: Human Perception and Performance* 26.6 (2000), p. 1797.

[16] O. Moravec and J. Stepanek. "Verbal descriptions of musical sound timbre and musician's opinion of their usage". In: *Fortschritte der Akustik* 31.1 (2005), p. 231.

[17] Anders Friberg. "A fuzzy analyzer of emotional expression in music performance and body motion". In: *Proceedings of Music and Music Science*. 2004, pp. 28–30.

[18] Kia C. Ng. "Music via motion: transdomain mapping of motion and sound for interactive performances". In: *Proceedings of the IEEE*. Vol. 92. 4. Apr. 2004, pp. 645–655. DOI: 10.1109/JPROC.2004.825885.

[19] Vienna Symphonic Library GmbH. *Vienna Symphonic Library*. URL: http://www.vsl.co.at (accessed on 2012-11-23).

[20] Marco Fabiani, Anders Friberg, and Roberto Bresin. "Systems for interactive control of computer generated music performance". In: *Guide to Computing for Expressive Music Performance*. Ed. by Alexis Kirke and Eduardo R. Miranda. Springer London, 2013, pp. 49–73. ISBN: 978-1-4471-4122-8. DOI: 10.1007/978-1-4471-4123-5_2.

[21] W. Buxton, W. Reeves, G. Fedorkow, et al. "A microcomputer-based conducting system". In: *Computer Music Journal* (1980), pp. 8–21.

[22] F. Haflich and M. Burnds. "Following a conductor: the engineering of an input device". In: *Proceedings of the 1983 International Computer Music Conference*. San Francisco, 1983.

[23] Tommi Ilmonen. "Tracking conductor of an orchestra using artificial neural networks". MS thesis. Helsinki University of Techonology, 1999.

[24] H. Morita, S. Hashimoto, and S. Ohteru. "A computer music system that follows a human conductor". In: *Computer* 24.7 (1991), pp. 44–53.

[25] S. Usa and Y. Mochida. "A multi-modal conducting simulator". In: *Proceedings of the International Computer Music Conference*. 1998, pp. 25–32.

[26] P. Kolesnik and M. Wanderley. "Recognition, analysis and performance with expressive conducting gestures". In: *Proceedings of the International Computer Music Conference (ICMC), Miami, USA*. 2004.

[27] E. Lee, I. Grüll, H. Kiel, et al. "conga: a framework for adaptive conducting gesture analysis". In: *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME06)*. 2006, pp. 260–265.

[28] B. Bruegge, C. Teschner, P. Lachenmaier, et al. "Pinocchio: conducting a virtual symphony orchestra". In: *Proceedings of the International Conference on Advances in Computer Entertainment Technology*. ACM. 2007, pp. 294–295.

[29] Jan Borchers, Eric Lee, Wolfgang Samminger, et al. "Personal orchestra: a real-time audio/video system for interactive conducting". In: *Multimedia Systems* 9.5 (2004), pp. 458–465. ISSN: 0942-4962. DOI: 10.1007/s00530-003-0119-y.

[30] Geoff Luck and Petri Toiviainen. "Perception of expression in conductors' gestures: a continuous response study". In: *Music Perception* 28.1 (2010), pp. 47–57.

[31] Guerino Mazzola. "Tempo curves". In: *Musical Performance*. Ed. by Guerino Mazzola. Computational Music Science. Springer Berlin Heidelberg, 2011, pp. 47–56. ISBN: 978-3-642-11838-8.

[32] Diemo Schwarz. "Concatenative sound synthesis: the early years". In: *Journal of New Music Research* 35.1 (2006), pp. 3–22. DOI: 10.1080/09298210600696857.

[33] Steinberg GmgH. *Steinberg Releases VST 2.4 Standard With New Features*. URL: http://www.steinberg.net/index.php?id=334&L=1 (accessed on 2012-11-29).

[34] MIDI Manufacturers Association Incorporated. *General MIDI 2 Specification*. 2003.

