Ludwig-Maximilians-Universität München
Department “Institut für Informatik”
Lehr- und Forschungseinheit Medieninformatik
Prof. Dr. Heinrich Hußmann

Diplomarbeit

Acceleration-Based Gesture Recognition for Conducting with Hidden Markov Models

Dominik Schmidt
schmidte@cip.ifi.lmu.de

Bearbeitungszeitraum:	21.03.2007 bis 22.10.2007
Betreuer:	Prof. Asim Smailagic, Ph. D.
	Prof. Daniel P. Siewiorek, Ph. D.
Verantw. Hochschullehrer:	Prof. Bernd Brügge, Ph. D.
Co-Referat:	Prof. Dr. Heinrich Hußmann

Abstract

This thesis focuses on developing new means of input for an existent system allowing users with any skill to conduct a virtual orchestra in realtime. As a first step, several user studies with musical conductors were performed to learn more about the interaction between conductor and orchestra.

The eWatch, a wearable computing and sensing platform, is selected as input device. It is worn like a regular wrist watch and transmits acceleration data via Bluetooth to the host computer.

Hidden Markov models are chosen to perform the gesture recognition task. While realizing a reliable and continuous gesture recognition, the proposed system is able to successfully reject non-meaningful hand movements, applying a threshold model. The system makes use of a discrete hidden Markov model in conjunction with a modified version of the Viterbi algorithm, which is specifically adopted for continuous gesture recognition.

By performing common conducting gestures, the user is able to indicate the measure. In addition, he or she can influence the orchestra's tempo and volume by varying the gestures in speed and size. The evaluation of the proposed approach with different users yields a detection ratio of more than 94% and a reliability of more than 90% with an average latency around 90 ms for the user-dependent recognition.

While focusing on the application domain of conducting, the proposed recognition approach is versatility applicable to arbitrary user interfaces benefiting from gesture input. Using a machine learning approach, new gestures can easily be trained.

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, October 22, 2007

. . . .

Preface

This thesis has been submitted as *Diplomarbeit* (diploma thesis) to the department of computer science, school of media informatics, of the Ludwig-Maximilians-Universität in München, Germany. While it was supervised by Prof. Bernd Brügge, Ph. D. of the Technische Universität München, I did the underlying research at the Institute for Complex Engineered Systems (ICES) at Carnegie Mellon University in Pittsburgh, PA, USA under the supervision of Prof. Asim Smailagic, Ph. D. and Prof. Daniel P. Siewiorek, Ph. D.

The input device used, namely the eWatch, has also been developed at ICES, who kindly provided me with an exemplar to work with. In the course of doing research for this thesis, the direction with respect to recognition methods and requirements was under constant change, either due to approaches that did not perform as expected, or due to new ideas that came up in the meantime. There is no written assignment of task.

As a by-product of the research done in the context of this thesis, a poster was submitted and accepted for the 11th IEEE International Symposium on Wearable Computers. A copy of this poster may be found in the appendix.

Contents

1 Introduction

When using a virtual orchestra system on the computer—with musicians that look and sound quite real—it does not seem to make a lot of sense to interact with it using traditional input devices. Telling the violist to play a little louder by turning the mouse's scroll wheel might be efficient—but is far from the way you would communicate with a real instrumentalist; and making the musicians play faster by hitting the "+" key on the keyboard a couple of times is certainly understood by the orchestra application—but again, this is not the way a conductor tells the orchestra to increase the tempo. The interaction between the user in the role of a conductor and the orchestra simulated in the computer would be much more realistic if the same means of communication which are used in the real world could be used instead, namely conducting gestures. This virtual orchestra is an example for a computer application which benefits from providing alternative ways of communicating with the user, from offering new and more natural means of interaction, making it easier—and certainly much more fun in this case—to use the application.

This thesis presents the theoretical background, implementation, and evaluation of a gesture-based input device for an existent virtual orchestra system. A wearable computer platform resembling an ordinary wrist watch and equipped with several sensors is used to capture acceleration data, and a machine learning approach is then applied to recognize conducting gestures. The user of the system takes over the conductor's role and can control the orchestra's tempo and volume by means of common conducting gestures and by varying execution speed and size. In the following section, I shall introduce the existent orchestra system to be provided with a gesture input device.

1.1 The Pinocchio System

Pinocchio[1] [Brügge et al., 2007] is a virtual orchestra system originally targeted at children to enhance their interest in classical music in a playful manner. It is being developed at the Technische Universität München in cooperation with the Bavarian Radio Symphony Orchestra.

Pinocchio features a virtual concert hall populated by musicians which are represented using synchronized audio and video playbacks. As a key difference to other existent systems the user has the freedom to configure the musicians' setup: Each musician can be selected and positioned freely in 3D space and can be addressed individually. Thus, it is possible to select certain instruments and to realize a virtual walk-through experience featuring a realistic 3D audio representation.

In the current version, the user can control the system with a baton whose movements are captured by a camera. Conducting gestures are recognized in order to control tempo and volume of the virtual orchestra. The audio and video footage used in the system has been acquired recording individual musicians of the Bavarian Radio Symphony Orchestra separately to increase the realism of the virtual orchestra. Currently, recordings of four musicians (two violin players, one violist, and one cellist) playing Mozart's Quartet in C for Strings, "Dissonant" (KV465) are available.

Pinocchio started in 2004 as a first prototype of a conducting system and was called Marris. It was able to track the baton and to perform 3D audio mixing. In 2005, the project was renamed into Virtual Symphony Orchestra (VSO) and real musicians were recorded. It was also possible to position them freely in 3D space. VSO was running on a cluster of six iMacs. VSO++ was introduced in 2006, realizing several optimizations that enabled the conducting system to run on a single machine.

First experiments with alternative input devices have been conducted. Lachenmaier [Lachenmaier, 2006] added a conducting learning module to VSO++, applying artificial neural networks (ANN) to optical gesture recognition. Bierbaum [Bierbaum, 2007] realized a reengineering project optimizing poorly performing parts in order to make Pinocchio capable of handling more complex scores and larger virtual orchestras on a single machine. Besides the work

[1]The name Pinocchio was inspired by a gaze—or nose—recognition experiment. It is an abbreviation for "**P**robability-based **IN**ference **O**f **C**omposition and **C**onducting be**H**aviors in **I**nteractive **O**rchestras".

presented in this thesis, Pinocchio is expanded to support a head-up display which presents the score to the conductor in realtime [Fenzl, 2007].

1.2 Motivation and Challenges

The motivation behind this research is to enable the user to interact with a system in a natural way, i.e. in a way he or she would interact intuitively, considering the system's application domain. In the case of Pinocchio, I shall develop an input device which is able to recognize hand gestures based on acceleration data, i.e. I am focusing on the application domain of musical conducting. The input device will be able to recognize and interpret conducting gestures—which are fairly complex and have a defined meaning—to control a virtual orchestra.

Choosing this specific application domain, however, does not limit the recognition method as such to conducting gestures. Since a general machine learning approach is applied, arbitrary hand gestures can be trained. That is, the hand gesture recognition approach to be developed in the context of this thesis may be applied to other application areas that benefit from a gestural input.

The work presented here covers several areas of computer science, thus posing challenges at different levels: First, in the context of human-computer interaction, adequate gestures have to be defined and a suitable hardware device has to be found and adopted. Second, a recognition algorithm has to be developed and evaluated. Third, the algorithm has to be implemented and embedded into an existent software architecture.

While accelerometers enable small an light devices, they only deliver relative information on motion and provide noisy data—making it more difficult to build a reliable gesture recognition system. The acceleration data thus needs to be preprocessed adequately. Since the user is likely to move the hand randomly during times he or she is not performing an intentional gesture, the recognition system must be able to discard non-meaningful movements without influencing the recognition rate of intentional gestures. In the following section, I shall give a brief introduction of the methods applied in order to realize a conducting gesture recognition system.

1.3 Principle Methods

The choice of input device fell onto the eWatch—a wearable computing and sensor platform developed at the Institute for Complex Engineered Systems at Carnegie Mellon University in Pittsburgh, PA, USA. The device is worn as like regular wrist watch and transmits raw acceleration data of the user's arm movements via Bluetooth to the host system.

After doing a literature research, covering existent orchestra systems on the one hand as well as more general gesture recognition approaches on the other hand, I chose discrete hidden Markov models (HMM) for recognizing conducting gestures. This approach requires several steps in order to preprocess the raw acceleration data, which have to be carefully chosen.

In order to evaluate the proposed gesture recognition approach, I collected several hours of gesture data performed by different users. Three different conducting gestures, representing different measures, were performed by the users to test the system's recognition performance.

1.4 Structure of this Thesis

The remainder of this thesis is structured as follows:

Chapter 2 provides an overview of related work. I shall discuss the application of hidden Markov models (HMM) to pattern recognition tasks before presenting the history of computer-based conducting and orchestra systems. This chapter enables the reader to gain an understanding of pattern recognition application areas and see Pinocchio in the context of similar systems.

In chapter 3 I shall introduce the application domain of musical conducting with a focus on conducting gestures. Additionally, this chapter contains a description of two user studies performed with conductors, and discusses their implications with respect to the development of a new user interface for conducting gestures.

After performing a requirement elicitation at the beginning of chapter 4, I shall explain the decisions I made regarding the general gesture recognition method as well as hardware and software. This chapter defines the foundation for the system to be developed.

Based on this foundation, I shall describe my approach for conducting gesture recognition in chapter 5. After motivating my choice for HMM, I shall discuss the different modules of the gesture recognizer to be realized in the context of this thesis. This chapter makes the reader familiar with the theoretical background the subsequent implementation is based on.

Chapter 6 describes the implementation of the conducting gesture recognizer as well as the integration into Pinocchio. Additionally, I shall introduce software tools I developed for the acquisition of user test data and the evaluation of the recognition performance.

In chapter 7, I shall evaluate the system's performance with regard to detection ratio and reliability. The presented evaluation is based on gesture data acquired from different users. I shall conclude this chapter with a discussion of the results' implications.

Chapter 8 summarizes strengths and limitations of the gesture recognition approach developed in the course of this thesis while chapter 9 shows an outlook on future work, regarding further improvements of the input device, the gesture recognition approach, as well as a general perspective on using and combining sensors in human computer interaction.

2 Related Work

In this chapter, I shall review related work that applies hidden Markov models (HMM) to different pattern recognition tasks—such as hand gesture, hand writing, or activity recognition, to name a few. Since this thesis focuses on the recognition of musical conducting gestures in the context of a virtual orchestra, I shall present prior work related to the application domain of computer-based conducting and orchestra systems separately in section 2.2, independent from the input modalities used.

2.1 Pattern Recognition with Hidden Markov Models

As motivated in section 5.1, I am applying HMM to the recognition of conducting gestures. In this section, I shall present an overview of prior systems and studies, applying HMM to a variety of pattern recognition tasks, while focusing on hand gestures as a general class of conducting gestures. In this review, I am classifying the presented systems by two categories, namely

- recognition type, i.e. which class of patterns does the system recognize, and
- sensor type, i.e. which sensors are used and where are they positioned.

Table 2.1 shows the occurrences represented in the reviewed systems.

Category	**Occurrences**
Recognition type	Hand writing
	2D shapes
	Hand gestures
	Body gestures
	Human activities
Sensor type	Mouse
	Camera
	Accelerometers
	Microphone
	Data glove
	Electro-magnetic position sensor

Table 2.1: Classification categories of HMM-based recognition system

The approach proposed by [He and Kundu, 1991] is able to classify 2D-shapes. [Starner et al., 1994] recognize cursive hand-writing, using discrete HMM. [Yamato et al., 1992] use an image-based approach for human action recognition; the presented system is able to recognize six different human actions, namely different tennis moves.

A variety of systems recognizes hand gestures, either using cameras (stationary or hat-mounted, i.e. wearable), data gloves and electro-magnetic position sensors, or accelerometers (body-worn or device-embedded). Some systems are able to recognize continuous gestures while others are restricted to the recognition of isolated gestures only. [Starner and Pentland, 1995, Starner et al., 1997, Brashear et al., 2003] present sign language recognizers, using either cameras, or a combination of camera and accelerometers. Several systems are designed for the recognition of various hand gesture commands—reaching from controlling TV sets or cell phones to steering robots—such as [Yang and Xu, 1994, Morguet and Lang, 1998, Rigoll et al., 1998, Iba et al., 1999, Mäntylä et al., 2000, Pylvänäinen, 2005, Kela et al., 2006]. [Campbell et al., 1996] propose a system that is able to recognize T'ai Chi gestures, using a stereo camera setup.

[Lukowicz et al., 2003, Minnen et al., 2005] focus on the recognition of human activities for the realization of context-aware applications. Both are using accelerometers and microphones in order to accomplish this task. Further activity recognition approaches are presented by [Krause et al., 2005, Krause et al., 2003]. These approaches do not apply HMM, but they are using the eWatch, respectively a predecessor of the eWatch, as input device, which is also used in order to realize conducting gesture recognition in the context of this thesis (compare section 4.3). In the following section, I shall explain how conducting and virtual orchestra systems developed.

2.2 History of Conducting Systems

Over the last 35 years[2], a multiplicity of systems have emerged around the topic of conducting a computer–or the computer following a conductor. Aiming at different goals, different approaches have been developed. Nevertheless, some ideas remain the same across many of the conducting systems.

This section consists of a structured overview of existent computer-based conducting systems[3] and the approaches they apply. This survey puts an emphasis on the gesture recognition part of the systems.

Section 2.2.1 starts with a brief description of an often used input device called the *Buchla Lightning*. All the systems that have been considered are also briefly described in short paragraphs which are categorized by their respective authors and ordered by their publish date. Two articles that do not include a detailed system description and mainly deal with studying the relationship between conductor and orchestra are presented in chapter 2.2.20. Chapter 2.2.21 closes this section with a comparison of the presented systems.

In addition, figure 2.1 shows a timeline which emphasizes the temporal sequence and illustrates dependencies among the systems. Table 2.2 summarizes relevant innovations and technologies together with their first appearance in a conducting system. A comparison of more general frameworks like *EyesWeb* or *Max/MSP* that are used in some of the presented conducting systems can be found in [Grüll, 2005].

2.2.1 Buchla Lightning

Before the introduction of the actual conducting systems, it is useful to have a look at a commercially available input device as it is used by many of the presented systems. In his review of the *Buchla Lightning II* which is used by several conducting systems, [Rich, 1996] discusses its features and capabilities. While the basic concepts remain the same, the version II adds internal sound, improved wands, and new software features.

Two hand held wands with an infrared emitter and a trigger button, resembling "high-tech drumsticks", are used as control devices. Detectors track position and trajectory of each wand in 2D space. The system can operate at two different ranges: At up to 2.4m and at up to 4.8 m, impacting the possible operation time.

Space is divided in four times two zones. Each zone can be used to define different behaviors. In general, MIDI events are assigned to hand gestures, but the system comes with a powerful software offering its own, object-oriented programming language. Therefore it is possible to define complex interactions.

Although Rich emphasizes the system's capabilities to be used as an instrument, they also mention the already included conduct program. The user can program conducting patterns by assigning a wand motion (that is, up, down, left, or right strikes) to each beat. The program

[2]Considering Mathews' *Conductor Program* mentioned in 1970 [Mathews and Moore, 1970] as first conducting system (compare 2.2.2).

[3]Although a considerable effort has been put into researching the history of computer-based conducting systems, the presented list might not be exhaustive.

Innovation	**System**	**Author (see chapter)**	**Year**
Conducting application	GROOVE/Conductor Program	Mathews (2.2.2)	1970
Stick interface	Sequential Drum	Mathews (2.2.2)	1979
Baton interface*	MIDI Baton	Keane et. al. (2.2.4	1989
Image-based recognition	Computer Music System that Follows a Human Conductor	Morita et. al. (2.2.5)	1989
Fuzzy logic	On-line Analysis of Music Conductor's Two-Dimensional Motion	Bien et. al. (2.2.7)	1992
Artificial neural networks	Adaptive Conductor Follower	Lee, M. et. al. (2.2.8)	1992
Acceleration-based recognition	Gesture Analysis Using 3D Acceleration Sensor for Music Control	Sawada et. al. (2.2.5)	1995
3D sound	Virtual Orchestra Performance	Ilmonen et. al. (2.2.12)	1997
HMM, recognition of common conducting gestures	A Multi-Modal Conducting Simulator	Usa et. al. (2.2.13)	1998
Animated, virtual musicians	Conductor Following with Artificial Neural Networks	Ilmonen et. al. (2.2.12)	1999
Unmodified baton for image-based recognition, left hand	Visual Interface for Conducting Virtual Orchestra	Segen et. al. (2.2.15)	2000
Real orchestra recordings	Personal Orchestra	Borchers et. al. 2.2.11)	2001
Recognition of expressive gestures	Recognition, Analysis and Performance with Expressive Conducting Gestures	Kolesnik et. al. (2.2.16)	2004
Variable orchestra configuration	Conducting a Virtual Orchestra	Schertenleib et. al. (2.2.18)	2004

Table 2.2: Innovations and their first appearance in conducting systems

*Although [Mathews, 1989] presented the *Mechanical Baton*, it resembles more a drumstick from the descriptions available.

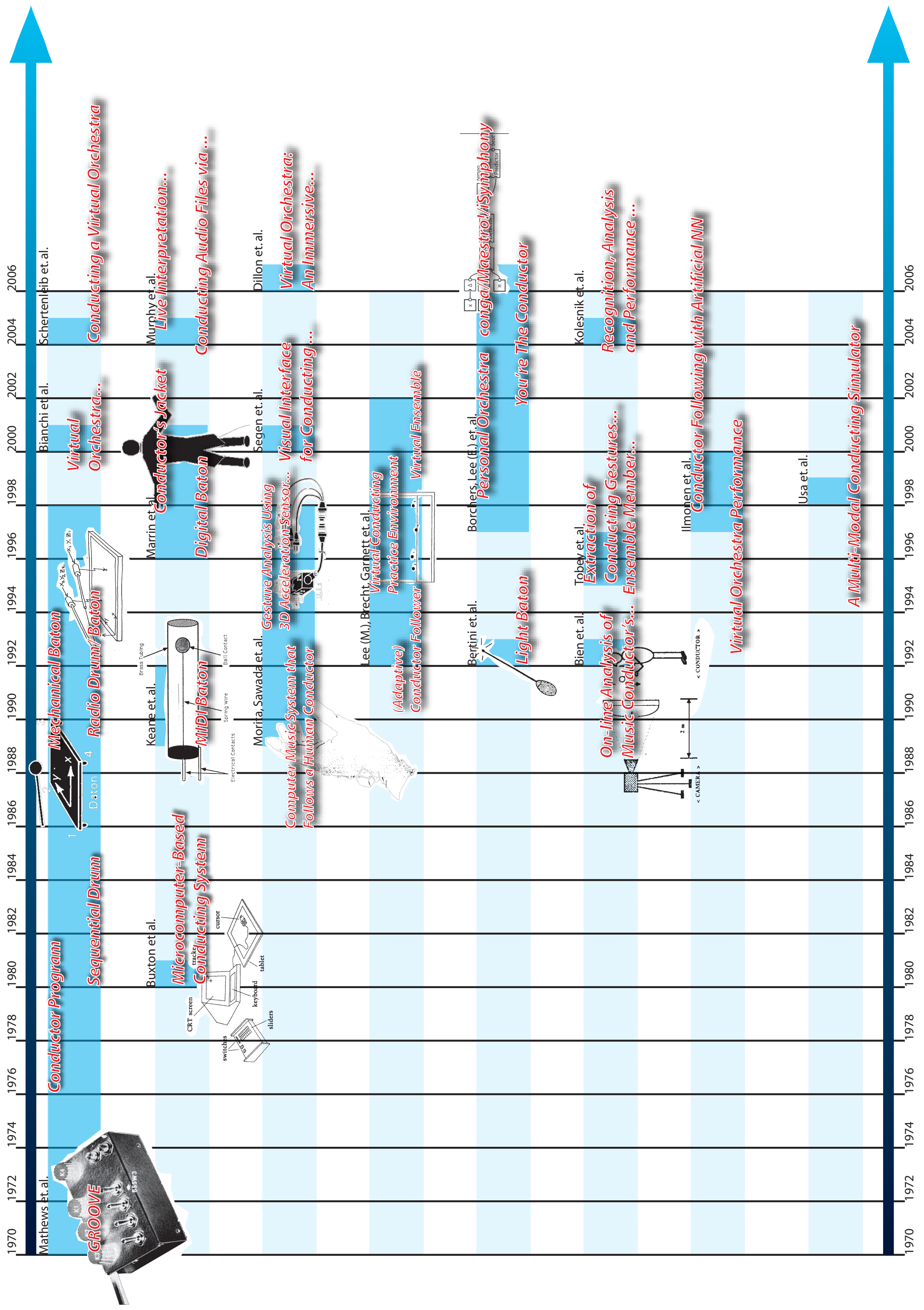


Figure 2.1: Timeline of conducting systems

derives tempo and generates MIDI clocks. Rich notes that the conduct program works best with simple up and down strikes. It seems to be difficult, though, to conduct more complex pattern like patterns used by orchestral conductors due to low recognition accuracy.

2.2.2 Mathews et. al. (1970 – 1997)

GROOVE and Conductor Program

Although [Mathews and Moore, 1970] presented a general purpose system for "creating, storing, reproducing, and editing functions of time", *GROOVE* has only been used for sound synthesis—with an application called the *Conductor Program*. The system reads stored samples from memory, combines these samples with realtime input from the user, and computes output samples for control. Four knobs, a 3D joystick, and a keyboard are used as input devices.

The *Conductor Program* introduces a new view on the interaction with musical systems: Mathews notes that "the desired relation between the performer and the computer is not that between the player and his instrument, but rather that between the conductor and the orchestra." Thus, the score is stored in the computer while the user influences the way it is played back—making the *Conductor Program* probably the first conducting system.

Some more details about the *Conductor Program* were presented by Mathews in 1976 [Mathews, 1976]. He emphasizes again that the supported functions are typically those which are done by a conductor. Possible areas of application are playing new musical compositions, serving as an accompaniment device, and "active listening", an application that allows the user to be an active participant in the performance of a piece of music by conducting and interpreting it [Boulanger et al., 1990].

The *Conductor Program* consists of three modes: Entering the score, rehearsal—that is adjusting the manner of playing—and performance. In performance mode, all voices play together. Overall dynamics and balance of individual voices can be controlled by using the joystick. There are two different ways to control the tempo: Either the performer uses the knob or he or she can exercise control by beating on a key—an interaction already resembling the one of input devices used later.

Sequential Drum

Presented in 1979, the *Sequential Drum* [Mathews, 1979, Mathews and Abbott, 1980] is the first of three attempts by Mathews to create a mechanical baton [Boulanger et al., 1990]. It is made of a rectangular surface that can be struck either by hand or by a drumstick. The system's output consists of a trigger whenever the surface was hit, along with information about the strike's strength (measured by microphones), and the strike's x and y position (determined from grounding wires).

Although the presented application controlling a synthesizer is able to follow the user's tempo inducted by his or her beats and to adjust the loudness according to the strikes' strength, the *Sequential Drum* was rather intended to be a new way of playing "intelligent instruments"; the term "conducting" as such was not thought of in this context.

Mechanical Baton (Daton)

In 1989, [Mathews, 1989] started using the *Conductor Program* in conjunction with a *Mechanical Baton* (or *Daton*), the successor of the *Sequential Drum*. It consists of a 14 inch square plate equipped with a strain gauge in each of the four corners. When the plate is struck with a relatively soft drumstick, the beat's position and strength are computed using the four electrical impulses generated by the strain gauges. This information is used to control the tempo. As can be seen from this setup, the *Daton* cannot produce continuous values. Therefore, the system uses a joystick and knobs as additional input devices to adjust volume and loudness, for example.

In this system, the performer is a conductor of an orchestra rather than a player of an instrument. In this context, Mathews also notes that “the program simulates a number of the functions of an orchestra following the baton of a conductor”. Like in an orchestra, the user does not have influence on the pitch; Mathews gave the task of pitch selection to the computer.

Besides containing information about pitches and durations of notes, the score also contains information about expected strokes which is used to control the tempo. According to Matthews, “each Daton stroke after the first stroke sets the tempo that will be used until the next stroke”. If a stroke occurs before the next beat mark in the score, the system will jump to this position. If the computer arrives at the beat first, it halts and waits for the next stroke. This tempo control makes the system follow the conductor very closely, a behavior which was preferred by users who tried the program.

Mathews proposes two potential usage scenarios. The system could be used as an accompaniment of soloists (as a cheaper alternative) or as an interactive record where the music appreciator “would purchase a computer score and conduct his own interpretation of the piece” rather than just listening to recorded music[4].

Radio Drum

While looking quite similar at a first glance, the *Radio Drum, Boie89*, also introduced in 1989, uses a different technology and is more powerful than the *Mechanical Baton*. It relies on the principle of capacitance sensing of electromagnetic waves. The surface is equipped with an array of receiving antennas which receive signals sent by transmitting antennas built into the drum sticks.

In contrast to the *Mechanical Baton*, an arbitrary number of drumsticks can be used as long as they operate at different frequencies. Furthermore, the *Radio Drum* continuously measures the sticks’ positions in three dimensions. While the accuracy of the x and y axis is about one percent of their maximal value when the stick is close to the surface, it decreases non-linearly with increasing z distance.

Creating two virtual surfaces the *Radio Drum* also works as a trigger. This mode was applied when the *Radio Drum* was used as an input device for the *Conductor Program* [Mathews, 1976]. Although this program takes continuous values as inputs, too, [Boie et al., 1989] does not mention using the *Radio Drum* in this way. In the related article, the device is seen as a controller for music, rather than a tool to actually conduct.

Radio Baton

In 1991, [Mathews, 1991] noted that it is possible to have different configurations of the *Radio Drum*; such a configuration is not being presented, though. The *Radio Drum* [Boie et al., 1989] is a special version of the *Radio Baton* with receivers located in a three-dimensional volume.

Interestingly, [Boulanger and Mathews, 1997] uses the term *Radio Baton* for the same device which was previously called *Radio Drum*. It has one square antenna and two batons. Besides the already discussed *Conductor Program* a second mode for improvisation is presented. The baton sends triggers and x, y, and z coordinates, when requested, to an application which is supposed to interpret this information. While the *Radio Baton* only delivers control information, Boulanger notes that “the user is expected to program their own improvisation algorithms”.

2.2.3 Buxton et. al. (1980)

A Microcomputer-Based Conducting System

Similar to Mathews’ *Conductor Program* [Mathews, 1976],[Buxton et al., 1980] built a system that enables controlling a digital sound synthesizer. They explicitly named it a conducting system.

[4]Mathews called this idea “active listening”, too [Boulanger et al., 1990].

Nevertheless, it is important to have a closer look at its capabilities with regard to user interaction in order to understand what is meant by conducting in this context.

The user interface of the presented system consists of a CRT screen, a keyboard for text input, a tablet, and a slider box. It allows the user to adjust eight different parameters (tempo and amplitude, among others), affecting the way stored scores are played back. This adjustment is done by using the controls available in the on screen textual user interface, resembling standard controls present in nowadays graphical user interfaces. They are "spatially distributed on the screen in much the same manner that knobs and dials are distributed on a mixing console." Buxton et. al. mention different techniques for changing values. The user can for example type numbers or use a dragging technique, i.e. moving the cursor to adjust values.

Although the proposed input techniques are advanced considering the year of publication, the system seems to be more a playback tool for synthesized music scores than a conducting application: It allows the user to affect the way music is played back in detail but the required interactions are not the same as an orchestra conductor uses to communicate with musicians. [Mathews, 1976] in contrast mentioned tapping a key in order to control the playback tempo in his system - which comes closer to conducting gestures than adjusting numerical values using a keyboard or a graphic tablet.

The system is a conducting system if conducting refers to influencing parameters in realtime while playing back scores. But it is not a conducting system if conducting means interacting with the system in a way that resembles the interaction between a real conductor and an orchestra. It would not be possible for example to simultaneously conduct live performers and the sequencer, using this system.

2.2.4 Keane et. al. (1989 – 1991)

The MIDI Baton and Successors

[Keane and Gross, 1989] introduced the *MIDI Baton* system in 1989 to simultaneously conduct live performers and a MIDI sequencer. Shaped like a tube, the device resembles a regular baton with a metal weight inside. During acceleration, this weight is forced against the tube's side and a electrical signal is produced. An interface connected via a wire takes care of removing noise and providing the signal to a synchronizer which in turn anticipates the tempo and advances the MIDI sequencer. The timing relationship can be adjusted to conduct either behind or ahead of the beat. Start and stop can be controlled using a foot pedal.

Due to its inherent design, the *MIDI Baton* is only able to provide discrete values that are used for tempo but not for volume control. In a succeeding version [Keane et al., 1990, Keane and Wood, 1991] of the *MIDI Baton*, a wireless transmitter was introduced eliminating the need for a wired connection between the baton and the MIDI unit. Moreover, switches for start and stop were moved from the foot pedal to the transmitter.

2.2.5 Morita, Sawada et. al. (1989 – 1997)

Computer Music System that Follows a Human Conductor

In 1989, [Morita et al., 1989] presented a system that realizes "an improvisational performance in realtime". It is the first conducting system to be based on visual recognition of the conductor's baton. A CCD camera in conjunction with feature extraction hardware is used to track either the baton augmented with a white marker or the conductor's hand wearing a white glove. A conventional PC is then used to recognize the conducting gestures.

Tempo, strength, start, and stop are detected and sent to MIDI control units. To determine the tempo, vertical turning points of the baton are analyzed. The strength is extracted from the path's length. Although Morita et. al. claim to use a "knowledge database of conducting" there is no further discussion on this topic. They apply an adaptive tempo adjustment by inserting a delay between conducting gestures and music playback in order to make the system sound “more natural in human ears”.

The system can be used as a stand-alone application to enjoy conducting but is also intended to be used together with real musicians or as accompaniment for plays or operas.

Beginning in 1990, [Morita et al., 1990, Morita et al., 1991] extended the system described above by a data glove worn on the left hand which is used to capture additional expressions according to the grammar of conducting. The authors try to reproduce the process a human musician goes through while playing an instrument in an orchestra.

The “musical performance expression” (MPX) is introduced as a formalism that captures the “artistic essence” of performance. In contrast to basic music information like pitch or duration of notes, MPX describes parameters such as “crescendo” or “ritardando” and consists of several components.

The user is able to input his or her evaluation of the system’s interpretation enabling the system to adapt to a user’s gestures and preferences. Morita et. al. intend their system to be used as an ensemble member together with human musicians or as an automated accompaniment.

Gesture Analysis Using 3D Acceleration Sensor for Music Control

In 1995, [Sawada et al., 1995], developed a “new musical performance system” that uses 3D accelerometers for gesture recognition which are fixed to the back of the right hand. In this version, the system has two modes of control: Performance control allows the user to direct performance advancements and timbral control is used for mapping kinetic to sound parameters. While not many details on the hardware and its usability are provided, applied algorithms are described more extensively.

[Sawada and Hashimoto, 1997] provide a more in-depth description of a later version of the same system in 1997—however, not covering the timbral control part, which is of lower interest in the conducting context. The presented system is supposed to “enable anyone to play music”. It uses two different approaches for gesture recognition:

On the one hand, maxima in the acceleration data are detected for tempo and volume control. Tempo prediction is applied to improve the system’s response. On the other hand, the system is able to learn arbitrary gesture commands using a recognition based on the distance between feature vectors. Therefore, several kinetic parameters are extracted out of the acceleration data. Using these features, an evaluation value is calculated for each gesture that has been trained before. If the minimum over all evaluation values is below a certain threshold, a gesture is recognized. The start of a gesture is detected by comparing the current acceleration values to the fluctuations observed before.

Sawada and Hashimoto report an accuracy of 100% for the gesture recognition if training data come from the same user and or accuracy of 90% otherwise. It takes between 0.5 s and 1.0 s in order to detect a gesture. The length of expected gestures has to be known beforehand. No further discussion is being held on the integration of the two proposed recognition approaches.

2.2.6 Bertini et. al. (1992)

Light Baton

[Bertini and Carosi, 1992] built a system to effectively synchronize live musicians with electronic synthesis systems in 1992. Similar to [Morita et al., 1989] they also use a visual approach. The baton is augmented with a bright LED on its tip. A CCD camera with an image acquisition board captures images at 25 Hz. The coordinates of the baton are then extracted and used for gesture recognition. Identifying times of minima and maxima of the vertical coordinate is used for the extraction of features like amplitude and duration. The next beat point is predicted by the system and tempo changes are applied smoothly, resembling the response of a real orchestra.

Although [Bertini and Carosi, 1992] mention a prior study leading to findings that indicate that, while the x-axis of the baton's trajectory is relevant for recognizing the downbeats, the y-axis identifies the order number of movements, this information does not seem to be used in the presented system.

2.2.7 Bien et. al. (1992)

On-line Analysis of Music Conductor's Two-Dimensional Motion

In 1992, [Bien and Kim, 1992] also used a visual approach to detect the conductor's gestures. A special purpose hardware takes care of extracting the baton's trajectory at a rate of 15 Hz (referred to as "high-speed"). This information is transferred to a PC which analyzes the gestures. No augmentation for the baton seems to be needed in order to detect its tip.

Motivated by a higher tolerance to noise, a fuzzy logic approach is applied to detect turning points of the baton movement rather than looking at direction change features directly. The system is able to detect the first beat of a measure and derive tempo as well as dynamics by applying simple rules. In addition, it detects static points ("fermata"). Music playback is realized by a 12 channel sound card. Bien and Kim do not mention any attempt to smooth the tempo adjustments in order to make it behave more like a real orchestra.

2.2.8 Lee (M.), Brecht, Garnett et. al. (1992 – 2001)

Adaptive Conductor Follower

[Lee et al., 1992b] use a *Buchla Lightning* and a *Mattel Power Glove* as input hardware for their artificial musician, or conductor follower. They are the first to apply artificial neural networks (ANN) to conducting gesture recognition. Their architecture consists of classification and parameter estimation modules—both using separate preprocessors and ANN. The system is supposed to recognize tempo, volume, and "other performance parameters". Unfortunately, not much detail on the implementation is provided. Also, the role of the *Power Glove* is not discussed.

The authors mention that they use baton movement in one dimension to derive the implied tempo, which is calculated from time points within a beat to achieve a faster responds time and to be able to vary tempo within a beat. ANN are used to approximate the tempo. It is noted that an adjustment for each conductor is recommended because of the wide variation in gestures.

Later in 1992 [Lee et al., 1992a], a slightly different version of the system is reported. Instead of the *Buchla Lightning*, Mathews' *Radio Baton* [Mathews, 1991] is used as input device. The software architecture remains the same, focusing on simultaneous classification and parameter estimation. In this version of the system, the data glove is used as a mode controller. Two different hand gestures can be recognized, one for volume and one for tempo, while the baton's velocity is mapped to either one of the parameters.

Conductor Follower

In enhancing the system proposed by [Lee et al., 1992b, Brecht and Garnett, 1995] introduced different models for tempo recognition and prediction in 1995. While the algorithm originally used suffered from not being able to change the tempo rapidly enough, a refined version maps more points along the beat's curve in order to enable more frequent updates. Another proposed model uses ANN which output the probability of the next beat happening as well as the according beat subdivision. Although claiming that this system is the first to use "standard conducting gestures", no further details are provided on this topic.

Virtual Conducting Practice Environment

In 1999, taking a different approach compared to the previously discussed systems, [Garnett et al., 1999] pursued an educational intention. Their *Virtual Conducting Practice Environment* is clearly targeted at conducting students, thus it offers graphical and auditory feedback in order to improve a student's learning curve. It uses the *Buchla Lightning* system. Besides an approach to detect different articulation styles, no information about the actual process of gesture recognition is presented.

Virtual Ensemble

In 2001, [Garnett et al., 2001] capture position and orientation information of both the user's hands and his or her head, using three wireless magnetic sensors with six degrees of freedom each. The proposed systems aims at being used in conductor training and as a conductor-based performance instrument. Only preliminary recognition methods are presented. Two beat types can be recognized with an accuracy of 85% applying a hybrid approach consisting of a segmentation step and an ANN for classification. Neither tempo nor dynamics recognition is discussed.

2.2.9 Tobey et. al. (1995 – 1996)

The Ensemble Member and the Conducted Computer

In 1995, [Tobey, 1995] presented a conductor-following system created by a professional conductor aiming at the synchronization of live performers and synthesizers as well as conductor training. A Buchla Lightning is used as input device for this MIDI-based system. Tempo is tracked along the continual shape of the baton's path using information from lower and upper turning points as well as hints coming from the acceleration in between. Tobey calls this approach "rubato control". The system also provides a training module which can be used to make adjustments to an individual conductor's style.

Extraction of Conducting Gestures in 3D Space

In 1996, [Tobey and Fujinaga, 1996] extended the previously presented system by adding another *Buchla Lightning* to the setup in order to be able to track the conductor's gesture in three dimensions. The additionally available z-axis is used to track movements away and towards the conductor's body which was found to be another important dimension in the communication with the orchestra.

In addition to tempo control, this revised version of the conducting system also provides control over dynamics, beat pattern and style, accentuation, and timbral balances. Besides the use in a hybrid computer-human ensemble, Tobey mentions that this system can also be used as a virtual orchestra.

2.2.10 Marrin et. al. (1996 – 2000)

Digital Baton

Marrin's [Marrin, 1997, Marrin and Paradiso, 1997] motivation for building the *Digital Baton* was to create a new instrument for the performance of computer music. On the one hand, it is intended to be used as an instrument for musical control to do both, execute individual notes and provide higher-level control for conducting. On the other hand, it is supposed to be used to perform mouse-pointing and clicking actions.

The baton was built to “replicate as closely as possible the feel of a traditional conducting baton” Weighing about 200 g, it contains an IR LED which is used together with a position-sensitive photodiode (PSD) camera for absolute 2D position tracking at a rate of approximately 50 Hz, accelerometers that provide 3D orientation and acceleration tracking for large gestures and tilt indication, and pressure sensors to measure surface pressure at five points. The baton as well as the PSD camera is connected via cable to a tracking unit which in turn is connected to a computer.

In her Master’s thesis [Marrin, 1996], Marrin develops a theoretical framework for conducting as a special instance of a gesture language. As such, it has similar characteristics as the spoken language and is also constructed from basic elements that are combined into larger structures. Although Marrin intended to build a conducting system, such a system has not been implemented. A beat recognition approach based on the acceleration data has been tested but not been used for tempo tracking.

The *Digital Baton* was used in several performances, though. It was one of the user interfaces in the Brain Opera [Paradiso, 1999] which was conceived and directed by Tod Machover in 1996. This large, public installation consists of a number of different unconventional user interfaces for musical interaction. Every user contributes to a combined artistic experience.

Conductor’s Jacket

[Marrin and Picard, 1998] built the *Conductor’s Jacket* as platform to collect and analyze data from conductors in order to understand how they “express affective and interpretive information while performing”. While the related article only reports on the method of data collection, Marrin’s PhD thesis [Marrin, 2000] presents an in-depth analysis and the *Gesture Construction* system to map gestural data to music.

The jacket holds several sensors, such as electromyography (EMG) sensors for measuring muscle tension, sensors for respiration, temperature, galvanic skin response (GSR), and heart rate. One of the eight constructed jackets also held a magnetic position-sensing device. The jacket is connected to a nearby computer for data collection. More than 10 hours of data were collected from three professional conductors and three conducting students during real performance situations.

Marrin analyzed the data manually by comparing sensor data to video recordings and looking at visual features. Thirty-five significant features were discovered, fourteen of them are relevant for all individuals and described in greater detail. Building on these features, Marrin developed ten hypothetical rules of expression.

As results indicated that muscle tension and respiration signals reflect the characteristics of conducting gestures only these two sensor types are used in the *Gesture Construction* system. The user is able to control several different aspects of musical performance, tempo, dynamics, and articulation, among others. Different filters and rules are applied to the sensor data in order to derive the according parameters. Marrin concludes that being a proof-of-concept, the system did not effectively reflect the character and quality of each gesture.

2.2.11 Borchers, Lee (E.) et. al. (1997 – 2006)

WorldBeat

Having a focus on user interface design issues, [Borchers, 1997] presents a system to be used in an interactive museum exhibit. The system can entirely be controlled by two *Buchla Lightning* batons, which are used for navigational as well as musical input.

Six different modules are implemented with one of them being a conducting module. This module does not rely on the built-in conducting recognition provided by the *Buchla Lightning* system but uses an adoption of the algorithms proposed by [Lee et al., 1992b]. While a simple downbeat turning point detection is used for tempo control, the gesture size is mapped to playback volume. No further tempo prediction or adaptation is applied.

Personal Orchestra

In 2001, [Borchers et al., 2001, Borchers et al., 2002, Borchers et al., 2004] presented the first conducting system that uses pre-recorded audio rather than synthesized music. The *Personal Orchestra* enables the user to control tempo, dynamics, and instrument emphasis with a *Buchla Lightning II* baton. Examining the first derivate of the y coordinate, downward turning points are recognized by detecting changes in sign. No details on dynamics or instrument emphasis recognition are provided. The authors propose an algorithm that allows the music to catch up when the conductor changes tempo. The system has successfully been used as a museum exhibit.

You're The Conductor

Similar to *Personal Orchestra*, *You're The Conductor* [Lee et al., 2004] is an exhibit on display in a museum. Targeted at children, it uses a different interaction style and a different, more robust technical implementation. Instead of a *Buchla Lightning* baton, a custom designed baton made from an aluminum tube with an LED at the tip is used. A PSD camera picks up the emitted light and tracks the baton's trajectory.

The recognition process is fairly simple: Gesture speed is mapped to tempo and gesture size to volume; no beat synchronization takes place. Instead of the pre-stretched audio in *Personal Orchestra*, *You're The Conductor* is capable of realtime audio stretching.

conga Framework and Maestro!/iSymphony

After *Personal Orchestra* and *You're The Conductor*, [Lee et al., 2006a, Grüll, 2005] presented their latest system called *Maestro!*—or *iSymphony*—[Lee et al., 2006c] in 2005. It is also installed as a part of a museum exhibit. A *Buchla Lightning II* is used as an input device. Aiming at building a system that does not need training but at the same time recognizes a variety of gestures the authors propose *conga*, a framework for conducting gesture analysis.

conga is realized as an acyclic graph consisting of linked nodes. Nodes can be either feature detector or data manipulation nodes. In its current implementation, three different gestures can be recognized. A gesture profile selector chooses the best match. From the description provided, it seems to be rather difficult to build a profile for a more complex gesture like the four-beat pattern. The authors report a recognition rate of more then 90% and a latency of up to 675 ms for the four-beat pattern.

Toward a Framework for Interactive Systems to Conduct Digital Audio and Video Streams

In this work, [Lee et al., 2006b] combine two of their systems taking the gesture recognition part from *Personal Orchestra* and the audio and video rendering engine from *You're the Conductor*. As already discussed in [Lee and Borchers, 2005], a semantic time framework based on beats and notes in case of musical applications is introduced. It allows non-uniform spacing between time units and is able to represent the possibly non-linear relation between real and semantic time. Therefore, operations that change the data's time base become possible without losing information about the semantic time.

2.2.12 Ilmonen et. al. (1997 – 1999)

Virtual Orchestra Performance

In 1997, [Takala, 1997] presented a conducting system that features an orchestra of animated virtual musicians. These rule-based agents adjust their tempo as well as other aspects of performance according to the conductor who wears the data gloves. Gesture recognition is performed using ANN. The system uses 3D sound.

Conductor Following with Artificial Neural Networks

Two years later in 1999, [Ilmonen and Takala, 1999, Ilmonen, 1999] used a data suite containing several magnetic motion trackers with six degrees of freedom each to acquire data for conducting gesture recognition. Applying different ANN and heuristic rules using a priori knowledge about the score, the system can detect tempo, dynamics, and articulation. No evaluation concerning the recognition accuracy has been provided. It is noted that the ANN should be trained anew for each conductor.

In 2000, [Ilmonen and Jalkanen, 2000] exchanged the above used magnetic tracker for accelerometers in order to build a lower priced system to be used in a public exhibition. A wired tube was equipped with two orthogonally mounted accelerometers.

Two inherent problems of accelerometers used for position tracking are discussed: Drift and rotation. In order to overcome those, the authors applied a digital signaling processing approach using several low- and high-pass filters. Although it is stated that the results are not comparable to those of magnetic trackers, they were sufficiently satisfying for this application.

2.2.13 Usa et. al. (1998)

A Multi-Modal Conducting Simulator

Building on the model of recognition by actual orchestra musicians, [Usa and Mochida, 1998a, Usa and Mochida, 1998b] presented a conductor follower system that can be used for training purposes. It uses 2D accelerometers attached to a baton at a sampling frequency of 100 Hz and HMM together with fuzzy production rules for gesture recognition. Supporting common conducting gestures as described by Rudolf, the user can control various parameters of the performance like tempo, dynamics, and articulation, among others.

The applied recognition process tries to imitate the way a real musician interprets conducting gestures. It consists of several agents, one of them integrating a HMM that is used to derive the order number of a beat within a measure. Acceleration data are filtered and quantified into 32 labels before used as input for the HMM. Thus, a discrete HMM is utilized. The start of a gesture is determined by detecting local peaks in the acceleration data.

Several other rule-based approaches are applied to detect further parameters, such as dynamics or articulation. In combination with fuzzy production rules that take into account the score and compute probabilities of the next beat occurring, an accuracy of 99.74% is reported if the system is used by the same person who also trained the HMM. The accuracy drops to 98.95% for other users.

In order to realize a more realistic simulation, Usa and Mochida insert a delay between the recognized beats and music playback as a linear function of tempo. The amount of delay to be introduced was determined in an experiment with Japanese conductors. In a later version, Usa and Mochida added eye tracking and breath-sensing to the system.