[35] *Boost C++ Libraries*. URL: http://boost.org (accessed on 2012-11-23).

[36] ISO. *ISO/IEC 14882:2011 Information Technology — Programming Languages — C++*. Geneva, Switzerland: International Organization for Standardization, Feb. 28, 2012, p. 1338.

[37] *OpenNI*. URL: http://www.openni.org (accessed on 2012-11-23).

[38] Primesense Ltd. *PrimeSense Natural Interaction*. URL: http://www.primesense.com (accessed on 2012-11-23).

[39] Raw Material Software Ltd. *JUCE (Jules' Utility Class Extensions)*. URL: http://www.rawmaterialsoftware.com/juce.php (accessed on 2012-11-23).

[40] Matthias Christian Schabel and Steven Watanabe. *Boost.Units: Zero-overhead Dimensional Analysis and Unit/Quantity Manipulation and Conversion*. URL: http://www.boost.org/doc/libs/release/doc/html/boost_units.html (accessed on 2012-11-23).

[41] D. Abrahams and A. Gurtovoy. *C++ Template Metaprogramming: Concepts, Tools, and Techniques From Boost and Beyond*. Addison-Wesley Professional, 2004.

[42] Tim Blechmann. *Boost.Lockfree: Lock-free C++ Data Structures*. URL: http://boost-sandbox.sourceforge.net/doc/html/lockfree.html (accessed on 2012-11-23).

[43] Jing Chen and Alan Burns. *A Fully Asynchronous Reader/Writer Mechanism for Multiprocessor Real-time Systems*. Tech. rep. 1997.

[44] Douglas Crockford. *RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON)*. Tech. rep. URL: http://tools.ietf.org/html/rfc4627.

[45] Roberto Bresin and Anders Friberg. "Evaluation of computer systems for expressive music performance". In: *Guide to Computing for Expressive Music Performance*. Ed. by Alexis Kirke and Eduardo R. Miranda. Springer London, 2013, pp. 181–203. ISBN: 978-1-4471-4122-8. DOI: 10.1007/978-1-4471-4123-5_7.

[46] A. M. Turing. "Computing machinery and intelligence". In: *Mind* 59 (1950), pp. 433–460.

Appendix A

Source Code Availability and Compatibility

At the time of publishing, the source code for the implementation is freely available on *GitHub*, at `https://github.com/sbergen/ConductorFollower`. The project was developed using *Microsoft Visual Studio 2010* (VS2010) on *Windows 7*. The implementation uses some of the C++11 features available in VS2010, but does not by default use any Windows specific features. Thus it should be rather trivial to port the code to any other platform with:

- A compiler that supports the used C++11 features,
- support for OpenNI,
- support for Juce, and
- support for Boost.

Porting will, however, require creating a build script for the given environment, as only VS2010 project files are included. Feasible platforms include at least *OS X* and *Linux*. Pull requests for compatibility fixes for different platforms are highly welcome.

Sample instrument patches for VSL and their respective instrument definition files, along with a beat pattern definition file can be found from the `data_content` directory in the repository. The instrument patches require the *Vienna Special Edition* volumes 1, 1+, 2, and 2+.

Appendix B

DSL Examples

B.1 Score Description Format

```
score {
  name: "Mozart: Trio for piano, clarinet and viola in E flat
     major, KV 498 (1786)",
  midi_file: "/scores/mozart_498.mid",
  instrument_file: "/scores/instruments.def",
  beat_pattern_file: "/scores/beat_patterns.def",

  tracks: [
    track {
      name: "clarinet",
      instrument: "clarinet Bb"
    },
    track {
      name: "viola",
      instrument: "chamber viola"
    },
    track {
      instrument: "ignore"
    },
    track {
      name: "piano right hand",
      instrument: "piano"
    },
    track {
      name: "piano left hand",
      instrument: "piano"
    }
  ],

  events: [
```

```
    tempo_sensitivity {
      position: 0|0,
      sensitivity: 1.0
    },
    tempo_sensitivity {
      position: 1|0,
      sensitivity: 0.8
    },
    tempo_sensitivity {
      position: 3|0,
      sensitivity: 0.5
    },
  ]
}
```