2.2.14 Bianchi et. al. (2000)

Virtual Orchestra: Technical and Creative Issues

The virtual orchestra system presented by [Bianchi and Campbell, 2000] is intended to be used in professional opera and theater productions as an alternative to the live orchestra. The fairly large system is therefore installed in the pit. It is able to use up to 32 audio channels and follows the conductor's tempo. No details on the recognition algorithms are discussed.

2.2.15 Segen et. al. (2000)

Visual Interface for Conducting Virtual Orchestra

In 2000, [Segen et al., 2000] presented a conducting system which relies on image processing for gesture recognition. Using two cameras, 3D trajectories of the baton as well as of the left hand are reconstructed. The user can control tempo and dynamics using common conducting gestures. While the hand is used for volume control only, the baton controls both, volume and tempo. Volume control by the baton controls the overall volume per measure while the hand controls overall volume per beat (using different levels of elevation) and volume per orchestra section (using a special volume gesture).

Recognition is based on analyzing position data and applying rules. In addition to the auditory output, a virtual dance ensemble consisting of 3D human models is synchronized to the user's input. No information about the system's accuracy is provided.

2.2.16 Kolesnik et. al. (2004)

Recognition, Analysis and Performance with Expressive Conducting Gestures

In 2004, [Kolesnik, 2004, Kolesnik and Wanderley,] used two USB cameras to track the user's left and right hand, wearing colored gloves. While the system recognizes indicative, continuous gestures for the right hand, it recognizes expressive, isolated gestures for the left hand.

Discrete HMM are used in the recognition process running in parallel with a "gesture analysis system" that performs a beat amplitude and transition points extraction based on extrema of absolute positional values. Using this information about temporal segmentation in addition to the probabilities output by the HMM, continuous gestures can be identified. The system is able to identify isolated gestures with an accuracy of 97.2% and continuous gestures with an accuracy of 94.6%.

2.2.17 Murphy et. al. (2004)

Conducting Audio Files via Computer Vision

[Murphy et al., 2004] presented a conducting system that uses one to two cameras in order to track a regular baton. Beats are extracted automatically from audio files to enable mapping between the conductor's beat and beats in the recorded file. Although it is noted that the system uses common conducting gestures, only the tempo can be influenced and no insights into gesture recognition algorithms are given.

Live Interpretation of Conductors' Beat Patterns

Building on the baton tracking system presented above, the author proposes a mathematical model for following and interpreting a conductor's gestures [Murphy, 2004]. Moreover, he notes that it is usually known what beat pattern is going to be issued and that the importance lies in how the conductor executes it. The proposed model determines the meaningful deviation from a neutral execution of a beat pattern.

A beat template represented by a continuous close parametric curve built from the user's execution of the beat pattern is used in the recognition process that keeps track of where the baton is in relation to the whole pattern. No results about the accuracy of this approaches are presented.

2.2.18 Schertenleib et. al. (2004)

Conducting a Virtual Orchestra

In 2004, [Schertenleib et al., 2004] presented a virtual reality system that is targeted at music amateurs providing an "entertaining multimedia experience". The presented work does not intend to advance existing conducting interfaces but tries rather to offer an immersive experience conducting a virtual orchestra.

It is the first system that allows the user to modify the orchestra's layout and to select an arbitrary viewing. While the virtual musicians are presented on a large projection screen, the main input interface is a PDA with an attached magnetic tracker. Simple conducting gestures control the orchestra's tempo and volume using the gestures' frequency and amplitude respectively.

2.2.19 Dillon et. al. (2006)

Virtual Orchestra: An Immersive Computer Game for Fun and Education

[Dillon et al., 2006] present their virtual orchestra system on the one hand as a "fun computer game" that introduces children to music and on the other hand as a "conducting training tool". Using a commercial air gyro mouse, conducting gestures are detected by recognizing sequences of "impulses", that is, fast changes in direction. The user is able to control tempo, dynamics, and articulation.

Besides the MIDI audio output, the system is also able to display virtual musicians. Dillon et. al. implemented a scoring system to evaluate a user's performance based on a comparison between conducted tempo respectively dynamics and those written in the score. Different levels of difficulty are selectable and the virtual audience shows reactions according to how the user performs.

The authors note that is it possible to build a standalone version of the virtual orchestra that is able to run on a standard PC. No evaluation on recognition accuracy or details about the mentioned training part are presented.

2.2.20 Studies

Improving Orchestral Conducting Systems in Public Spaces: Examining the Temporal Characteristics and Conceptual Models of Conducting Gestures

In 2005, [Lee et al., 2005] investigated the temporal characteristics of conducting gestures with regard to beat placement comparing conductors to non-conductors. Determining parameters that makes it possible to clearly distinguish between these two user group allows for building an adaptive conducting system.

Twenty-three users took part in the study. They were instructed to conduct a fixed orchestral recording using a simple up-down-motion, being aware that they can not influence the performance. No automatic recognition was used. Beats in the audio track as well as lower turning points in conducting gestures were annotated manually.

The findings show that conductors lead by an average of 152 ms compared to non-conductors who lead by an average of 47 ms. Also, non-conductors vary their beat-placement 50% more than conductors. In contrast, non-conductors do not make more beat errors than conductors. Additional observations showed that non-conductors often follow the musical rhythm rather than the actual beat. Moreover, as "spiral of death" was observed: In response to a slowdown in music the user slows down the gestures which in turn slows down the music even more. The authors assumed that these findings also apply to an active system which still has to be investigated.

Modeling the Tempo Coupling between an Ensemble and the Conductor

In 2006, [Baird and Izmirli, 2001] analyzed the tempo coupling between conductor and ensemble, that is, the measure indicating "how closely the tempo of the ensemble corresponds to the tempo of the conductor", especially during changing tempos when a difference can be observed.

A position sensor attached to the conductor's baton or index finger is used to gather data from live performances. After creating a compact representation of the coupling, results are generalized and a model is built which is used in a performance system to simulate different levels of ensemble responsiveness.

2.2.21 Comparison

Table 2.3 provides a compact comparison of the presented computer-based conducting systems with focus on the input and gesture recognition part. Since there are many systems with different properties, the table's categories are kept quite general in order to make it possible to fit the systems' characteristics into one single table for easier comparison. Further details can be found in the corresponding chapters of this section. Only properties that are described in the related articles are considered.

The systems are ordered in the same way as in the beginning of this section. Those by the same group of authors are presented consecutively. The following categories are used:

System contains either the name of the system or the title of the corresponding paper if the system does not have a clear name. If a system introduces only minor changes compared to an earlier system, it is listed in the same row.

Input device lists the input devices (and their functional principle, if custom-made) that are available in the corresponding system to enable the user to conduct.

Recognition algorithms categorizes the techniques that are used to detect the user's intention, i. e., algorithms that are applied to translate raw data coming from the input devices into control data used to influence parameters of music playback. Turning point analysis refers to techniques that use position, velocity, or acceleration data in order to determine the point in time where the baton changes its direction, indicating either a down- or an up-beat. Systems that follow a rule-based approach apply rules that combine different facts in order to derive the conductor's intention. When a system uses frequency and amplitude analysis it does not map beats but simply correlates gesture speed and size to tempo and volume, for example.

Controllable parameters lists the musical parameters that can be influenced by the user through conducting. Only parameters that are mentioned in the corresponding articles to be actually used in the systems are listed.

Year	Author (see chapter)	System	Input Device	Recognition Algorithms	Controllable Parameters
1970/1976	Mathews, Moore (2.2.2)	GROOVE/Conductor Program	Knobs, buttons, joystick, keyboard	N/A	Tempo, dynamics (overall and per voice)
1979	Mathews, Abbott (2.2.2)	Sequential Drum	Plate equipped with microphones and grounding wires, stick	N/A	Tempo, dynamics
1989	Mathews (2.2.2)	Mechanical Baton	Plate equipped with strain gauges, stick	N/A	Tempo
1989/1991	Boie, Mathews, Schloss (2.2.2)	Radio Drum/Radio Baton	Plate(s) and sticks equipped with antennas, based on capacitance sensing	N/A	N/A
1980	Buxton, Reeves, Fedorkow, Smith, Baecker (2.2.3)	Microcomputer-Based Conducting System	Screen and tablet, keyboard, slider box	N/A	Tempo, dynamics, etc. (total of eight parameters)
1989 – 1991	Keane, Gross, Wood (2.2.4)	MIDI Baton	Tube with weight that closes circuit when acclerated	N/A	Tempo
1989 – 1991	Morita, Hashimoto, Ohteru, Watanabe, Harada (2.2.5)	A Computer Music System that Follows a Human Conductor	CCD camera and augmented baton, data glove	Baton: turning point analysis, hand: feature vector	Baton: tempo, dynamics, articulation; hand: cresc., rit., etc.
1995/1997	Sawada, Ohkura, Hashimoto (2.2.5)	Gesture Analysis Using 3D Acceleration Sensor for Music Control/Gesture Recognition Using an Acceleration Sensor and Its Application to Musical Performance Control	3D accelerometers	Tempo: frequency analysis, commands: feature extraction, distance to standard patterns	Tempo, commands (e.g. start/stop, etc.)
1992	Bertini, Carosi (2.2.6)	Light Baton	CCD camera, baton augmented with LED	Turning point analysis	Tempo, dynamics, order number of beat, stand still
1992	Bien, Kim (2.2.7)	On-line Analysis of Music Conductor's Two-Dimensional Motion	CCD camera, baton	Rule-based, fuzzy logic	Tempo, dynamics, first beat, fermata

Continued on next page ...

Year	Author (see chapter)	System	Input Device	Recognition Algorithms	Controllable Parameters
1992/1995	Lee (M.), Garnett, Wessel, Brecht, Garnett (2.2.8)	Adaptive Conductor Follower/Conductor Follower	Buchla Lightning (later Radio Baton), Mattel Power Glove	Neural networks	Tempo, dynamics, misc. performance parameters
1999	Garnett, Malvar-Ruiz, Stoltzfus (2.2.8)	Virtual Conducting Practice Environment	Buchla Lightning	N/A	Tempo, dynamics, articulation
2001	Garnett, Jonnalagadda, Elezovic, Johnson, Small (2.2.8)	Technological Advances for Conducting a Virtual Ensemble	Three body-worn 6DOF magnetic motion trackers	Segmentation, neural networks	Beat types
1995	Tobey (2.2.9)	The Ensemble Member and the Conducted Computer	Buchla Lightning	Turning point analysis	Tempo
1996	Tobey, Fujinaga	Extraction of Conducting Gestures in 3D Space	Two Buchla Lightning	N/A	Tempo, dynamics, beat pattern, beat style, timbral balance
1997	Marrin, Paradiso	Digital Baton (2.2.10)	Baton with accelerometers, pressure sensors, IR LED and PSD camera	N/A	N/A
1998/2000	Marrin, Picard (2.2.10)	Conductor's Jacket	Data suite with EMG, respiration, temperature, GSR, heart rate, and magnetic position sensors	Tempo, dynamics, articulation, misc. performance parameters	
1997	Borchers (2.2.11)	WorldBeat	Two Buchla Lightning	Turning point analysis	Tempo, dynamics
2001 – 2004	Borchers, Lee (E.), Samminger, Muser, Mühlhäuser (2.2.11)	Personal Orchestra	Buchla Lightning	Turning point analysis	Tempo, dynamics, instrument emphasis

Continued on next page ...

Year	Author (see chapter)	System	Input Device	Recognition Algorithms	Controllable Parameters
2004	Lee (E.), Marrin, Borchers (2.2.11)	You're The Conductor	PSD camera, baton with IR LED	Frequency and amplitude analysis	Tempo, dynamics
2006	Lee (E.), Grüll, Kiel, Borchers, Dedenbach, Karrer, Wolf (2.2.11)	conga/Maestro!/iSymphony	Buchla Ligthning	Rule-base graph approach	Tempo, dynamics, instrument emphasis
1997	Takala (2.2.12)	Virtual Orchestra Performance	Data gloves	Neural networks	Tempo, misc. performance parameters
1999/2000	Ilmonen, Takala, Jalkanen (2.2.12)	Conductor Following With Artificial Neural Networks	Data suite with 6DOF magnetic motion trackers	Neural networks, heuristic rules	Tempo, dynamics, nuancing
1998	Mochida, Usa (2.2.13)	A Multi-Modal Conducting Simulator	Baton with 2D accelerometers	HMM, fuzzy rules	Tempo, dynamics, articulation start/stop, fermata
2000	Bianchi, Campbell (2.2.14)	Virtual Orchestra	N/A	N/A	N/A
2000	Segen, Gluckman, Kumar (2.2.15)	Visual Interface for Conducting Virtual Orchestra	Two cameras, baton and hand tracking	Turning point analysis, rule-based	Tempo, dynamics, beat phase
2004	Kolesnik, Wanderley (2.2.16)	Recognition, Analysis and Performance with Expressive Conducting Gestures	Cameras (tracking both hands)	HMM	Tempo, expressive control
2004	Murphy, Andersen, Jensen (2.2.17)	Conducting Audio Files via Computer Vision	One or two cameras, baton	N/A	Tempo
2004	Murphy (2.2.17)	Live Interpretation of Conductors' Beat Patterns	One or two cameras, baton	Mathematical model	N/A
2006	Schertenleib, Gutierrez, Vexo, Thalmann (2.2.18)	Conducting a Virtual Orchestra	Magnetic tracker attached to PDA	Frequency and amplitude analysis	Tempo, dynamics

Continued on next page ...

Year	Author (see chapter)	System	Input Device	Recognition Algorithms	Controllable Parameters
2006	Dillon, Wong, Ang (2.2.19)	Virtual Orchestra: An Immersive Computer Game for Fun and education	Air gyro mouse	Turning point analysis ("impulses"), rule-based	Tempo, dynamics, articulation

Table 2.3: Comparison of conducting systems

2.2.22 Discussion

Due to their great variety in a number of characteristics, it would be quite difficult to compare the presented systems in general. In fact, it seems to be more beneficial to focus on the respective input devices and recognition techniques used, since this is the emphasis of the present thesis as well.

Several types of sensors have been used for conducting gesture recognition: Cameras in conjunction with more or less augmented batons, muscle tension, heart rate, respiration, and galvanic skin response sensors, data gloves, magnetic motion trackers, and accelerometers. For the actual recognition, most of the systems rely on only one of those sensor classes. Although [Marrin, 2000] used a combination of several sensor types to conduct a user study, only two of them were used in her recognition system. As can easily be seen from table 2.3, the majority of systems use a visual approach to capture the conductor's movements; almost half of them chose the *Buchla Lighting* as input device. Accelerometers are only part of three of the adopted input devices. Several other approaches were taken, ranging from magnetic tracking over custom-built approaches (especially in the early years) to combinations of a multiplicity of different sensors.

A second important choice, that has to be made when designing a computer-based conducting system, is which recognition technique to use (including feature extraction and representation). Together with the input device, this decision defines the system's limits in terms of possible variety of recognizable gestures, and thus the number of controllable parameters as well as the achievable recognition accuracy.

Most of the presented system use a rule-based approach, often analyzing the turning points of the baton's trajectory in order to recognize beats. While one approach develops a fairly complicated mathematical model [Murphy, 2004], another system incorporates a recognition process that uses a graph structure with nodes for data manipulation and feature detection [Lee et al., 2006a]. Four of the systems make use of ANN, two of HMM.

In this context, it is important to have the intended application in mind. Several of the presented systems are intended to be used in museum exhibits where a single user only has a very limited time to get familiar and interact with the system. In such a case, it does not make sense to provide a large number of controllable parameters, for example. Others put an emphasis on conductor training applications. In this area, it might be desirable to support a larger set of conducting gestures. Some of the systems are designed for playing along with human musicians. Here, it is important to reliably support at least a basic set of parameters that allow the system to follow the conductor. Another application area can be found in the entertainment category.

It is interesting to note that the output capabilities changed over the years as well. Beginning with systems that controlled a synthesizer, the MIDI standard is used in the majority of the systems. A more realistic performance was achieved by systems that used prerecorded audio which was stretched according to tempo. In addition, several approaches for visual output were presented, ranging from virtual, animated characters to professional recordings of real symphony orchestras.

In this chapter, I presented an overview of approaches that apply HMM to pattern recognition tasks on the one hand, and the history of computer-based conducting and orchestra systems on the other hand. In the course of the document, I shall refer to the prior worked mentioned in this chapter in the course of the document, where applicable. In the next chapter, I shall give an introduction to the application domain of conducting and describe initial user studies which were performed in the context of this thesis.

3 Preceding Studies

After presenting an overview of related work in the previous chapter, I shall briefly introduce the application domain of conducting—with a focus on conducting gestures—at the beginning of this chapter. In section 3.2, I shall describe two initial user studies with conductors and their implications for the gesture recognition approach to be developed.

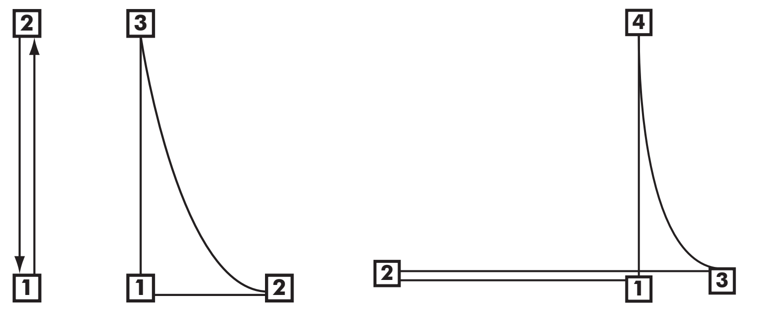
3.1 Conducting Gestures

The following comments are mainly based on [Rudolf, 1950], who claims to be the first publisher of a textbook on conducting techniques. When interacting with an orchestra, the conductor takes the role of a musical leader; he is part musician and part actor. The task of directing an orchestra is complex, since many musical details have to be communicated.

A conductor uses his gestures in order to convey his intention to the orchestra. Besides indicating tempo, much more information may be included in conducting gestures, such as dynamics or articulation—information that is required to achieve an artistic result. Usually, the right hand is used for time beating, while the left hand performs expressive gestures.

[Rudolf, 1950] drastically emphasizes the importance of a good baton technique by stating that "a conductor who fractured his left arm would still be able to exercise complete control of his group, provided that he had a good baton technique." He notes that an orchestra can be directed without a baton as well—it is the technique of performing gestures that is relevant. However, a baton makes it easier for musicians to follow the conductor which is especially important for large ensembles with a considerable distance between musicians and conductor.

In the grammar of conducting, there are different gestures defined for the indication of different types of measures, e.g. 2-beat or 3-beat measures. These gestures can be performed at varying speed in order to communicate tempo, and at varying size in order to communicate dynamics. Moreover, the gestures may vary in style in order to communicate articulation.



(a) 2-beat (b) 3-beat (c) 4-beat

Figure 3.1: Light-staccato conducting gestures [Rudolf, 1950]

Figure 3.1 contains three diagrams which illustrate how light-staccato conducting gestures for 2-, 3-, and 4-beat measures are supposed to be performed. These diagram depict the baton's movement in a plane, i.e. contain information in which order up, down, left, and right movements have to be performed. Squares indicate short halts in the movement, the numbers denote beats within a measure.

For the remainder of this thesis, I chose light-staccato over alternative articulations, such as non-espressivo or legato, since the according gestures are relatively easy to learn and perform and contain the same information with regard to time beating. I shall focus on developing an approach which recognizes conducting gestures of three different measures, performed in light-staccato, and which is able to extract conveyed information about tempo and dynamics. Pinocchio in its current version is able to process this information and influence the virtual orchestra's performance accordingly. Since [Rudolf, 1950] indicates that conducting gestures can be performed with or without baton, I decided not to use a baton for the evaluation presented in this thesis for practical reasons.

An evaluation on how the usage of a baton affects the data captured by the accelerometers, and thus the recognition performance, is left to future research (compare chapter 9). Additionally, different styles of articulation may be added in the future by means of simply training the recognition module accordingly. Future work may also include an extension that enables the recognition of additional expressive features.

3.2 Initial User Studies

Daniel Meyer At the beginning of this research project, I was given the opportunity to perform a first user study with conductor Daniel Meyer[5] during a rehearsal of the Pittsburgh Youth Symphony Orchestra. Figure 3.2 shows a picture of him wearing the eWatch.

The intention of this study is twofold: First, I am interested in the interaction between a conductor and his orchestra, especially with respect to timing. Second, I am interested in the eWatch's suitability as an input device to capture conducting gestures. That is, I am investigating if the acceleration data provided by the eWatch contains sufficiently enough information for gesture recognition on the one hand, and if the user is comfortable wearing the eWatch while conducting on the other hand.

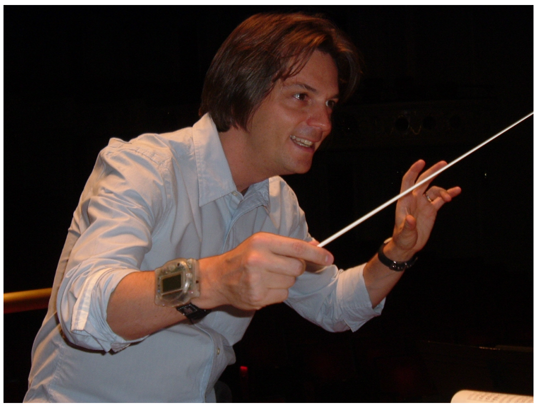


Figure 3.2: Conductor Daniel Meyer wearing the eWatch during a rehearsal

During the rehearsal, I recorded Daniel Meyer using a Panasonic DVC-30 digital camcorder while he was wearing the eWatch. Additionally, I positioned a microphone (connected to the camcorder) close to the conductor's platform in order to capture the same audio information observed by the conductor at the same time. Using this recording setup, I captured video paired with audio footage as well as acceleration and light data. The synchronization of the resulting two data sources, which is a required processing step to be able to annotate acceleration data by analyzing image sequences, is described in detail in section 6.4.

I selected a two minute long section from Brahm's Symphony No. 3 in F major, Allegro for further analysis. That is, I manually annotated the conducting gesture's lower turning points—indicating beats—in the acceleration data with the help of the simultaneously recorded video. Since a reliable audio beat extraction tool is not available, I implemented a tool in MATLAB which allows to play a digital music file and tap along the beat by hitting a key repeatedly. The recorded beat times are then adjusted to match the actual music beats using an audio editor. Comparing

[5]Daniel Meyer is resident conductor of the Pittsburgh Symphony Orchestra and the music director of the Pittsburgh Youth Symphony Orchestra [Meyer, 2007].

conducting to music beats, it turns out that the conductor is ahead by 272 ms on average in the selected piece.

Furthermore, the results show that the eWatch's acceleration data can be used to identify conducting beats manually. Its suitability for automatic beat identification—or more general for conducting gesture recognition—is discussed in section 4.3. During the rehearsal, Daniel Meyer was wearing the eWatch for approximately three hours with intermissions. He stated that he was aware of wearing the eWatch while conducting. However, he was able to perform conducting gestures as usual. These results affirm that the eWatch is a suitable input device for a virtual orchestra system.

This user study's results were integrated into a poster which was accepted for the 11th International Symposium on Wearable Computers [Schmidt et al., 2007]. The according paper is shown in figures 11.2 and 11.3; the poster presented at the conference's poster session is shown in figure 11.4 in the appendix.

Walter Morales Analyzing the previous recording with Daniel Meyer, which was taken during a live orchestra rehearsal, I realized that there is more information conveyed in conducting gestures—thus making the gestures very complex—than can be recognized and interpreted in a first approach of building a conducting gesture recognizer. Therefore, I chose to use a more controlled environment in this second user study by restricting the types of gestures to be performed with regard to measure, tempo, and articulation.

For this user study, I asked conductor Walter Morales[6] to perform different conducting gestures following a metronome on the one hand, and selected pieces of classical music of different measures on the other hand. This user study's goal is to learn more about how conducting gestures are performed; watching a conductor doing so gives much more insight than referring to a text book only. A description of the recording setup may be found in section 6.4.

In this user study, the conductor was wearing the eWatch differently in order to collect data that better capture a conducting gesture's features (compare section 4.3). Additionally, he performed the same gestures with and without baton for later analysis of potential differences (compare section 9.3). The video footage's analysis was a valuable help in understanding the nature of conducting gestures. While, at the beginning, the gestures performed by Walter Morales seemed to be very complex and even not accurate for the untrained eye, a frame by frame analysis revealed that the beat placement matched with the metronome's ticks, in fact. Figure 3.3 shows an image sequence of Walter Morales performing a 2-beat conducting gesture.

In this chapter, I explained how conducting gestures are performed and which information may be communicated during conducting. Moreover, I presented results of two user studies with conductors performed in different environments. These studies affirmed the eWatch's suitability as input device for a virtual orchestra system. In the next chapter, I shall introduce design decisions made with regard to the realization of this project.

[6]Walter Morales is the music director of the Edgewood Symphony Orchestra as well as the music director and conductor for the Carnegie Mellon Contemporary Ensemble. He is also the assistant director of orchestral studies at Carnegie Mellon University and assistant conductor for the Carnegie Mellon Philharmonic [Morales, 2007].

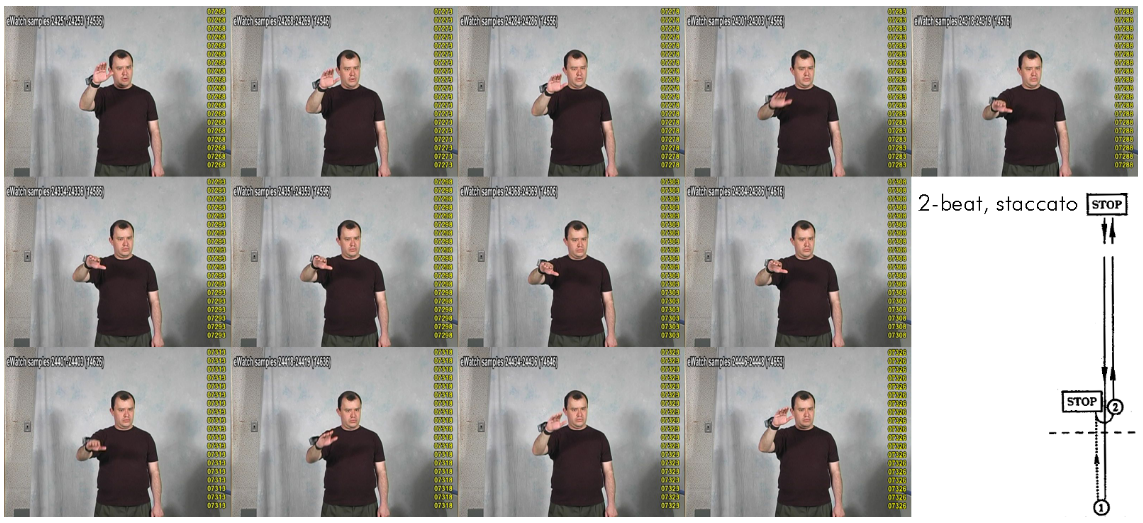


Figure 3.3: Conductor Walter Morales performing a 2-beat conducting gesture

4 Design Decisions

Based on a requirement elicitation at the beginning of this chapter, I shall explain the decisions I made regarding the gesture recognition method as well as hardware and software to be used in the realization of the gesture recognition module. Furthermore, I shall show the reasoning that led to these decisions, as a consequence of the requirements on the one hand, and the components' availability on the other hand.

Due to the fairly comprehensive nature of integrating a gesture recognition module into an existing software system—potentially requiring a complex approach—there are multiple decision steps at different levels to be taken. Figure 4.1 depicts the different decision steps that led to these decisions and illustrates how they depend on each other.

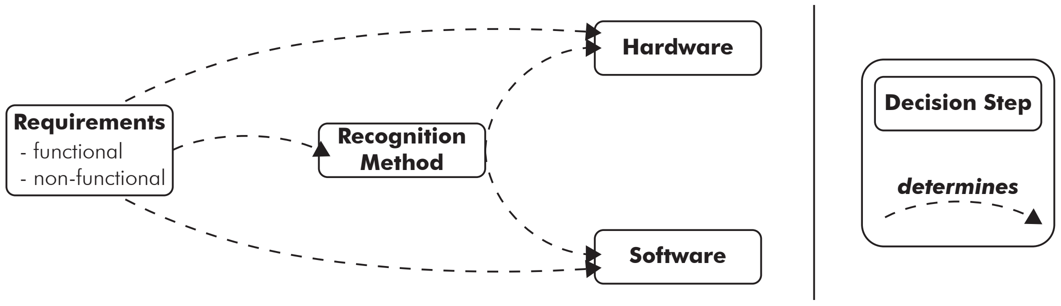


Figure 4.1: Decision steps

Deciding on a recognition method, specific hardware components, and software tools defines the resulting system's capabilities as well as its limitations, which I shall also discuss in the following sections. The remainder of this chapter starts with a requirements elicitation, identifying functional and non-functional requirements, and continues with a discussion about recognition method, hardware, and software choices.

4.1 Requirements Elicitation

For a clear definition of the requirements relevant to the context of this thesis, one has to differentiate between the requirements for the Pinocchio system in general and the requirements for the conducting gesture recognition part (the subject of this thesis) in particular. I shall not cover an elicitation of requirements for the Pinocchio system in general; this has already been discussed and can be found in [Bierbaum, 2007] and [Fenzl, 2007].

In fact, I shall focus on those requirements which are relevant for the decisions to be made regarding the gesture recognition subsystem in the remainder of this section. Following the terminology described in [Brügge and Dutoit, 2003], I shall stick to the distinction between functional and non-functional requirements to provide a more structured compilation of requirements.

I identified the functional requirements listed in table 4.1, which "describe the interactions between the system and its environment independent of its implementation" [Brügge and Dutoit, 2003, page 125]. To simplify matters, the term "system" is meant to refer to "gesture recognition subsystem" in the following tables.

Table 4.2 lists the identified nonfunctional requirements, which "describe aspects of the system that are not directly related to the functional behavior of the system" [Brügge and Dutoit, 2003, page 126]:

Functional Requirement	Description
Gesture identification	The system must be able to find and classify common conducting gestures (compare [Rudolf, 1950]) which are continuously performed by the user with his or her arm and hand. This requirement is twofold: The system needs to segment the continuous user input on the one hand, and to recognize the type of gesture that has been performed on the other hand.
User-independent recognition	The system must be able to recognize conducting gestures from different users at a given time without changing the system's implementation.
Extensible recognition approach	The chosen recognition approach must be extensible, that is, it must be possible to setup the system in a way that enables it to recognize arbitrary gestures performed with the user's arm and hand without changing the system's implementation. One must be able to add a conducting gesture for a different measure easily, for example.
Gesture interpretation	In addition to the identification of gestures, the system must be able to interpret the gestures it identified. In the context of conducting gestures this means that (at least) the intended tempo and volume as conveyed in the gestures need to be extracted by the system.

Table 4.1: Functional requirements of the gesture recognition subsystem

4.2 Recognition Method Decision

Before a final decision on either the hardware or the software to be used is possible, after reviewing the requirements introduced in the previous section, there is one more step in the chain of decisions to be tackled. Decisions have to be made on

1. what class of input device will be used and on
2. what type of recognition algorithm will be applied (this decision is closely related to the previous one since the recognition algorithm depends on the data provided by the input device).

It is necessary to make these decisions at this point in order to choose the appropriate hardware, and later the software. As presented in section 2.2.21, previous conducting systems were using a broad variety of different input devices (ranging from knobs and drum sticks via cameras and data gloves to other type of sensors) and a few different recognition approaches (rule-based recognition and machine learning approaches).

Class of Input Device Reviewing the requirements defined in section 4.1 and bearing in mind the onsite availability of the eWatch device (for details see 4.3), I chose to use an input device incorporating accelerometers for the following reasons. Accelerometers can easily be positioned to capture the motion of the user's arm and hand, as well as meeting the gesture recognition requirement defined above. Furthermore, they are able to provide data at higher frequencies compared to visual approaches incorporating regular cameras thus ensuring not to discard higher frequency parts of the user's motion. At the same time they meet the responsiveness requirement as identified above. With a closer look at the usability requirement defined above it becomes apparent that

Non-Functional Requirement	Description
Availability	The system and its components must be able to provide the gesture recognition service to its host system (Pinocchio) as long as this runs.
Usability	The input device to be used by the system must not limit the user's execution of motions while conducting, nor must it restrict the user's operating range, that is the user must be able to chose his or her position to conduct freely while in the same room with the computer running Pinocchio, regardless of his or her orientation towards this computer.
Reliability	The processing chain consisting of the input device, the recognition module, and the connection to Pinocchio as well as the interaction between these components need to work properly and create reproducible results.
Robustness	The system must be able to differentiate between meaningful gestures, that is gestures relevant for conducting, and arbitrary movements.
Responsiveness	Conducting is a time-critical process (compare [Schmidt et al., 2007, Lee et al., 2005]). Therefore, the system must have a response time between zero and twenty milliseconds in order to provide sufficiently fast feedback to the user.
Interface	The system has to be seamlessly integrated into the existing Pinocchio virtual orchestra system. It must be accessible through Pinocchio's graphical user interface and it must be able to control tempo and volume of the piece of music being played.

Table 4.2: Non-functional requirements of the gesture recognition subsystem

accelerometers are an excellent choice to build an input device which does not rely on the line of sight and—because of the relatively small amount of data being generated—is able to transfer its sensor data over the air.

Type of Recognition Algorithm For choosing a recognition algorithm, all of the functional requirements defined above have to be considered. I chose to use a machine learning approach because such approaches have been successfully applied in similar systems before and because they are able to handle the recognition of sequences in related areas dealing with time series data (such as speech or handwriting recognition) at the same time. Machine learning approaches seem to be more suitable to recognize gestures coming from different users. Additionally, since they are being trained, it seems fairly simple to teach new gestures.

4.3 Hardware Decisions

This section illustrates the decision process for the hardware to be used in this project, guided by the requirements elicitation and recognition method decision discussed above.

Computer Platform The choice of the hardware platform which is to run the actual virtual orchestra system is predetermined by the interface requirement listed in table 4.2: Pinocchio has been implemented for the Apple Macintosh computer platform and is still under constant development to improve its capabilities. Since this thesis presents an extension to Pinocchio enhancing its input capabilities, I shall use the same platform for the final implementation of the gesture input subsystem. For a discussion of resulting limitations which result from this choice, refer to paragraph 4.4.

Input Device This thesis' goal is to explore new methods for gesture recognition, applying machine learning techniques to acceleration data. Conducting gestures for controlling a virtual orchestra are used as an example application area. Due to the nature of conducting gestures, the accelerometers to be used have to be positioned as close as possible to the hand executing the gestures (as mentioned under the functional requirement of gesture identification).

Elaborating further on the previously non-functional requirement of usability, the following additional requirements regarding the acceleration sensors can be identified. It is necessary that they are

- comfortable to wear (they must be light enough not to interfere with the gesture execution),
- that they are easy to put on and easy to remove (the user must be able to accomplish these tasks without help in a reasonable amount of time, not taking longer than putting on a watch for example),
- and that they can easily communicate with the host application (not limiting the user's range of action as they would with a wired connection for example).

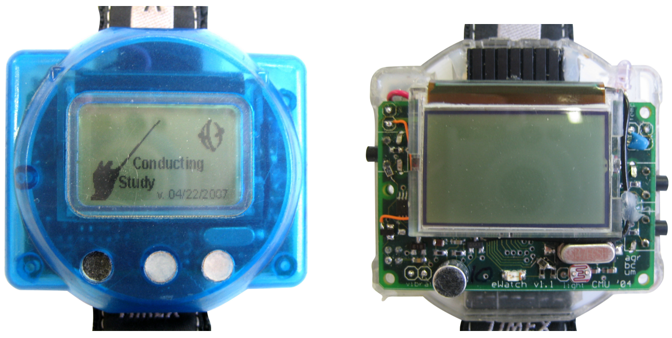
In late 2006, Nintendo introduced its new gaming console Wii featuring the Wiimote, an advanced input device housing accelerometers among others. It communicates with its host system using the Bluetooth standard, making it possible to be used with different systems as well[7]. [Ailive, 2006] offers a commercial software package that allows to train and recognize gaming gestures, using a machine learning approach; further details are not given. The Wiimote's housing resembles a regular TV or stereo remote control. This design might be suitable for a variety of games, but it is not appropriate for a conducting gesture input device. A conductor often uses a baton [Rudolf, 1950]—which is not possible if the input device itself has to be held at the same time. Even if the conductor is not using a baton, holding a remote control would severely limit the movements of the hand.

The eWatch (see figure 4.2) is a "wearable sensing, notification, and computing platform built into a wrist watch form factor" that houses several sensors [Smailagic et al., 2005, Maurer et al., 2006]. Besides accelerometers it features built-in light, audio, and temperature sensors. It incorporates a software controlled CPU and internal flash memory for persistent data storage. Without the battery the eWatch weighs 85 g. Additionally, it is capable of visual, audio, and tactile feedback and provides a Bluetooth communication module. Unfortunately, the current version of the eWatch provides acceleration data in two dimensions only. It is planned to provide 3D acceleration data in upcoming revisions.

After a close look at the requirements defined above the eWatch turns out to be well suited. It is worn like a normal wrist watch and just as easy to put on. In addition, it is fairly lightweight and can provide wireless communication as well.

The relevant difference for this conducting gesture recognition application between the eWatch and the Wiimote is clearly the form factor. Wearing a watch is much more unobtrusive than holding

[7]Kimura Hiroaki developed DarwiinRemote which is a framework that allows to read the Wiimote's sensors under Mac OS X [Darwiin, 2007]. This framework has been used to conduct experiments integrating the Wii as an input device for Pinocchio.



(a) with housing (b) without housing

Figure 4.2: Top view of eWatch

a remote control and therefore better suited to capture motion while conducting. Additionally, the eWatch is more powerful since it offers on-board computing power which can potentially be used to recognize gestures online on the eWatch itself rather than transmitting raw acceleration data and performing the recognition on the host system. The eWatch is currently being used to automatically classify locations frequently visited by the user. Using it for gesture recognition is a new application area. Besides the advantages mentioned above, the eWatch was readily available for this project and knowledge transfer was easy to realize on site.

On the downside, this specific placement of the eWatch on the user's wrist also posts several limitations to the kind of motion that is detectable. Clearly, motions of the fingers (which are also part of a complex conducting gesture) can not be detected using this setup. Furthermore, acceleration data coming from a motion of the hand alone might contain less information than a motion that incorporates the whole arm and thus causes a higher acceleration of the wrist. As already mentioned above, another limitation of the current version of the eWatch is the restriction to 2D accelerometers.

On the one hand, these accelerometers are mounted in a way that enables them to capture accelerations in the plane of the eWatch. On the other hand, the majority of the information conveyed in conducting gestures is traditionally contained in the plane parallel to the user's face [Rudolf, 1950]. In consequence, if the user wears the eWatch as it is supposed to be worn (i.e. like a regular watch on the wrist), the plane of conducting gestures in front of the user unfortunately is not completely covered; the eWatch captures movements to the right or to the left (which is desired) as well movements toward and away from the user's body (rather than up and down movements).

A decision on how to deal with this shortcoming requires the identification of potential solutions first. I identified the following three potential solutions:

1. Using a single eWatch worn like a regular wrist watch. This setup limits motion capturing to right/left and forward/backward movements.

2. Using two eWatches worn as depicted in figure 4.3 (a). This setup not only allows capturing movements in the conducting gesture plane but also enables the use of three-dimensional data.

3. Using a single eWatch worn perpendicularly as depicted in figure 4.3 (b). The second eWatch in this picture serves the single purpose of holding the first eWatch in the shown position; it does not capture acceleration data. This setup captures movements in the conducting gestures' plane.

Unfortunately, solution one and two were not feasible because for the following reasons. When the eWatch was worn like a regular wrist watch as described in solution 1, the resulting data did

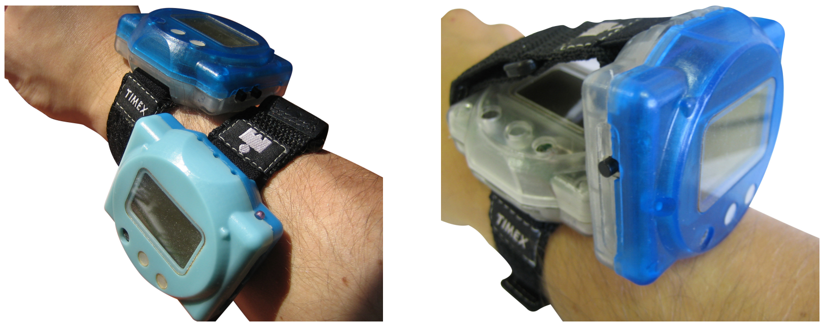


Figure 4.3: Wearing two eWatches

not contain enough information to discriminate between different conducting gestures because of the reasons explained above.

Using two eWatches as described in solution 2 seemed to work very well at first glance. Several test runs I conducted with data collected using this setup showed quite promising results regarding detection ratio and reliability of gesture recognition. Nevertheless, when going from offline testing to online testing, it became apparent that it is not an easy matter to synchronize the two eWatches in realtime. Since the recognition process relies on the accurate alignment of the acceleration data's different dimensions synchronization is crucial. The eWatches do not send information that could be used for synchronization, such as time stamps. Considering the time that would have been necessary to first change the eWatches firmware and second come up with a reliable synchronization approach, I opted for solution three. A disadvantage of this solution is that because to the looseness of the mounting, a certain amount of noise is introduced, depending on how firmly the eWatch is strapped to the user's wrist.

Table 4.3 summarizes the hardware choices discussed in this section.

System Part	**Hardware Component**
Computer platform for Pinocchio	Apple Macintosh
Input device	eWatch

Table 4.3: Summary of hardware components to be used

4.4 Software Decisions

The last decisions to be made are the ones regarding the software to be used in the system. In the remainder of this section, I shall introduce these software decisions.

Target Application As mentioned above (compare paragraph 4.3), Pinocchio is an already existent system. It has been implemented using the Objective-C programming language designed for the Mac OS X operating system thus determining an interface requirement (compare table 4.2). Therefore, I shall also use Objective-C in order to realize the integration of the gesture recognition subsystem. This choice limits the usage of Pinocchio to the Apple Macintosh computer platform.

Research Environment Nevertheless, keeping in mind the research character of this thesis (with its uncertainties at the beginning and in the course of the project) it did not seem feasible to start the implementation in the above mentioned programming language right away. Combining the eWatch (being a research device and not a commercial product which means a less stable behavior) with a machine learning recognition approach (compare section 5.1) that has rarely been used (if at all) for this specific combination of input device, sensor types, and target application requires a considerable amount of trial and error evaluating different approaches.

At the beginning of this project it was not yet clear which of the numerous, often vague ideas of what would be feasible would work out in the end. Additionally, many new ideas were generated during this iterative process. A more flexible and powerful solution than what the programming language mentioned above offers became necessary. Therefore, I chose MATLAB in order to implement, evaluate, and compare different types and iterations of algorithms. MATLAB offers flexible and powerful tools to accomplish these tasks. It is relatively easy to create prototypes of algorithms and to deal with test data. Furthermore, MATLAB offers excellent plotting capabilities in order to visualize recognition and evaluation results.

Application	**Software Tool**
Input device communication	Java
Data processing and gesture recognition	MATLAB (generates C libraries)
System integration	Objective-C

Table 4.4: Summary of software tools to be used

MATLAB itself runs on several computer platforms, such as Microsoft Windows and Mac OS X, among others [MatlabRequirements, 2007]. It is capable of generating C libraries from MATLAB functions for the platform it is running on. Additionally, the integration of Java classes is fairly straightforward and well documented. Since Objective-C's JavaBridge [Apple, 2007] also provides a seamless integration of Java, I chose to implement the software component responsible for the Bluetooth communication with the eWatch in Java. This approach offers the advantage that the communication part of the code only has to be developed and tested once, and can subsequently be used in the research environment with MATLAB as well as in Pinocchio itself. A summary of software tools to be applied can be found in table 4.4.

In this chapter, I discussed the necessary steps of the decisions to be taken, including defining requirements, the choice of recognition method, and decisions regarding hardware and software to be used. In the next chapter I shall explain the design and implementation of the actual gesture recognition method.

5 Conducting Gesture Recognition

After deciding on using a machine learning approach to recognize conducting gestures in the previous chapter, I shall describe the actual conducting gesture recognition method in this chapter. In section 5.1, I start with a discussion of the reasons for choosing hidden Markov models (HMM). In sections 5.2 through 5.3, I explain my approach to classify conducting gestures and the realization of a continuous gesture recognizer. I shall finish this chapter with a description of how gestures are eventually interpreted in order to derive gather information, such as tempo or volume.

5.1 Choice of Machine Learning Approach

In the following, I shall describe the reasons that led to the choice of HMM as the machine learning approach to be used. First, I introduce artificial neural networks (ANN) and discuss their potential application to gesture recognition. Second, I explain the concept of HMM and my decision to use them for recognizing conducting gestures.

5.1.1 Artificial Neural Networks

Artificial neural networks have been used in a few of the conducting systems presented in chapter 2. These systems either used a visual approach and a data glove [Lee et al., 1992b, Takala, 1997] or magnetic motion trackers with six degrees of freedom [Ilmonen and Takala, 1999, Garnett et al., 2001]. Accelerometers have not yet been successfully used in conjunction with ANN for the recognition of conducting gestures. Nevertheless, I chose ANN as a first machine learning approach to evaluate because I already had prior experience in this field. Additionally, I conducted several preliminary gesture recognition experiments using ANN together with the eWatch during my first contact with the Pinocchio project.