B.2 Instrument Definition Format

```
[
  // Special "keyword" for ignoring a track
  instrument { name: "ignore" },

  instrument {
    name: "clarinet Bb",
    shortest_note_threshold: 0.05,
    longest_note_threshold: 1.0,

    channels: [ 1, 2 ],

    patches: [
      patch {
        name: "perf-rep staccato"
        keyswitch: C1
        length: 0.05,
        attack: 0.7,
        weight: 0.5
      },
      patch {
        name: "staccato"
        keyswitch: C#1
        length: 0.1,
        attack: 0.7,
        weight: 0.5
      },
      patch {
        name: "portato"
        keyswitch: D1
        length: 0.2,
        attack: 0.4,
```

```
      weight: 0.5
    },
    patch {
      name: "perf-rep legato"
      keyswitch: D#1
      length: 0.5,
      attack: 0.2,
      weight: 0.5
    },
    patch {
      name: "sfortzato"
      keyswitch: E1
      length: 0.8,
      attack: 0.9,
      weight: 1.0
    },
    patch {
      name: "fortepiano"
      keyswitch: F1
      length: 0.8,
      attack: 0.4,
      weight: 1.0
    },
    patch {
      name: "sustain"
      keyswitch: F#1
      length: 1.0,
      attack: 0.3,
      weight: 0.4
    },
    patch {
      name: "legato"
      keyswitch: G1
      length: 1.0,
      attack: 0.5,
      weight: 0.5
    },
    patch {
      name: "legato-sus"
      keyswitch: G#1
      length: 1.0,
      attack: 0.4,
      weight: 0.6
    }
  ]
},

instrument {
  name: "piano",
```

```
    channels: [ 3, 4 ],

    patches: [
      patch {
        name: "regular"
        keyswitch: C0
        length: 0.5,
        attack: 0.5,
        weight: 0.5
      }
    ]
  }
]
```

B.3 Beat Pattern Definition Format

```
[
  // 2/4

  beat_pattern {
    meter: 2/4,
    beats: [
      beat { time: 0.0 },
      beat { time: 1.0, },
    ]
  },

  beat_pattern {
    meter: 2/4,
    beats: [
      beat { time: 0.0 },
    ]
  },

  // 3/4

  beat_pattern {
    meter: 3/4,
    beats: [
      beat { time: 0.0 },
      beat { time: 1.0, },
      beat { time: 2.0, },
    ]
  },

  beat_pattern {
    meter: 3/4,
```

```
    beats: [
      beat { time: 0.0 },
    ]
  },

  // 4/4

  beat_pattern {
    meter: 4/4,
    beats: [
      beat { time: 0.0 },
      beat { time: 1.0 },
      beat { time: 2.0 },
      beat { time: 3.0 },
    ]
  },

  beat_pattern {
    meter: 4/4,
    beats: [
      beat { time: 0.0 },
      beat { time: 2.0, },
    ]
  },

  beat_pattern {
    meter: 4/4,
    beats: [
      beat { time: 0.0 },
    ]
  },

  // 6/8

  beat_pattern {
    meter: 6/8,
    beats: [
      beat { time: 0.0 },
      beat { time: 0.5 },
      beat { time: 1.0 },
      beat { time: 1.5 },
      beat { time: 2.0 },
      beat { time: 2.5 },
    ]
  },

  beat_pattern {
    meter: 6/8,
    beats: [
```

```
      beat { time: 0.0 },
      beat { time: 1.5 },
    ]
  },

  beat_pattern {
    meter: 6/8,
    beats: [
      beat { time: 0.0 },
    ]
  },
]
```