The initial user study I conducted when I began working on this thesis yielded about one and a half hours of conducting gesture data (compare section 3.2). Using parts of these data, together with some additional gesture data I collected myself, I started to evaluate the usage of ANN to recognize gestures with Netlab, a MATLAB toolbox “designed to provide the central tools necessary for the simulation of theoretically well founded neural network algorithms” [Netlab, 2007].

Different topologies may be used for ANN: While feed-forward ANN are based on an acyclic graph—thus allowing to travel the signal one way only—recurrent ANN contain feedback loops (compare [Mitchell, 1997]). In the following, I shall evaluate the usage of feed-forward ANN. Recurrent ANN may be considered in future evaluations.

I conducted numerous experiments with different parameters and model settings, such as varying numbers of hidden neurons, different preprocessing methods for acceleration data, and several thresholds designed to prevent multiple recognition of the same gesture. However, I was not able to get sufficiently satisfying results to justify a further investigation in this direction. The recognition rate was less than 50% for any of the tested settings. Especially for the recognition of gestures in a continuous data stream, i.e. the gesture spotting, ANN seem to be unsuitable since they are not able to automatically detect gesture boundaries [Yang and Xu, 1994]. I tested a sliding window approach, which feeds input data covered by the window to the artificial neural network and then moves the windows forward. This approach was not able to reliably identify gestures. [Lee and Kim, 1999] indicates that ANN are more effective for the recognition of static than dynamic patterns.

5.1.2 Hidden Markov Models

Moving forward, I evaluated the second machine learning approach that has been used in prior conducting systems, namely HMM [Usa and Mochida, 1998b, Kolesnik, 2004].

[Rabiner and Juang, 1986, page 5] defines an HMM as a "doubly stochastical process with an underlying stochastic process that is not observable (it is hidden), but can only be observed through another set of stochastic processes that produce the sequence of observed symbols." HMM are able to characterize sequences of observations produced by real-world processes and to represent a signal model—they thus make it possible to build prediction or recognition systems.

Application Areas HMM have been widely used for speech recognition [Huang et al., 1990]. According to [Bishop, 2006, page 610], other application areas for HMM can be found in bioinformatics, shape classification, handwriting recognition, and gesture recognition. Section 2.1 presents several example applications of HMM.

Elements An HMM consists of several elements which I shall introduce in the following: There are a finite number of hidden states N within which "the signal possesses some measurable, distinctive properties" [Rabiner and Juang, 1986, page 7]. These states are connected by edges representing transitions. A transition probability is assigned to each of these edges (including self-transitions), or, put differently, each state has a transition probability distribution. Therefore, the transition probability between states depends on the current state only (commonly referred to as Markov property). The states and transitions trained for recognizing a pattern represent its subpatterns and their sequential order [Lee and Kim, 1999]. When an HMM was trained to recognize a gesture for example, states and transitions represent subgestures.

Each state also has an observation probability distribution which determines the distribution of observation symbols. In addition, an initial state distribution determines the state where the process starts in. An HMM λ is thus determined by its initial, state transition, and observation symbol probability distribution; $\lambda = (A, B, \pi)$. Figure 5.1 shows an HMM with states $Q = \{q_1, q_2, q_3\}$, transition probabilities a_{ij} and observation symbol probability distributions b_j which in turn determine output symbols v_k. Table 5.1 provides a formal model notation for a discrete observation HMM after [Rabiner and Juang, 1986], summarizing an HMM's elements as discussed above.

The HMM discussed so far produce discrete output symbols. There are also HMM producing observations of continuous vectors; the observation symbol probability distribution can be modeled by Gaussian M-component mixture densities, for example.

T	$=$	length of observation sequence
N	$=$	number of states
M	$=$	number of observation symbols
Q	$=$	$\{q_1, q_2, \dots, q_N\}$, states
V	$=$	$\{v_1, v_2, \dots, v_N\}$, discrete set of possible symbol observations
A	$=$	$\{a_{ij}\}, a_{ij} = \Pr(q_j \text{ at } t+1 \mid q_i \text{ at } t)$, state transition probability distribution
B	$=$	$\{b_j(k)\}, b_j(k) = \Pr(v_k \text{ at } t \mid q_j \text{ at } t)$, observation symbol probability distribution in state j
π	$=$	$\{\pi_i\}, \pi_i = \Pr(q_i \text{ at } t = 1)$, initial state distribution

Table 5.1: Formal model notation for a discrete observation HMM [Rabiner and Juang, 1986]

Topologies HMM can have different types of topologies, depending on how transition probabilities a_{ij} are chosen. Figure 5.1 shows an ergodic HMM, i.e., any state can be reached from any other state; all state transition probabilities a_{ij} are larger than zero. It is possible to restrict state transitions by setting selected transitions probabilities to zero.

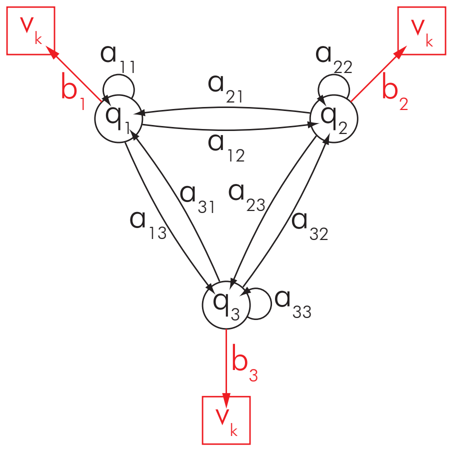


Figure 5.1: Ergodic hidden Markov model with three states

In order to realize a left-to-right, or forward chaining, HMM, which only allows state transitions from lower to higher numbered states (i.e. $j \geq i$ for a state transition from q_i to q_j), all transition probabilities a_{ij} with $i > j$ have to be set to zero. This results in an upper triangular state transition matrix. Furthermore, it is possible to limit the maximum number of states that can be skipped by setting further state transition probabilities to zero. The initial state distribution for left-to-right HMM is chosen so that $\pi_1 = 1$ and $\pi_i = 0, 1 < i \leq N$. Left-to-right HMM impose a temporal order; observations in lower numbered states occur before observations in higher number states.

Dynamic Behavior In the following, the dynamic behavior of an HMM creating observation sequences is illustrated: After determining the initial state, at each clock time t, an observation symbol is produced according to the observation probability distribution of the current state. Then, the transition probability distribution of the current state is used to determine the next state which in turn determines the next observation probability distribution, and so on.

Basic Problems To apply HMM to an actual recognition task, three basic problems have to be solved according to [Rabiner and Juang, 1986][8]. First, the HMM needs to be trained. That is, for a given observation sequence O, I want to adjust the model parameters $\lambda = (A, B, \pi)$ to maximize $\Pr(O|\lambda)$. Second, given an observation sequence O and the model $\lambda = (A, B, \pi)$, I want to compute probability $\Pr(O|\lambda)$ of this observation sequence. Third, given an observation sequence O, I want to compute the most likely state sequence that produces this sequence, given a model $\lambda = (A, B, \pi)$.

There exist well-established algorithms to solve each of the three problems. The Baum-Welch algorithm solves the problem of determining model parameters $\lambda = (A, B, \pi)$ while the Forward-Backward algorithm computes the probability of an observation sequence for a given model. In order to determine a model's most likely state sequence for an observation sequence, the Viterbi algorithm is applied. Although it would be beyond the scope of this thesis to describe all of these algorithms in detail—a comprehensive description can be found in [Rabiner and Juang, 1986, Rabiner, 1989]; I shall furthermore discuss and propose a modification to the Viterbi algorithm realizing a continuous recognizer in section 5.3.

[8]I shall list the three HMM problems in a different order than the one introduced by [Rabiner and Juang, 1986] for a better illustration of the application to the gesture recognition discussed in this thesis.

Realization Having the HMM toolbox for MATLAB [HMMToolbox, 2007] at hand, I conducted some initial evaluations which are described in section 6.1. Since the obtained results were quite promising, I chose HMM as machine learning approach to be used in this project.

5.2 Gesture Classification

The goal of the conducting gesture recognition subsystem to be developed in the context of this thesis is to identify conducting gestures in a continuous data stream. In order to identify a gesture, its boundaries within the input data have to be extracted first, i.e., the input data stream has to be segmented. This step is also called gesture spotting [Morguet and Lang, 1998]. After it has become apparent where a gesture starts and where it ends, the next task is to classify this gesture to determine its type.

HMM are able to combine these two subtasks and solve the problem of gesture identification in an integrated approach; HMM "can automatically absorb a range of model boundary information for continuous gesture recognition" [Yang and Xu, 1994]. Nevertheless, it makes sense first to have a look a the gesture classification as such, since the implementation of a continuous recognizer, that is able to segment and to classify, is more complicated. I chose to explain the classification before the segmentation approach, because classifying gestures, without a segmentation solution in place, works quite well, while the reverse constellation is difficult to realize using HMM.

This section describes how HMM are used to classify isolated gestures. It starts with several preceding considerations regarding the input signal at hand. A description of the HMM parameters I chose for the conducting gesture classification follows. In this context, I shall explain necessary steps to preprocess and extract features out of the raw acceleration data, so that the data can be used with the HMM. This section concludes with a description of the approach I took to reject non-meaningful gestures, i.e. random or non-gesture movements of the hand.

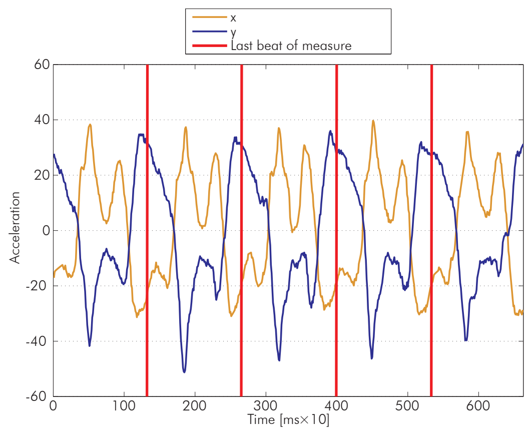
5.2.1 Preceding Considerations

As mentioned in the introduction, I shall describe the necessary steps to realize conducting gesture classification using HMM. This means that the data to classify gestures in, is not a continuous stream of acceleration data, but consists of already segmented pieces of acceleration data. The question is whether there is a gesture contained in a segment and if so, what gesture it is.

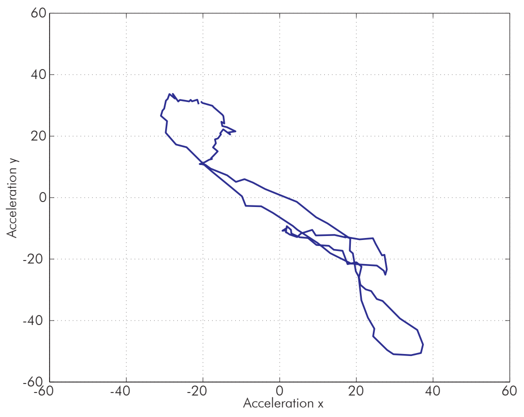
After deciding for HMM, there remain several decisions regarding the model itself left. Since it is often not obvious which parameter settings make sense for the application discussed in this thesis, I relied on both, related work and experiments (compare section 6.1), to guide my decisions.

Before these decisions can be made, it is essential to look at the characteristic of the input signal to model. In this application, the eWatch provides an input signal consisting of acceleration samples in two dimensions, i.e., a feature vector consists of one sample for x and one for y acceleration per time step t. These samples take integer values from -128 through 127, depending on the strength of acceleration. A combined sample consisting of acceleration values for x and y at time t thus represents the acceleration's direction in the plane in front of the user (see 4.3 for a discussion of the eWatch's orientation). The eWatch is able to provide sensor data at controllable sampling intervals up to 100 kHz [Maurer et al., 2006]. The first step is to find a suitable sampling frequency for recognizing conducting gestures.

Sampling Frequency As sampling frequency I chose 100 Hz, which is sufficient to capture motions at a frequency of up to 50 Hz, according to the Nyquist-Shannon sampling theorem. Figures 5.2 through 5.4 contain plots of acceleration data recorded with the eWatch at a sampling frequency of 100 Hz. Plots on the left hand side, labeled with (a), show x and y acceleration series over time for repeated conducting gesture performances, while plots on the right hand side,

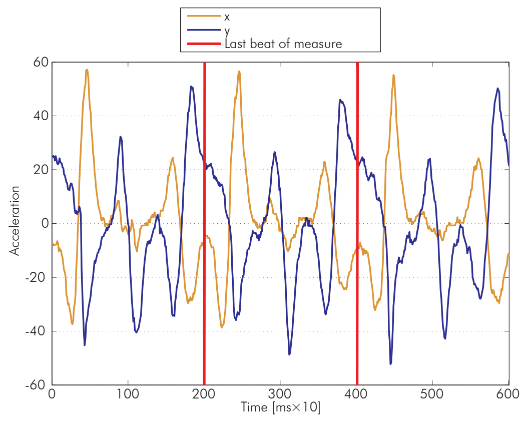


(a) 5 repetitions

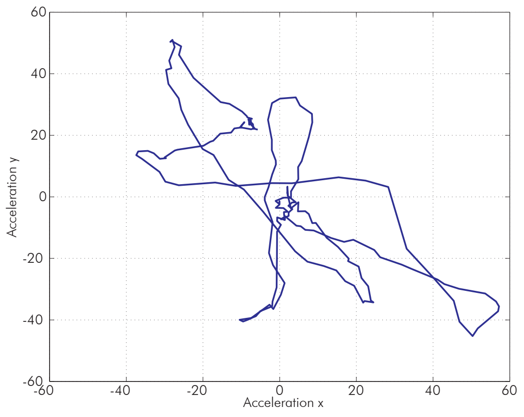


(b) 1 repetition

Figure 5.2: Plot of user A performing 2-beat conducting gestures

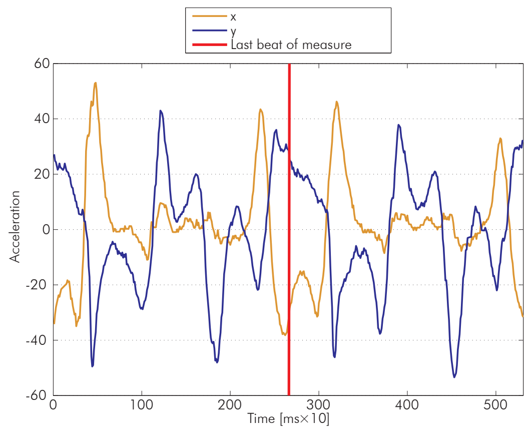


(a) 3 repetitions

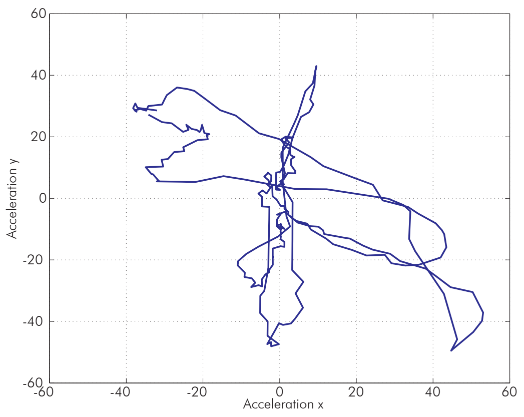


(b) 1 repetition

Figure 5.3: Plot of user A performing 3-beat conducting gestures



(a) 2 repetitions



(b) 1 repetition

Figure 5.4: Plot of user A performing 4-beat conducting gestures

labeled with (b), show x and y acceleration of a single gesture performance. Examining the plots manually, different gestures are clearly distinguishable—suggesting the possibility of a successful automatic gesture detection.

Looking at related literature covering the field of human activity recognition with body-worn accelerometers, I found the following sampling frequencies to be used (for a summary of selected systems refer to table 5.2):

[Lukowicz et al., 2003] used a sampling frequency of approximately 93 Hz to recognize woodshop activities; [Minnen et al., 2005] used a sampling frequency of 100 Hz for the recognition of human activities in context-aware computing. For the recognition of control gestures for video games [Keir et al., 2006] used a sampling frequency of 60 Hz. [Chambers et al., 2002] recognized complex human gestures in sports at a sampling frequency of 96 Hz. [Mäntyjärvi et al., 2004] implemented a DVD player control based on gestures using a sampling frequency of 46 Hz. Although not body-worn, [Kallio et al., 2003] applied accelerometers at a sampling frequency of 50 Hz to the recognition of gesture for interaction with mobile devices (also housing the accelerometers).

System	Application Area	Sensor Type	Frequency [Hz]
[Morita et al., 1991]	Conducting gesture recognition	CCD camera	30
[Bien and Kim, 1992]	Conducting gesture recognition	CCD camera	15
[Bertini and Carosi, 1992]	Conducting gesture recognition	CCD camera	25
[Sawada and Hashimoto, 1997]	Conducting gesture recognition	Accelerometers	25
[Usa and Mochida, 1998a]	Conducting gesture recognition	Accelerometers	100
[Chambers et al., 2002]	Sport gesture recognition	Accelerometers	96
[Lukowicz et al., 2003]	Workshop activity recognition	Accelerometers	93
[Mäntyjärvi et al., 2004]	DVD player control gesture recognition	Accelerometers	46
[Minnen et al., 2005]	Activity recognition	Accelerometers	100
[Keir et al., 2006]	Video game control gesture recognition	Accelerometers	60

Table 5.2: Comparison of sample frequencies for selected human activity recognition systems

Conducting gesture systems using a visual approach, such as the ones described in [Morita et al., 1991, Bertini and Carosi, 1992], are limited to the frame rate that the visual sensor provides. In the case of regular cameras, the frame rate is 25 Hz for the PAL standard and approximately 29.97 Hz for the NTSC standard. [Bien and Kim, 1992] only processes the incoming images at a rate of 15 Hz, though.

For the recognition of conducting gestures with accelerometers, [Sawada and Hashimoto, 1997] used a sampling frequency of only 25 Hz, while [Usa and Mochida, 1998a, page 278/279] state that "the lower limit of the sampling frequency to recognize a conducting gesture is about 100 Hz", and thus uses a sampling frequency of 100 Hz in the according conducting system. This statement, together with the fact that neither of the approaches described above was using a sampling frequency higher than 100 Hz, confirmed my decision to use a sampling frequency of 100 Hz in this project.

5.2.2 Hidden Markov Model Parameters and Feature Extraction

After carefully examining the available input signal, I shall determine suitable HMM parameters and feature extraction methods in the following. To realize isolated, i.e. non-continuous, recognition of gestures (as described in this section) a respective HMM λ_g is built and trained for each gesture that is supposed to be recognized by the system[9]. A segment of input data—the observation

[9]Continuous recognition of gestures requires a more integrated approach and is described in section 5.3.

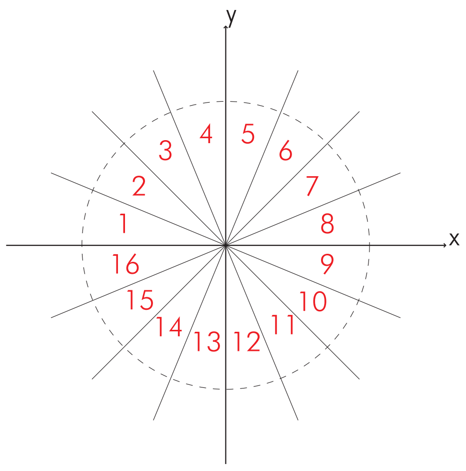


Figure 5.5: Division of directions into sixteen codewords

sequence O^g—is then passed to each of these HMM λ_i. Applying the Forward-Backward algorithm, the probabilities of the observation sequence $\Pr(O^g|\lambda_i)$ are calculated. The HMM yielding the highest probability $\lambda_{\operatorname{argmax}_{1\leq i\leq G}(\Pr(O^g|\lambda_i))}$ (G being the number of trained HMM) is selected, i.e., the corresponding gesture g was recognized.

As can be seen from the description above, this procedure always selects one among the known gestures, even if the input data do not contain such a gesture. The rejection of non-meaningful movements is described in section 5.2.3.

Continuous vs. Discrete HMM The first decision regarding HMM to be made is choosing to use continuous or discrete observation values. HMM can either produce discrete output symbols or continuous vectors (compare section 5.1.2). Both of the prior systems [Usa and Mochida, 1998b, Kolesnik, 2004] which applied HMM to conducting gesture recognition successfully used discrete output symbols.

Also considering the character of input data provided by the eWatch—namely the acceleration's direction at each time step, which can easily be discretized—I chose to use an HMM producing discrete output symbols (discrete HMM in the following).

Smoothing A common preprocessing step before quantization is to smooth the input data in order to eliminate noise which potentially decreases the recognition performance. By applying smoothing, high frequencies in the input signal are discarded, while low frequencies, which are presumably more important with regard to gesture recognition, are preserved—hence realizing a low-pass filter.

Such a low-pass filter can easily be implemented by means of a moving average algorithm. For each input sample, this algorithm calculates a new value by taking the mean of n surrounding samples. There exist different variants, such as a weighted moving average which assigns different weights to surrounding samples, depending on their distance to the current sample. While eliminating noise, a smoothing filter introduces latency. An evaluation of a smoothing filter's effectiveness in this specific application is presented in chapter 7.

Vector Quantization After deciding for a discrete HMM, it becomes necessary to choose a quantization method in order to derive discrete symbols from the continuous acceleration vectors available. A conducting gesture consists of a series of directions, varying over time. Therefore, it makes sense to extract exactly this information and to use it as output symbols for the HMM. In dependence on [Lee and Kim, 1999], who applied HMM to the recognition of hand gestures for browsing PowerPoint using a camera-based approach, I shall use a vector quantization approach that converts continuous, two-dimensional feature vectors of acceleration into a finite number of directions, more generally referred to as labels or codewords. Figure 5.5 depicts the division of directions into sixteen codewords. The number of labels to be used in this application is empirically determined in chapter 7.

Vector quantization assigns a directional codeword to every feature vector in the following way: At time step t, the directional feature vector a_t is determined by using values x_t and y_t of the two-dimensional acceleration data (compare figure 5.6). This approach requires the calibration of input data as discussed in the following section. a_t is then normalized using equation 1 (if not zero).

$$\begin{aligned} a_t &= \begin{pmatrix} x_t \\ y_t \end{pmatrix}, \widehat{a_t} = \begin{pmatrix} \widehat{x_t} \\ \widehat{y_t} \end{pmatrix} \\ \widehat{a_t} &= \frac{a_t}{\|a_t\|}, \|a_t\| > 0 \end{aligned} \tag{1}$$

Equation

$$\cos\alpha = \frac{\widehat{a_t} \cdot \widehat{e_t}}{\|\widehat{a_t}\| \cdot \|\widehat{e_t}\|} = \widehat{a_t} \cdot \widehat{e_t} = \widehat{x_t}, \|a_t\| > 0 \tag{2}$$

shows that $\cos\alpha = \widehat{x_t}$, $\widehat{x_t}$ being the x acceleration value after normalization. To assign a directional codeword it is not necessary to compute α itself. Note that $\widehat{x_t}$ only covers a semicircle (that is segments one through eight in figure 5.5). In order to cover the whole circle, i.e. all possible directions, I also have to take y_t—or $\widehat{y_t}$ since the sign does not change with normalization—into account. In the following equation, n stands for the number of desired codewords (sixteen in figure 5.6); n must be even.

$$cw_t = \begin{cases} \text{round}\left(\widehat{x_t} \cdot \frac{\frac{n}{2}-1}{2} + \frac{\frac{n}{2}-1}{2} + 1\right), & \widehat{y_t} \geq 0 \wedge \|a_t\| > 0 \\ \text{round}\left(\widehat{x_t} \cdot \frac{\frac{n}{2}-1}{2} + \frac{\frac{n}{2}-1}{2} + 1\right) + \frac{n}{2}, & \widehat{y_t} < 0 \wedge \|a_t\| > 0 \\ \text{round}\left(\frac{\frac{n}{2}-1}{2} + 1\right), & \|a_t\| = 0 \end{cases} \tag{3}$$

Multiplying and rounding realizes the assignment to a codeword $cw_1 \ldots cw_{\frac{n}{2}}$ for $y_t \geq 0$ as shown in equation 3. Using the case differentiation, I assign directions $cw_{\frac{n}{2}+1} \ldots cw_n$ to feature vectors that lie in the other semicircle, i.e. $y_r < 0$. If $\|a_t\| = 0$, i.e. $x_t = 0 \wedge y_t = 0$, I assign the directional codeword determined by round $\left(\frac{\frac{n}{2}-1}{2} + 1\right)$. In order to minimize the influence of low-amplitude noise, the vector quantization algorithm allows the specification of a cutoff threshold th_c. If $\|a_t\| > th_c$, a_t is set to $\binom{0}{0}$. An evaluation of this cutoff threshold is presented in section 7.2.3.

Even if there is no motion at all, acceleration values are not always zero due to gravity and depending on the eWatch's orientation. Since the extraction of directional codewords as described here relies on non-shifted acceleration values for x and y, the acceleration data have to be calibrated first, in order to produce meaningful codewords (figure 5.6 shows already calibrated acceleration data). The calibration method is responsible for eliminating the gravity's effect, while maintaining the signal's characteristics at the same time (compare [Keir et al., 2006]). Because of its complexity, I shall describe the calibration method separately in the following section.

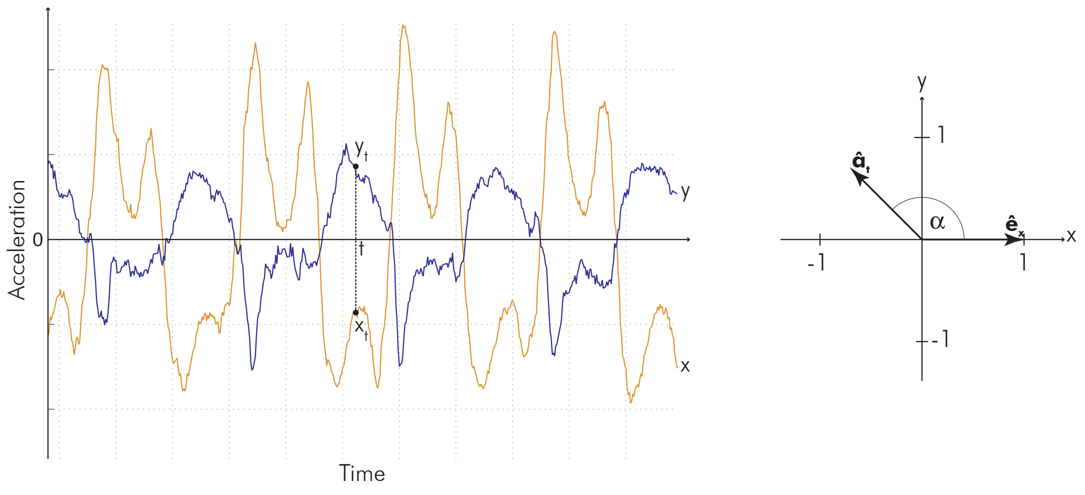


Figure 5.6: Illustration of vector quantization approach

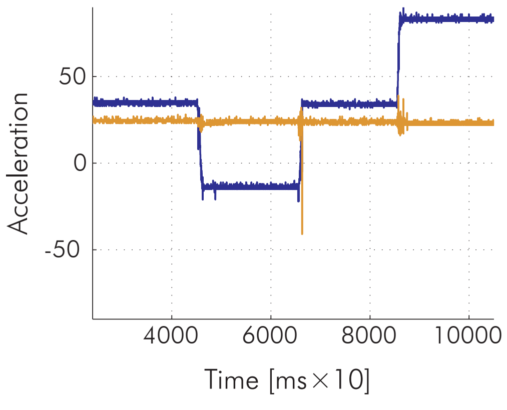
It is also possible to use a vector quantization algorithm that automatically clusters the feature vectors, rather than using fixed classes of directions. Nevertheless, I chose to use the direction extraction method because it fits the nature of hand gestures well and can be implemented efficiently. Besides the described method of label extraction in the signal's time domain, a feature extraction in the frequency domain is possible, too. An analysis in the frequency domain requires a signal transformation, using a sliding window approach. I decided for the vector quantization method in the time domain as described above because, first, one of the requirements defined in section 4.1 is a fast response, and second, a sliding window introduces a certain latency.

Data Calibration Effective calibration is crucial to the vector quantization method introduced above. The algorithm I propose for calibration works as depicted in listing 1. It uses a buffer of size *n* to calculate a running average, which is then subtracted from the current data sample (line 4). Note that the buffer is filled with already calibrated data (line 5). *input_data* is an array of one-dimensional acceleration data, that is, the calibration has to be executed for each dimension separately. The average is not just computed by taking the buffer's mean value but, as shown in line 6, an *adapt_factor*<0 is used to adjust the value slowly.

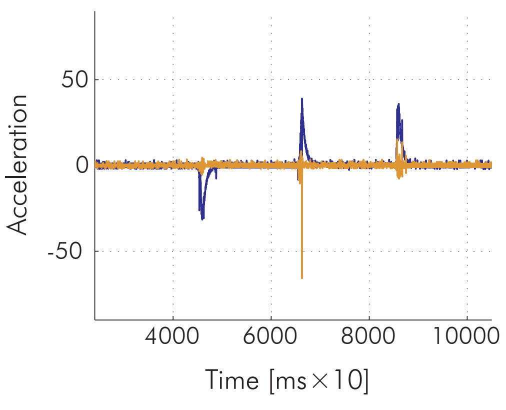
Listing 1: Calibration algorithm

```
1  avg = 0;
2  buffer(1 : n) = 0;
3  for i = 1 : length(input_data)
4      calibrated_data(i) = input_data − avg;
5      store_in_buffer(calibrated_data(i));
6      avg = avg + adapt_factor ∗ mean(buffer);
7  end
```

To show the approach's effectiveness I recorded acceleration data from the eWatch while it was turned from lying with the LCD screen facing up, to the side with one button facing up, to the back side facing up, and then to the side with two buttons facing up. Figure 5.7 shows the

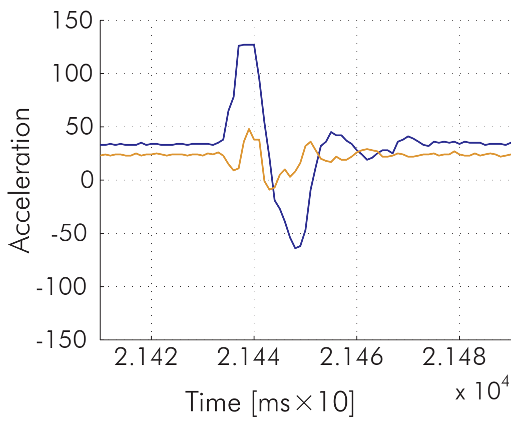


(a) original signal

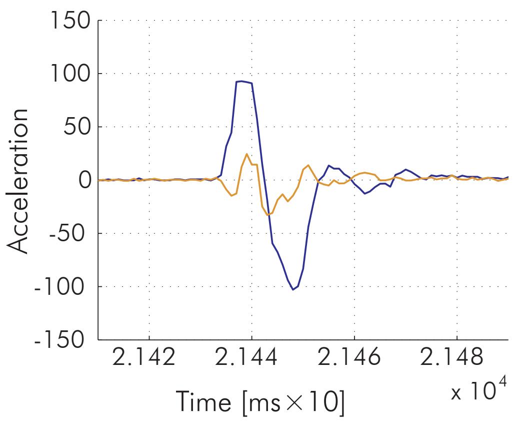


(b) after calibration

Figure 5.7: Normalization effect of calibration algorithm



(a) original signal



(b) after calibration

Figure 5.8: Preserving effect of calibration algorithm

resulting data's plot before (a) and after (b) calibration[10]. As can be seen from the plots, the calibration algorithm efficiently shifts the acceleration data.

Moreover, it also preserves the signal's characteristic. The eWatch, lying on a surface, was quickly moved to the right. Figure 5.8 shows the resulting acceleration data before (a) and after (b) calibration. As can be seen from the plots, the signal's characteristics did not change after calibration; the motion is still clearly recognizable.

HMM Topology Another decision to be made is what topological configuration to apply. That is, one has to decide whether it is more suitable to use an ergodic, i.e. fully connected, or a left-to-right HMM. [Kolesnik, 2004] and [Usa and Mochida, 1998b] used a left-to-right model for conducting gesture recognition. This makes sense for the following reason:

As explained in section 5.1.2, a left-to-right HMM imposes a temporal order. Lower states account for observations that occur before those in higher states. Thus, this HMM topology matches the nature of a (conducting) gesture, which is a temporal sequence of directional movements having a starting and an ending point. Movements at the beginning of the gesture obviously happen before those in the end of the gesture. This temporal characteristic can be represented by a left-to-right HMM in a suitable way.

[10]An *adapt_factor* of 0.03 was used.

Number of HMM States While there exist well-established training algorithms to optimize an HMM's probability distributions (compare section 5.1.2), the number of states is often selected manually [Yamato et al., 1992, Min et al., 1997, Usa and Mochida, 1998b, Lee and Kim, 1999] using a priori knowledge about the complexity of the information to be modeled [Huang et al., 1990, Pylvänäinen, 2005]. In the context of this thesis, this means that, the longer a conducting gesture lasts, the more states are required to adequately represent it using an HMM; "more states can distinguish more different characteristics" [He and Kundu, 1991].

Since it is not apparent what number of states is suitable for recognizing conducting gestures from two-dimensional acceleration data using a discrete HMM, I used cross validation [Wilson and Bobick, 1999, Günter and Bunke, 2003] to empirically determine an optimum number of states. This evaluation procedure is described in detail in section 7.2.1. As assumed, a state setting assigning more states to longer, i.e. more complex, gestures yielded better recognition results than a state setting assigning the same number of states to all gestures.

5.2.3 Non-Meaningful Gesture Rejection

As mentioned in the introduction to section 5.2.2, the approach described so far always selects a gesture, even if the provided input data did not contain one. This behavior is due to selecting the maximum probability of the input sequence being produced by one of the trained HMM.

Garbage Model One approach commonly used to reject non-gesture hand movements is the concept of a garbage or filler model [Lee and Kim, 1999]. An additional HMM is trained which is supposed to recognize non-meaningful gestures, i.e. yield the highest probability of all used HMM if the input data do not contain a gesture. Of course, this garbage HMM also has to be trained. It is not apparent, however, what training data are suitable in the context of this project. I evaluated the usage of both, random movements and noise recorded when the eWatch was held still, as training data (compare [Rigoll et al., 1998]). Neither of the two training data classes were able to successfully reject non-meaningful gestures, though.

Threshold Model A simple solution to the rejection is the introduction of a threshold value: If the maximum probability is smaller than the threshold value, the system would return the result that no gesture was recognized. Unfortunately, such a simple thresholding often does not work (compare [Lee and Kim, 1999]). It is for example not apparent how to select a suitable threshold value.

During the evaluation of the threshold approach, I tested using the smallest probability $\Pr_{min_g} = \min\left(\Pr\left(O_{T_g}|\lambda_g\right)\right)$ that was output by the Forward-Backward algorithm when being applied to the training sequences O_{T_g} used to train model λ_g representing gesture g, as proposed by [Hofmann et al., 1998]. That is, I used the same observation sequences O_{T_g} as input to the Forward-Backward algorithm which I also used above to train the according HMM λ_g (by means of the Baum-Welch algorithm). This approach was not able to successfully reject non-meaningful gestures; it constantly misclassified gestures as garbage and vice versa. Additionally, the usage of an arbitrarily selected "quality factor" to be multiplied with the determined threshold in order to compensate for unsatisfactory performance, as described in [Hofmann et al., 1998], indicates that a simple threshold is not a generally applicable approach.

[Lee and Kim, 1999] propose the usage of a threshold model which computes an adaptive threshold to be used in the decision on whether the input data represent a gesture. The proposed threshold model λ_T, described as "a weak model for all trained gestures", yields a lower probability for a given gesture g than the appropriate HMM λ_g, if the input data contained gesture g. Therefore, it can be used as an adaptive threshold. That is, if an HMM λ_g corresponding to a gesture g yields a lower probability than the threshold model λ_T for a given observation sequence O, O did not contain any meaningful gesture.

The threshold model has the same number of states as all gesture HMM combined; it is assembled by copying the states from the (already trained) gesture HMM λ_i as follows: All states, i.e. their observation symbol probability distributions and their self-transition probabilities, stay the same, but all state transition probability distributions are discarded. Instead, every state gets connected with every other state, using transition probabilities of

$$a_{ij} = \frac{1 - a_{ij}}{N-1}, \text{ for all } j, i \neq j \tag{4}$$

—resulting in an ergodic model. An initial and final state is added which connects to respectively is reachable from all inner states. The integration of the threshold model into a continuous recognizer is described the next section.

5.3 Gesture Spotting

After solving the task of classifying different gestures as described in the previous section, the next step is to implement a recognizer which is able to identify gestures from a continuous data stream. In addition to the classification, such a recognizer needs to segment its input data, i.e. it needs to find the gesture's starting and end point. This procedure is also referred to as gesture spotting (in the style of word spotting in speech recognition, the first application area of HMM).

It is possible to realize the input segmentation as a separate module, i.e. independent from the HMM classifier. [Hofmann et al., 1998] follow such an approach. They separate gestures from each other by means of a preprocessing method which thresholding to the input data's velocity profile. [Kang et al., 2004] follow a similar approach. They identify possible segmentation points which have an "abnormal velocity" or other special characteristics that distinguish them from regular samples. Again, the gesture spotting is realized as a separate module. [Morguet and Lang, 1999] compare the two approaches and report that using a single-stage HMM yields better performance than using a separate segmentation module.

5.3.1 Integrated Gesture Recognizer

Compound Model To realize the recognition of conducting gestures in this thesis, I decided in favor of an integrated approach, similar to the one proposed in [Lee and Kim, 1999]. Figure 5.9 shows a compound gesture recognizer for three different conducting gestures (2-beat, 3-beat, and 4-beat as depicted in figure 3.1 on page 27), including a threshold model as described in section 5.2.3. In this example, the number of states per HMM is eight, twelve, and sixteen, respectively. This constellation thus results in a threshold model consisting of thirty six states. The submodels representing the gestures use a left-to-right topology which allows to skip at most one state.

Additionally, an initial state with transitions to each HMM's first state is introduced and the single model's last states are connected to its initial state. The single models and their probability distributions are not touched any further. I shall discuss the meaning of event and final states as depicted in figure 5.9 in section 5.3.2.

As can be seen from this description, combining the individual gesture models with the threshold model results in a new HMM. That is, it is possible to apply the well-established Viterbi algorithm for finding an optimal state sequence to this new HMM. In the following, I shall illustrate how gestures can be spotted and classified using the proposed model. For this, I modified the Viterbi algorithm in a way that allows handling of continuous input data as well as integrated spotting and classifying of potentially contained gestures.

The Viterbi Algorithm Before describing the modifications, I shall outline the Viterbi algorithm's basic functionality first (for further details compare [Rabiner, 1989]). Table 5.1 on page 40 defines the elements of an HMM used in the following description.

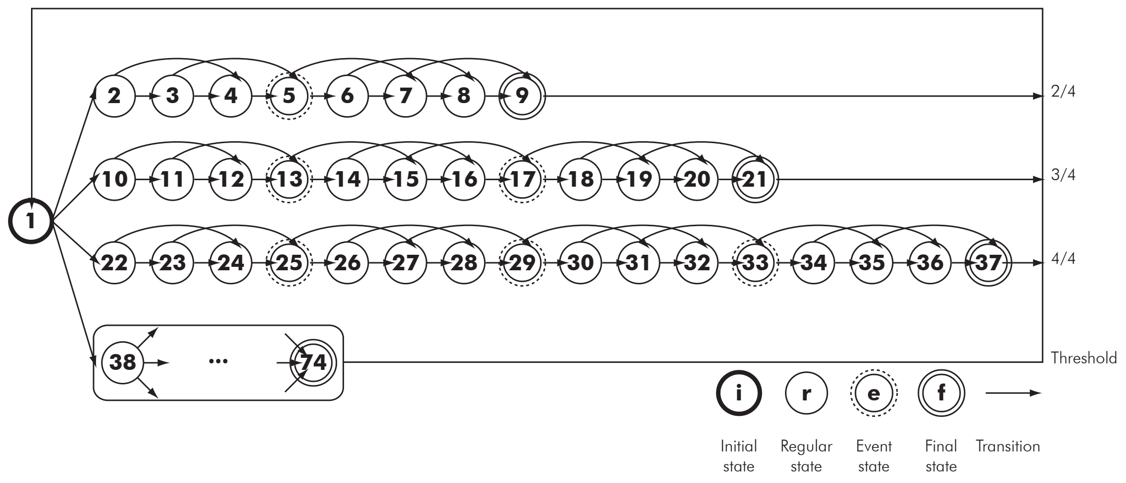


Figure 5.9: Compound HMM for three conducting gestures

The Viterbi algorithm finds the single best state sequence, $Q = \{q_1, q_2 \dots q_T\}$, for a given observation sequence $O = \{O_1, O_2 \dots O_T\}$ and an HMM λ. At each point in time t, the highest probability for each state j is calculated, taking into account the first t observation symbols. I introduce

$$\delta_t(j) = \left[\max_{1\le i\le N} \delta_{t-1}(i)\, a_{ij}\right] \cdot b_j(O_t), \qquad 2 \le t \le T, \quad 1 \le j \le N \tag{5}$$

which thus contains the probability of the most likely single path that leads to state j at time t, given the observations symbols $O_1, O_2 \dots O_t$. In order to retrieve the state sequence that led to this state, I introduce the array

$$\psi_t(j) = \operatorname*{argmax}_{1\le i\le N} \left[\delta_{t-1}(i)\, a_{ij}\right], \qquad 2 \le t \le T, \quad 1 \le j \le N \tag{6}$$

keeps track of the argument that maximized equation 5. Using backtracking, the state sequence can be uncovered. The variables are initialized with

$$\delta_1(j) = \pi_j b_j(O_1), \qquad 1 \le j \le N \tag{7}$$

$$\psi_1(j) = 0. \tag{8}$$

The algorithm terminates as follows:

$$P^* = \max_{1\le i\le N} \left[\delta_T(i)\right] \tag{9}$$

$$q_T^* = \operatorname*{argmax}_{1\le i\le N} \left[\delta_T(i)\right]. \tag{10}$$

P^* determines the probability for the most likely state q_T^*. The Viterbi algorithm produces a Trellis, the nodes are represented by states q_i which are connected by transitions a_{ij}. When implementing the algorithm it is necessary to introduce a scaling technique in order to avoid mathematical underflow (for further details refer to [Rabiner, 1989]).

Modification The algorithm described so far determines the most likely state sequence for a given observation sequence O of known length T. In order to realize a gesture recognizer for continuous input data, as it is the intention of this thesis, the algorithm has to be extended. First, some considerations have to be given the problem of input sequences of arbitrary length—because, as long as the system runs, there are input data coming from the eWatch. Second, the modified algorithm has to output when a gesture was detected, which one was detected, and when it started—events have to be generated. In the following, I shall describe the approach I used to tackle these problems.

In order to identify a gesture within the input data, I shall make use of the property of states and transitions representing subpatterns of a gesture (compare section 5.1.2). Also taking into consideration the left-to-right topology applied here, it is apparent that the end of a gesture is indicated by being in the last state of the according HMM when the Viterbi algorithm is applied. These states are labeled final states in the compound model (see figure 5.9). While constructing the compound model, references to these states are stored along with the model for later usage during recognition.

During the recognition process, equation 5 is executed at each time step. In addition, equation 10 is used to determine the currently most likely state (here, T refers to the last sample of input data so far). If the detected state q_T^* is one of the final states, we know that the currently most likely path of states goes through the according submodel; due to the left-to-right topology there is no other possible path to reach a submodel's final state. That is, it is likely that the input data contained the gesture the corresponding submodel was trained for.

Since the final state represents a part of the gesture as well, i.e. it has an observation probability symbol distribution, it is not appropriate to instantly report on the detection of a gesture once its final state has been reached. I determined the number of time steps to wait and stay in a final state, before reporting on a gesture event, by using the training data which contain beat annotations. By means of applying the recognition algorithm to the training data, it is easily possible to calculate the mean of time steps during which a particular model remained in the final state, before the beat, i.e. the end of the gesture, occurred.

Once a gesture was detected, i.e. the gesture's end point as well as the type of gesture was identified, the gesture's starting point can be determined by using $\psi_t(j)$ to track back the path of states. Finally, values stored so far for $\delta_t(j)$ and $\psi_t(j)$ can be discarded, and the algorithm starts anew. If we ended up in the threshold's final state, we can also discard previously stored values $\delta_t(j)$ and $\psi_t(j)$ and restart the algorithm—without reporting on a gesture event.

Considering the computational costs required during the recognition process, the modified Viterbi algorithm is more efficient compared to the usage of a simple classification approach, i.e. computing the input sequences' probabilities for every single HMM separately and comparing them afterwards (as described in section 5.2.2). Using the modified Viterbi algorithm, equations 5 and 6 have to be evaluated once for every input data sample. A simple classification approach, however, requires a sliding window in order to realize continuous recognition. That is, the recognition algorithm used to calculate the probability of an observation sequence covered by the sliding window has to be applied for every single HMM at each time step, involving as many steps as the input sequence contains each time—resulting in a higher number of required calculations compared to the modified Viterbi algorithm. Regarding memory complexity however, the modified Viterbi algorithm is more demanding than a simple classification approach, since it keeps track of probabilities for each state and time step. Additionally, the number of states being involved at a given time is larger because the modified Viterbi algorithm uses a compound HMM.

An evaluation of the modified Viterbi algorithm using example conducting gesture data can be found in section 7.2.

5.3.2 Single Beat Detection

The approach proposed above is able to recognize conducting gestures as a whole. As figure 3.1 depicts, a conducting gesture contains several beats. The number of beats depends on the type of measure which is represented by the according gesture.

If single beats contained in a conducting gestures can be detected, a closer coupling of user input and music playback would be possible, because more points of reference in time would exist. Additionally, it would be possible not only to control tempo and volume, but to synchronize measures in conducting and playback.

Separate Training I tested two approaches to realize this feature. First, I trained a separate HMM for each beat of a gesture. Taking the example of three conducting gestures (2-beat, 3-beat, and 4-beat), nine models were trained. After training, I connected the models representing the same measure, resulting in three HMM, very similar to those that the training with whole gestures yielded. However, states that correspond to single beats are known (namely the states where a connection was inserted). Applying this knowledge, the modified Viterbi algorithm is able to report on single beat occurrences. Yet, it is not apparent how to chose transition probabilities for the connection of single beat models, or if it makes sense to possibly insert more connection between states[11]. In addition, one has to decide in advance how many states are assigned for the representation of a single beat.

Single Beat Derivation Taking these shortcomings into consideration, I followed a different approach: Instead of training models for single beats, I stuck to training models for measures as a whole, as initially described. In order to determine what states correspond to the occurrences of a beat, I applied the same technique already introduced in section 5.3.1 (for determining the number of time steps to wait before reporting on a gesture event). That is, training data which contain beat annotations is used to identify those states that correspond to a single beat. This information is stored together with the average number of time steps the model remains in the states for utilization in the modified Viterbi algorithm.

Modification The Viterbi algorithm was modified to reset after a gesture is detected. This behavior is not suitable after detecting a single beat, though. If the algorithm was reset in such a case, it would not be possible to detect later beats, or the end of a gesture. Therefore, I introduced the principle of event states as shown in figure 5.9. Event states are states that trigger the recognizer to report on a detected event—either a single beat or the end of a gesture. Final states are thus event states, too. However, only final states account for the algorithm to reset.

The state diagram depicted in figure 11.1 on page 89 illustrates the operating principle of the modified Viterbi algorithm. *dur* refers to the number of time steps an event state has been visited, i.e. the duration this state was the most likely state for. The number of time steps to remain in an event state before reporting is referred to by *stateOffset*. *minDur* refers to the minimum number of time steps that are required to trigger an event in case an event state is left, i.e. a different state became more likely, before *stateOffset* is reached. It is also determined by analyzing training data.

Since Pinocchio, in the version available when this thesis was written, does not support the processing and synchronization of beat information, the detection of single beats is not integrated in the implementation described in chapter 6. It went beyond the scope of this thesis to integrate such a mechanism into Pinocchio.

[11]For example, it might be beneficial to connect the last state of the model representing the first beat of a 2-beat gesture with the first states of the models representing the second beat of a 3-beat and 4-beat gesture. Since the first beat is very similar for all three types of conducting gestures, this modification might increase the recognition performance.

5.4 Gesture Interpretation

The method described above is able to find and classify conducting gestures. The next step is to interpret recognized gestures. That is, tempo and volume have to be extracted as musical control information to be sent to the host system.

5.4.1 Tempo

The tempo of conducting gestures is determined by their execution speed. Extracting the tempo is a fairly straightforward task to accomplish, having the gesture recognition in place. When a gesture is recognized, its type defining the number of contained beats b, as well as its duration is known (due to the extraction of starting and end points t_s respectively t_e). In addition, the sampling frequency F and thus the length of a time step t is known.

$$tempo_g = \frac{b}{\frac{t_e - t_s}{F}} \cdot 60 \tag{11}$$

calculates the number of beats per minute (bpm) of gesture g. For example, if a 3-beat gesture with a duration of 250 samples was detected at a sampling frequency of 100 $\frac{samples}{s}$, equation 11 would yield a tempo of

$$tempo_{g_{ex}} = \frac{\text{3 beats}}{\frac{\text{250 samples}}{100\ \frac{\text{samples}}{\text{s}}}} \cdot 60 = 72\ \frac{\text{beats}}{\text{min}}.$$

Before the tempo is send to the host system, a simple moving average filter is used to smooth the result and compensate for small spotting errors. [Borchers et al., 2004] use a more complex approach for adjusting the orchestra's tempo: They let the orchestra play faster after the conductor sped up in order to get in phase again. As can be seen from their method, a tighter coupling between the conductor's and the orchestra's tempo is used. Such a synchronization requires the actual orchestra system to be able to receive information about single beats as well as their position within a measure. Pinocchio in its current version does not support this feature (compare section 5.3.2).

5.4.2 Volume

The volume information conveyed in conducting gestures is determined by the gesture size. To extract a gesture's size, i.e. the length of the path that the user's hand covered while executing the gesture (compare [Morita et al., 1989]), acceleration data $a_1, a_2 \cdots a_T$ collected for the duration T of the gesture is used.

Summing up acceleration twice yields the length of the path which is used together with the previously extracted tempo and the number of beats contained in the recognized gesture to approximate the intended volume.

$$volume_g = G \cdot \sum_{i=1}^{T} \left| \sum_{j=1}^{i} ||a_j|| \right| \tag{12}$$

In addition, a conducting gesture specific factor G is applied, since the path lengths is also dependent on a gesture's shape. A simple moving average filter is applied to smooth the extracted volume values.

In this chapter, I motivated my choice for HMM as machine learning approach to be used for the recognition of conducting gestures in this project. After a short introduction to HMM, I described and illustrated how HMM can be applied to the task of conducting gesture classification, including a discussion of input signal characteristics, choice of HMM properties, preprocessing and vector quantization methods, and methods to reject non-meaningful gestures.

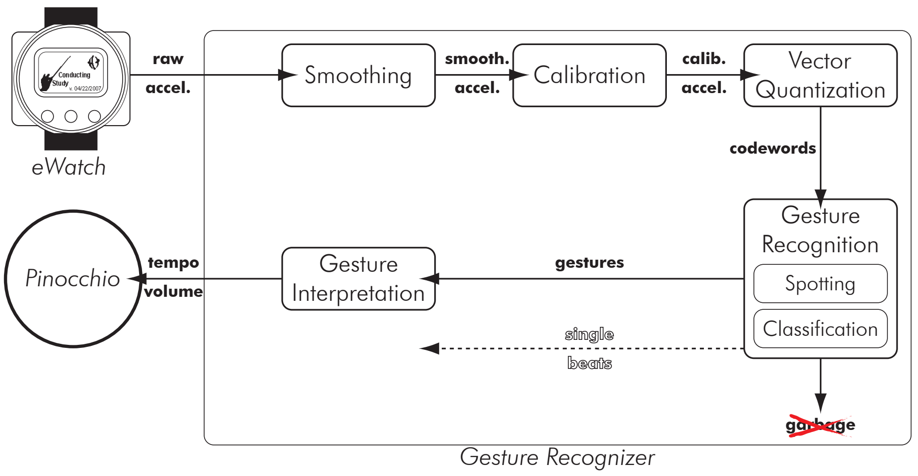


Figure 5.10: Gesture recognizer

Once I explained how gestures can be classified, I discussed the problem of gesture segmentation, alternatively gesture spotting, and how it can be tackled in an integrated recognition approach. This included the modification of the Viterbi algorithm for continuous input data and gesture event generation. Furthermore, I introduced methods to realize a subgesture recognition of single beats.

Finally, I explained the methods applied to interpret gestures, i.e. to extract tempo and volume to be used for musical control. In the following chapter, I shall describe the implementation of the modules introduces so far, summarized in figure 5.10. Chapter 7 will cover an evaluation of the methods proposed so far, utilizing recorded gesture data from different users.

6 Implementation

In the previous chapter, I explained methods and approaches for a conducting gesture recognition module, including data calibration and feature extraction, gesture spotting and classification, as well as gesture interpretation. This chapter describes the implementation of the module and required tools in several steps. I shall introduce a testbed for gesture recognition in section 6.1 which I used to guide my implementation of the recognition algorithms described in section 6.2. After explaining the eWatch communication module in section 6.3, I shall describe tools for data collection and evaluation in sections 6.4 and 6.5. This chapter concludes with integrating the gesture recognizer into Pinocchio.

Reading through previously presented descriptions, it might appear that design choices and parameter decisions were fairly straightforward. In reality, they involved several iteration steps of test implementations and evaluation. This is due to the fact that the gesture recognizer is built from scratch; no prior implementation to build on was available. Furthermore, not only the recognition algorithms have to be implemented, but it is also necessary to develop modules and tools to verify design and parameter choices in a controlled environment, to reliably retrieve data from the eWatch, and to collect, analyze, and annotate large amount of training data for evaluation.

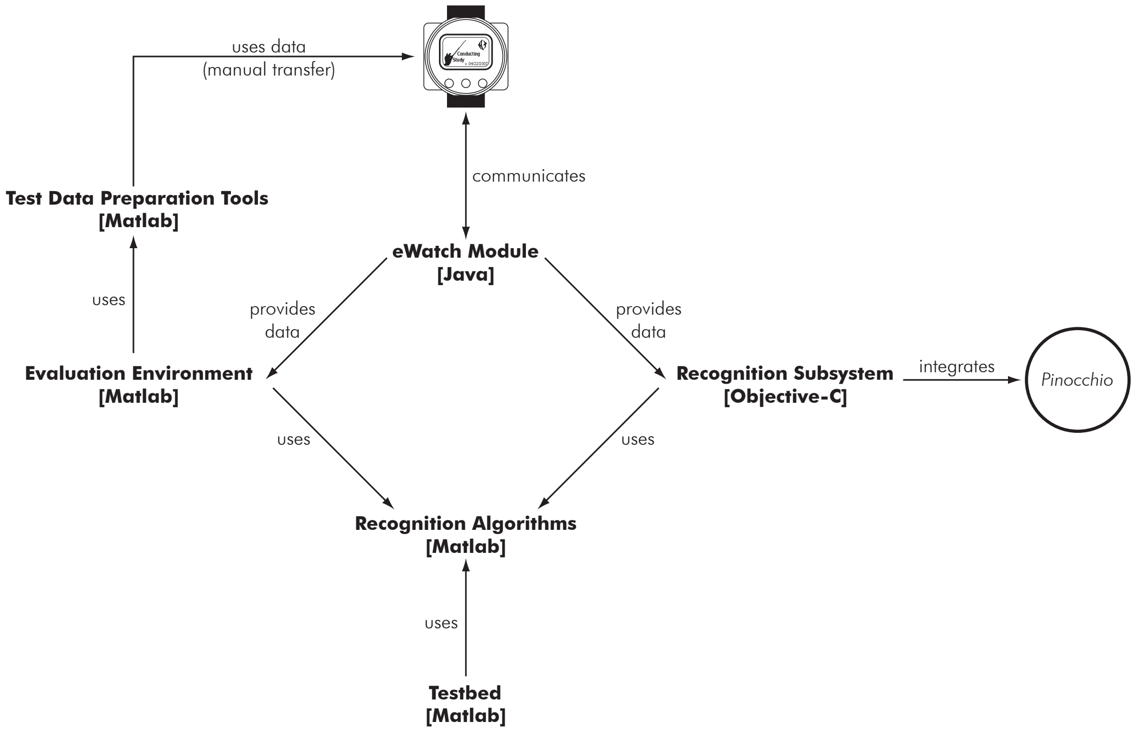


Figure 6.1: Implementation components

Figure 6.1 depicts the components discussed in this chapter, as well as the implementation languages used for realization. The eWatch module (compare section 6.3) provides an interface to communicate with the eWatch device over Bluetooth. Both, the evaluation environment (compare section 6.5) and the recognition subsystem (which integrates gesture recognition into Pinocchio, compare section 6.6), access the eWatch module to retrieve acceleration data. The recognition algorithms (compare section 6.2) are also accessed by both of the modules mentioned above—as well as by the testbed (compare section 6.1), which was used for a first evaluation of the algorithms. In addition to accessing the eWatch module, evaluation data are prepared manually to be used in the evaluation environment, i.e. data are recorded on the eWatch, transferred, and annotated using the test data preparation tools (compare section 6.4).

6.1 Testbed

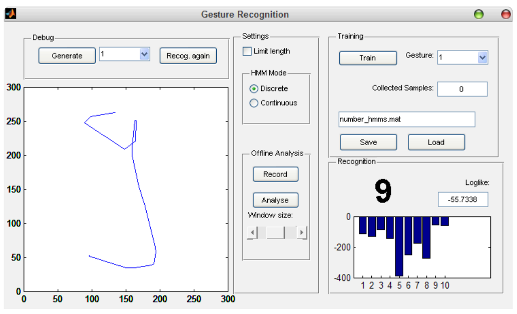
I decided to implement a separate testbed for recognizing gestures using HMM[12] for the following reasons:

- *Elimination of uncertainties.* The tasks of using the eWatch as input device and of preprocessing raw acceleration data already impose several uncertainties which may potentially affect the recognition process. This makes it thus infeasible to initially evaluate the usage of HMM in such an environment, because the increased number of error sources may lead to an unpredictable behavior.

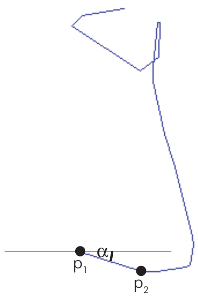
- *Ease of test data generation.* The acquisition and preparation of conducting gesture test data using the eWatch is not an easy matter and consumes a considerable amount of time; refer to section 6.4 for a description of the data recording and analysis process.

I designed the testbed to accept mouse gestures as input and to recognize figures from “1” to “9”. Since mouse input data are easier to acquire and since they also contain less noise than acceleration data coming from the eWatch, uncertainties concerning steps happening before the actual recognition process with HMM can be reduced. Moreover, it is easier to collect sample data as the majority of computers is equipped with a mouse which can easily be accessed.

The implementation is done in MATLAB, allowing me to get familiar with the Hidden Markov Model toolbox [HMMToolbox, 2007] which implements basic HMM algorithms. The testbed is designed to classify isolated gestures. It is not capable of recognizing gestures in a continuous input stream, i.e. it is not capable of performing gesture spotting. To indicate gesture boundaries, the user has to hold down a mouse button while executing a gesture.



(a) graphical user interface



(b) quantization

Figure 6.2: Mouse gesture recognition testbed

To make the testbed more usable for a first evaluation of HMM, I implemented a simple graphical user interface which allows the training and recognition of mouse gestures[13], shown in figure 6.2 (a).

The input data used for training and recognition is quantized in a similar way as described for acceleration data in the previous chapter (compare section 5.2.2). However, instead of extracting

[12]The testbed was inspired in parts by [Starner et al., 1994, Yang and Xu, 1994] who implemented a handwriting recognizer, and by [Kolesnik, 2004] who implemented a symbol recognition as a testbed for a conducting gesture recognition system.

[13]Using the training module, it is of course possible to train other gestures in addition to the figures from “1” to “9” mentioned above

directions of acceleration, positional directions of the mouse path are used. That is, the angle α of a connecting line between two sampling points p_1, p_2 is used to generate codewords (compare figure 6.2 (b)).

In training mode, the user selects the gesture to be trained and then repeats the gesture for an arbitrary number of times. After training all gestures, the resulting HMM can be saved for later usage and the system is switched to recognition mode. When the user draws a figure, the testbed passes the quantized gesture data to each of the trained HMM, which in turn calculate the probability of the input sequence produced by the according model, using the Forward-Backward algorithm (an example of resulting probability values can be seen in the bar chart in figure 6.2 (a)). The resulting probabilities are compared, and the gesture corresponding to the HMM that yielded the largest probability is selected as the gesture that was recognized.

Using this testbed, I also evaluated the implementation of the threshold model (compare section 5.2.3) which showed promising results; the testbed now refused gestures that were not similar to one of the trained figures—while still recognizing them. Before the introduction of the threshold model, the testbed always recognized—or rather selected—one of the trained gestures for sure. The testbed is also able to reliably recognize gestures drawn at different speeds or in different sizes. Furthermore, I tested the user-independence of the HMM gesture recognition approach by recognizing gestures from a different user than the one who provided the training data. Again, the promising results confirmed my choice of model. Several other options, such as the usage of continuous HMM, a window approach for continuous recognition, or the generation of gestures using HMM, were tested in addition, but did not perform well enough to find their way into the actual implementation of the conducting gesture algorithms.

I did not conduct any formal evaluations using this testbed; this was not the intention of implementing it. In fact, I rather see the testbed as a sort of playground that enabled me to get familiar with the properties and peculiarities of gesture recognition with HMM—without having to deal with the difficulties that arise from hard-to-acquire and noisy input data. It was indeed an invaluable tool for collecting experiences regarding this project's realization. In the next section, I shall describe issues related to the actual implementation of the conducting gesture recognizer.

6.2 Recognition Algorithms

As motivated in section 4.4, I chose MATLAB for the implementation of the required algorithms to recognize gestures; MATLAB offers a flexible and powerful environment for the efficient evaluation of different approaches. Moreover, it was easy to deal with gesture test data, i.e. to manage different test sets and according annotations; test data acquisition and annotation is described in detail in section 6.4.

I followed the decisions introduced in sections 5.2 respectively 5.3 in implementing the gesture recognition algorithms. Since the underlying principles already have been described above, I shall focus on implementation specific issues in this section.

The recognition algorithms are split into different modules responsible for data processing and gesture analysis, as depicted in figure 5.10 on page 55. These modules are realized as either a single MATLAB function or, if appropriate due to increased complexity, as a collection of MATLAB functions. The algorithms implemented here are used by the evaluation environment and by the recognition subsystem (and were used by the testbed during the initial stage of the realization process).

Since there is only one single implementation of the recognition algorithms, testing has do be done only once. That is, after I implemented and tested the recognition algorithms using the evaluation environment, I did not have to implement and test them again for the usage in the recognition subsystem, i.e. for the integration into Pinocchio. It was easier to perform testing and verification in the evaluation environment than it would have been in Pinocchio directly. MATLAB, which is used in implementing the evaluation environment as well, also allows the visualization of intermediate steps, which enables potential errors to be quickly identified and eliminated.

Pinocchio is implemented in Objective-C (compare section 4.4). Therefore, the recognition subsystem being responsible for the integration into Pinocchio uses the same implementation language. In order to be able to use the recognition algorithms written in MATLAB from within the recognition subsystem, they have to be converted so that they are accessible from an Objective-C application: MATLAB comes with a compiler [MatlabCompiler, 2007] which generates function libraries in C or C++. If MATLAB is not installed on the target machine where Pinocchio is deployed, the MATLAB Component Runtime (MCR) must be installed in order to be able to use the libraries generated by MATLAB.

After making Objective-C aware of the libraries' location, MATLAB functions can easily be accessed after they have been compiled using the MATLAB compiler. It is possible to pass parameters to and retrieve results from MATLAB functions without any difficulties. I implemented the following modules, sometimes consisting of several algorithms, for gesture recognition in MATLAB: Smoothing, calibration, and vector quantization (compare section 5.2.2), modified Viterbi algorithm for gesture spotting and classification (compare section 5.3.1), and—realized as part of the Viterbi module—gesture interpretation (compare section 5.4).

The first versions of the recognition modules which I implemented were only able to handle offline recognition, because that is the way how I first approached the problem, in order to keep things simple at the beginning. That is, the algorithms expected the whole sequence of input data to be present at the time of execution. Apparently, it is impossible to have the whole input sequence at hand during an online, i.e. realtime, recognition. In fact, input samples are available at a certain frequency one after another only, and an output is expected at each of these steps. Therefore, I had to modify the recognition algorithms to be able to handle realtime recognition.

Doing so is straightforward for algorithms that are not stateful, like the vector quantization; the output of this algorithm is dependent on the current sample only. In contrast, the calibration algorithm, the Viterbi algorithm, and the gesture interpretation algorithms require information to be stored over several time steps. The calibration algorithm for example needs to keep track of a running average. Therefore, I extended all stateful algorithms to return variables, that are supposed to be kept over several time steps, as output on the one hand, and to also accept these state variables as input on the other hand. The host application, i.e. either the evaluation environment or the recognition subsystem in this case, buffers state variables which are returned by functions implementing the algorithms, and passes them as parameters in the subsequent function call. In doing so, the current state of an algorithm is saved by the host application.

This approach allows for a flexible linking of different algorithms at the level of the host application. That is, the algorithms can be executed independently, and their results can be used as input in turn. Therefore, this approach is suitable for the utilization in the recognition subsystem, written in Objective-C. The recognition subsystem's architecture, depicting the integration approach amongst others, is described in section 6.6.

Besides the algorithms applied for recognition, there are also algorithms required for the model's training. Using the Baum-Welch algorithm, an HMM can be trained for a certain gesture, for example (compare section 5.1.2). After having trained every gesture that is supposed to be recognized by the system, a threshold model has to be built, and the single HMM have to be integrated into a compound HMM (compare sections 5.2.3 and 5.3.1). I implemented all required algorithms in MATLAB, respectively used algorithms by Netlab where a suitable implementation was available. Currently, these algorithms are used in the evaluation environment only. That is, they are used to evaluate different types of training configurations. A training module integrated into the recognition subsystem, i.e. a training module which is accessible by the end user in Pinocchio to train new gestures, is an option to increase the system's usability but does not improve the actual task of gesture recognition (compare chapter 9).

As can be seen from this section, all core algorithms, i.e. algorithms that deal with the recognition of gestures and the necessary preprocessing as well as training steps, are implemented in MATLAB. Algorithms that are required during the realtime recognition are made accessible to the recognition subsystem for integration into Pinocchio by means of using C libraries exported from MATLAB.

6.3 eWatch Module

The eWatch module is responsible for communicating with the eWatch. As it is implemented in Java (this decision is motivated in section 4.4), it can be used in both, in the evaluation environment, as well as the integration of the gesture recognition subsystem into Pinocchio.

Since the eWatch is a Bluetooth device providing a serial port, the communication is realized using virtual COM ports[14], which are available as soon as the eWatch has been successfully paired with the computer. Sending data to and retrieving data from the eWatch is then possible by means of standard Java input and output streams. The eWatch accepts simple text commands with parameters. After the sensors are set up with regard to sensor types and sampling frequency to be used, and the data capturing is started, the eWatch continuously sends acceleration data. On the computer side, the eWatch module waits for incoming data and parses them as soon as they arrive. Parsed data are stored in a circular buffer.

The client application—either the evaluation environment or the recognition subsystem in the context of this project—queries the eWatch module in regular intervals; the content of the circular buffer mentioned above is then returned and the buffer is emptied. This approach bears the disadvantage of requiring an additional thread in the client application. However, it decouples the eWatch module from its client, making it possible to be used in different types of applications, which even might be implemented in different languages, as it is the case here. Additionally, the eWatch module can easily be replaced by a communication module supporting different input devices. The potentially resulting decrease in performance when using this approach is tolerable for a prototype implementation which is under steady change, thus requiring a flexible structure. A more efficient way of implementation may be considered for a productive system.

In order to be able to test applications without having the eWatch at hand, I implemented a function that allows the eWatch module to read acceleration data from a file instead from the actual device. The data source, which can be selected at runtime, does not affect the client application; the eWatch module behaves identically, independent from its data source.

6.4 Test Data Preparation Tools

Before it is possible to evaluate any algorithm or parameter setting, test data have to be captured and particularly prepared. After recording gesture data of a user performing conducting gestures, it is necessary to annotate them, i.e. to identify those points of time that correlate to the event of a beat or a gesture. Only after these annotations have been made, it is possible to extract data segments representing single gestures for training. The annotations are also essential in order to evaluate a recognizer's detection ratio and reliability, by comparing detected events to expected (or rather annotated) events.

The more training data available, the more useful an evaluation becomes, because more variations are represented. Therefore, it is desirable to collect as much training data as possible with regards to the time being at disposal. In the remainder of this section, I shall describe several methods for annotating test data, which depend on the circumstances the data were recorded under, and therefore vary in the tools required.

[14]While the port identifiers for serial ports over Bluetooth vary from operating system to operating system ("COM*X*" for Windows, "/dev/tty.*X*" for Mac OS, e.g.), the method of accessing the device stays the same.

Video Annotation The first gesture data that I annotated were recorded for the initial user study described in section 3.2. The intention of this study is to learn more about the interaction between a conductor and his orchestra on the one hand, and to verify that the eWatch is a suitable tool for capturing gesture data on the other hand. During a rehearsal, conductor Daniel Meyer was wearing the eWatch while I recorded him using a Panasonic DVC-30 digital camcorder.

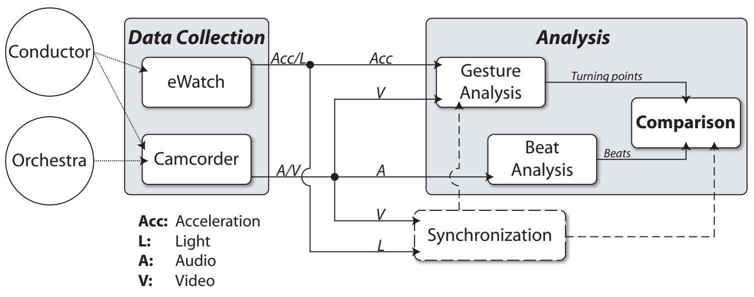
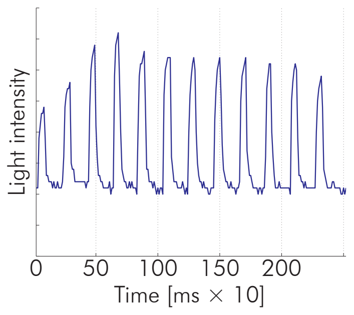


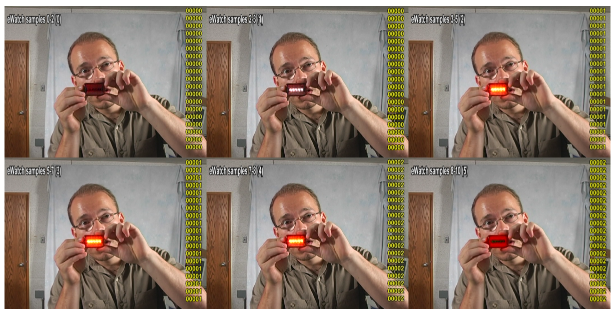
Figure 6.3: Data collection setup

As mentioned in the introduction to this section, the goal of test data preparation is to identify acceleration samples that correspond to conducting beats, since these samples represent gesture boundaries which the system is supposed to identify. Figure 6.3 depicts the recording setup used during data capturing. The principle idea of annotating gesture data with the help of video recordings is to go through the recorded video footage, and find those frames that correspond to a conducting beat. These frames are clearly identifiable by manually expecting image by image. In order to double the resolution in time, I used a simple AviSynth script [AviSynth, 2007] to separate whole frames into fields; one frame consists of two fields as long as the video is recorded in interlaced mode.

Since the eWatch and the digital camcorder are separate devices, means of synchronization have to be found. Otherwise, it would not be possible to correlate video images to acceleration samples. Synchronization serves two purposes: First, it initially aligns video and acceleration data. Second, it accommodates for clock drift. The synchronization approach I chose for this recording is described in the following.



(a) Light sensor data from eWatch



(b) Image sequence from video

Figure 6.4: Synchronization using LED flashes

I am using a LED emitting bright flashes of approximately 50 ms in duration that are clearly recognizable in both, the digital camcorder's video track and the eWatch's light sensor data (compare figure 6.4). LED flashes are recorded at the beginning of the rehearsal and in its end. I am aligning eWatch and camcorder data and match their lengths manually. Since light and acceleration data from the eWatch, and video and audio from the camcorder, are already synchronized respectively, synchronizing light data and video results in a synchronization of acceleration data and video. The actual annotations are entered manually: After identifying a conducting beat, I simply calculated the corresponding eWatch data sample (sample frequencies are known and data sources were aligned before) and stored their indices manually using a MATLAB matrix. In the next section, I shall describe how audio recordings may be used to realize an annotation method that requires fewer manual steps.

Audio Annotation For the second user data acquisition, which is introduced in section 3.2, I asked conductor Walter Morales to perform conducting gestures in a controlled studio environment, following a metronome and selected pieces of classical music. I used the same setup of recording devices as described in the previous section, namely the eWatch and a Panasonic DVC-30 digital camcorder. The LED flash synchronization is used as well. In contrast to the previous data collection, I am using a different approach for data annotation—at least for those gestures that were recorded following the metronome. The camcorder's audio track contains the distinctive sound of the metronome's ticks, which is easy to isolate automatically by analyzing the audio signal's amplitude. The gestures were recorded in a sound studio, thus making the metronome the only significant source of sound.

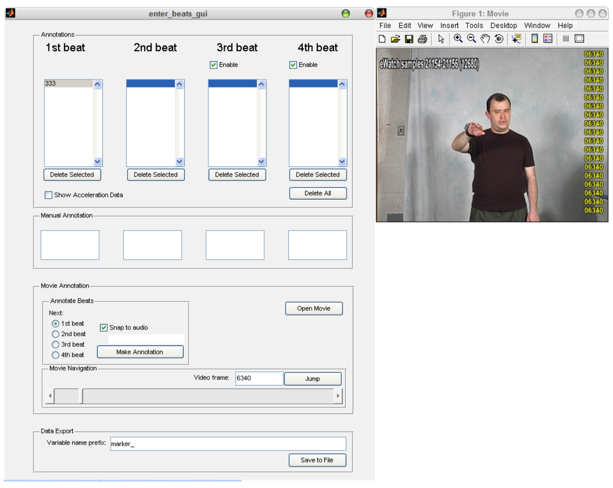


Figure 6.5: Beat annotation tool

In order to be able to annotate gesture data quickly while minimizing errors (which are likely to happen due to the annotation task's monotony), I implemented a tool to make annotating gesture data easier and more efficient in terms of required time. The tool's graphical user interface is shown in figure 6.5. It allows to inspect a video file frame by frame. Once the user identified a gesture beat, he or she tells the system to insert an annotation for this frame, which is then automatically stored.

If the "snap to audio" feature is activated, a window of length n[15] surrounding the currently selected frame in the recording's audio track is scanned for a metronome tick. If the system is able to identify a tick, the annotation is aligned with it. The "snap to audio" feature makes the annotation of data easier, since the user is not required to exactly specify a conducting beat's position. In fact, it is sufficient to roughly point to a conducting beat.

In this section, I described how to extend the previously introduced, video-based annotation method by means of using metronome ticks which are audible in the recording's audio track. This method provides a more efficient and accurate way of annotating conducting gesture data, but it is limited to recordings with an identifiable beat pattern in the according audio track. In the following

[15]The window size n is chosen to be shorter than twice the length of a beat in order to prevent the system from selecting a metronome tick belonging to one of the surrounding beats.

section, I shall introduce a method to collect large numbers of gesture data, without the necessity for manual annotation.

Automated Annotation In order to evaluate the algorithms used for recognizing conducting gestures, I had to collect large numbers of gesture test data, preferably covering different users, measures, tempos, and sizes. Acquiring data using the approaches described so far would have taken too long with respect to time limits that apply for realizing this thesis. Therefore, I chose a different approach for collecting gesture data which I shall describe in the following.

I asked users to perform conducting gestures following a metronome, similar to the user study described in the previous section. However, I only used the eWatch's accelerometers to record data for approximately two minutes per setting; I did not make any audio or video recordings. Knowing how a beat's acceleration data plot looks from previous data collections, I was able to clearly identify the start of the first as well as the last beat performed in a recording session. Since the user was following the metronome with a fixed tempo, it is trivial to insert intermediate beat annotations.

That is, in order to annotate two minutes worth of continuous gesture data, the only steps to be performed are identifying the first and last beat and specifying tempo and measure used during recording (in order to be able to calculate distances between single beats). This approach is apparently limited to the annotation of repeatedly performed gestures of the same type. Moreover, the annotation's accuracy depends on the user's timing while performing the gestures. Nevertheless, taking into account the time saving potential, I chose this automated approach in order to collect gesture data for evaluating the recognition algorithms.

Using this method, I collected seventy-two data sets of two minutes each, or almost two and a half hours of pure gesture data, covering four users, three beat types, five tempos, and three different gesture sizes. Due to time limitations, I did not collect data for every possible combination of the variables mentioned above. In fact, I focused on several settings in order to evaluate different aspects of the recognition system systematically, as described in chapter 7.

In all three of the approaches described in this chapter, I transferred data manually from the eWatch, rather than using the eWatch module described in section 6.3, which is targeted at being used in realtime recognition tasks. Unfortunately, there was no tool available to easily transfer data from the eWatch to an arbitrary computer. Therefore, we implemented an eWatch reader application in Java[16] (using the same communication classes as the eWatch module) to transfer data stored in the eWatch's flash memory to the computer. The reader application is currently being used in several other projects applying the eWatch at Carnegie Mellon University.

6.5 Evaluation Environment

The evaluation environment is also implemented in MATLAB. It uses the recognition algorithms described in section 6.2 and applies them either to live data coming from the eWatch directly, or it uses an offline evaluation with prerecorded data of known content (which have been collected using the approach described in section 6.4) as described in chapter 7. While the recognition of online data is fairly straightforward, the offline evaluation of potentially large data sets, with a multiplicity of different settings to be tested, requires more implementational efforts which I shall describe in the following.

After test data are prepared using the tools described in the previous section, the evaluation environment allows to specify parameter values of the gesture recognizer's calibration, vector quantization, and gesture recognition components to be tested (compare chapter 5 and figure 5.10 on page 55 in particular). It performs a cross-validation on each of the settings specified. In doing

[16]While I was responsible for implementing the communication part of the eWatch reader, Brian French, a graduate student at Carnegie Mellon's electrical and computer engineering department, implemented the data decoding.

so, it assures that data loss due to errors in algorithms to be tested or due to external interruptions, such as power outages, is minimized by saving intermediary results to the hard drive.

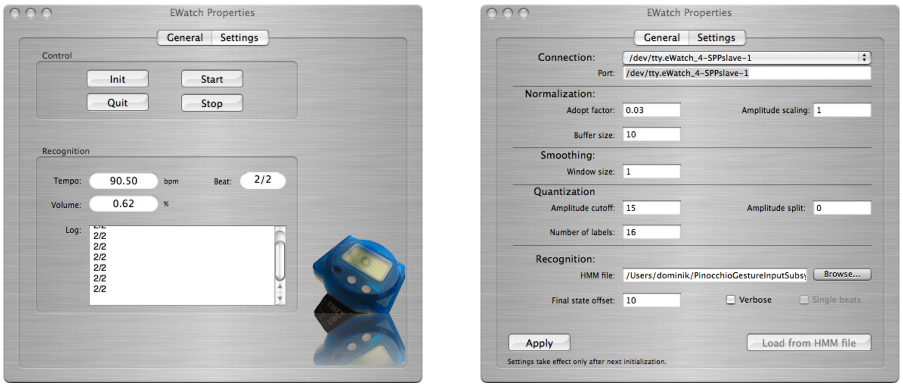
As described in [Bishop, 2006, page 32], an S-fold cross-validation is performed by partitioning the available data into S groups. $S-1$ groups are used to train a model which is then evaluated on the remaining group. This procedure is repeated S times and the performance scores of the single runs are averaged. In the context of this project, I used a two-minute test recording set as one group. Please note that the gesture recognizer requires a test recording for each gesture that is supposed to be recognized.

To better illustrate the procedure of cross-validation, I shall give an example in the following. In order to evaluate the effect of using different number of labels during vector quantization (for a description of the applied vector quantization approach refer to section 5.2.2) on the user-dependent recognition at a certain tempo and gesture size, all available training sets matching the selected user, tempo, and gesture size are selected as input for the cross-validation.

In order to keep the procedure simple, the evaluation environment requires that all gestures to be recognized are represented by the same number of data sets. That is, if there are five recordings of user X performing a 2-beat gesture at 90 bpm, there must also be five recordings each of user X performing a 3- and 4-beat gesture at 90 bpm. In every step of cross-validation for a fixed number of labels (which is the variable to be evaluated in this example), a data set for 2-, 3-, and 4-beat each is chosen and held out; all remaining sets are used for training the recognition model. The three sets previously held out are then used to evaluate the recognition performance. This procedure is repeated five times (in this example) and the final evaluation result is calculated by taking the average over the results of the single test runs. The whole cross-validation procedure is then repeated for each number of quantization labels to be evaluated (compare section 7.1).

The evaluation environment allows the user to specify which test data sets to be used, and which parameter values to be tested. It then performs the cross-validation in a batch mode. Since a file is generated for each evaluation setting, it is easy to use multiple computers to perform the testing in parallel. All results presented in chapter 7 have been produced using the evaluation environment described here.

6.6 Recognition Subsystem



Tempo: 90.50 bpm
Beat: 2/2
Volume: 0.62 %
Log:
2/2
2/2
2/2
2/2
2/2
2/2

Connection: /dev/tty.eWatch_4-SPPslave-1
Port: /dev/tty.eWatch_4-SPPslave-1
Normalization:
Adopt factor: 0.03
Amplitude scaling: 1
Buffer size: 10
Smoothing:
Window size: 1
Quantization
Amplitude cutoff: 15
Amplitude split: 0
Number of labels: 16
Recognition:
HMM file: /Users/dominik/PinocchioGestureInputSubsy
Final state offset: 10
☐ Verbose
☐ Single beats
Settings take effect only after next initialization.

(a) main screen (b) settings screen

Figure 6.6: Graphical user interface of recognition subsystem

In the final section of this chapter, I shall describe the implementation of the recognition subsystem which is realized in Objective-C. The recognition subsystem connects the conducting gesture recognition algorithms with the eWatch module and integrates them into Pinocchio, enabling the user to control the virtual orchestra's tempo and volume by means of common conducting gestures. Moreover, I followed a flexible approach while designing the recognition subsystem which allows for later replacement of single components. For example, the recognition algorithms currently implemented in MATLAB may be replaced by more efficient Objective-C implementations without requiring major changes. Additionally, it is possible to realize modules in order to support other input devices than the eWatch.

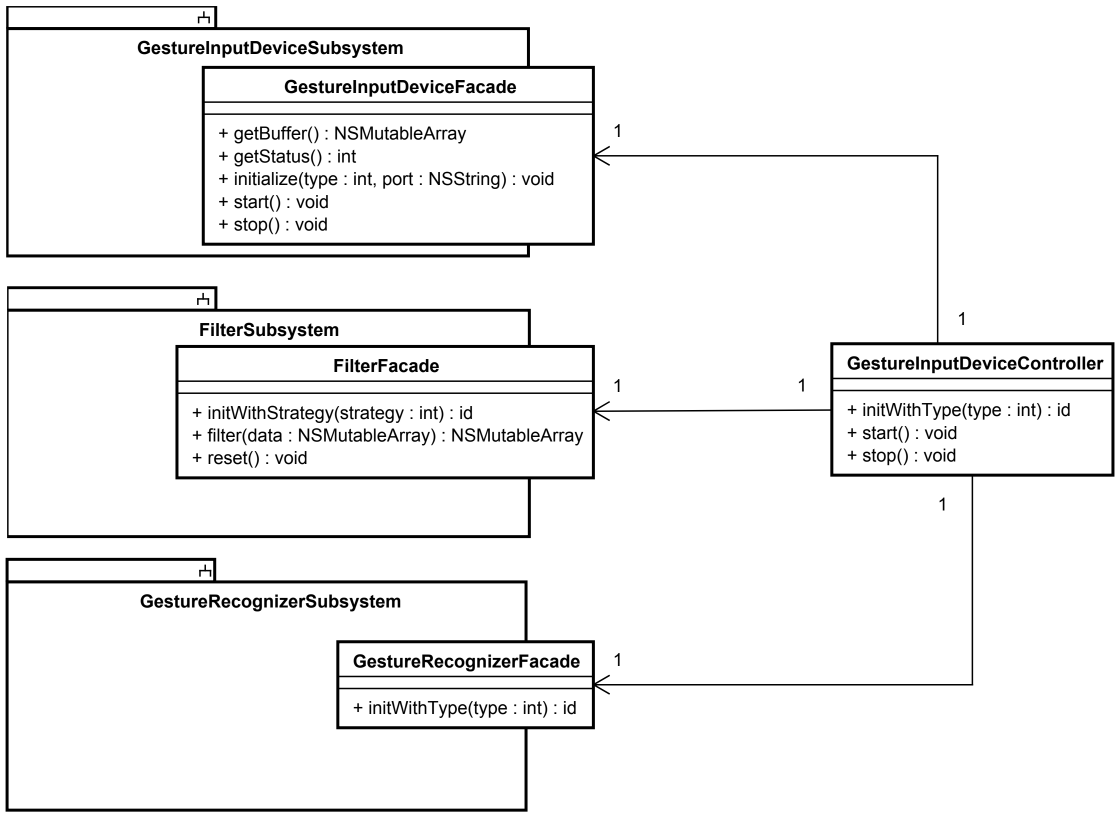

GestureInputDeviceSubsystem
GestureInputDeviceFacade
+ getBuffer() : NSMutableArray
+ getStatus() : int
+ initialize(type : int, port : NSString) : void
+ start() : void
+ stop() : void
FilterSubsystem
FilterFacade
+ initWithStrategy(strategy : int) : id
+ filter(data : NSMutableArray) : NSMutableArray
+ reset() : void
GestureRecognizerSubsystem
GestureRecognizerFacade
+ initWithType(type : int) : id
GestureInputDeviceController
+ initWithType(type : int) : id
+ start() : void
+ stop() : void
1
1
1
1
1
1

Figure 6.7: Class diagram of subsystems and controller

The recognition subsystem itself is split into three further subsystems (for the sake of simplicity referred to as subsystems in the course of this section) as shown in figure 6.7, namely into gesture input device, filter, and gesture recognizer subsystem. Each of the three subsystems contains a facade class, providing a simplified interface to the respective subsystem to be accessed by the *GestureInputDeviceController*, which controls the subsystems and acts as an access point for the host application, Pinocchio in this case, in turn. Figure 6.6 shows the recognition subsystem's graphical user interface, integrated into Pinocchio. It allows the user to select the port the eWatch is connected to and to adjust several recognition settings.

6.6.1 Gesture Input Device Subsystem

Being responsible for communicating with the input device, the gesture input device subsystem (shown in figure 6.8) is potentially able to handle different types of input devices. Currently, only the *EWatchJavaAdaptor*, which uses the Objective-C's JavaBridge to access the eWatch module (compare chapter 6.3), and its according factory classes are realized. Future versions of this subsystem may contain implementations for a native eWatch module written in Objective-C, or a module to access a Wiimote controller. Basic operations to be supported by all input devices are start and stop, to begin and end receiving input data, as well as a function to retrieve buffered input data.

6.6.2 Filter Subsystem

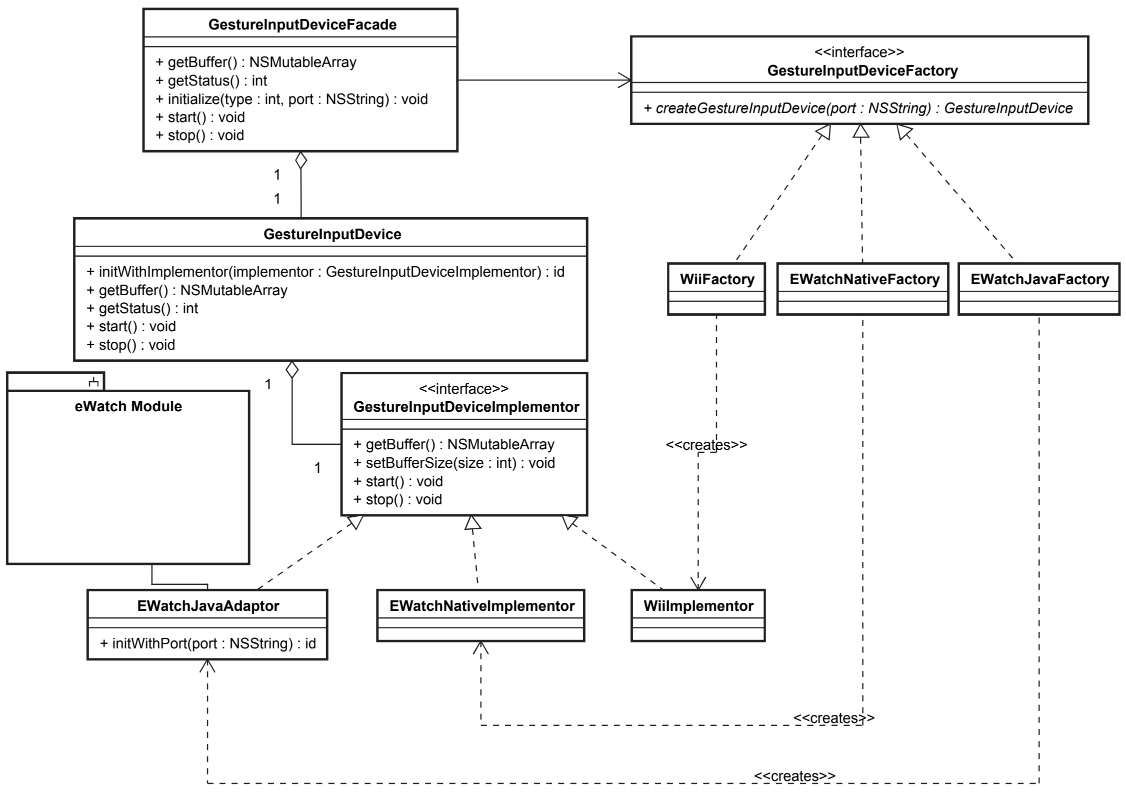

GestureInputDeviceFacade
+ getBuffer() : NSMutableArray
+ getStatus() : int
+ initialize(type : int, port : NSString) : void
+ start() : void
+ stop() : void
<<interface>>
GestureInputDeviceFactory
+ createGestureInputDevice(port : NSString) : GestureInputDevice
1
1
GestureInputDevice
+ initWithImplementor(implementor : GestureInputDeviceImplementor) : id
+ getBuffer() : NSMutableArray
+ getStatus() : int
+ start() : void
+ stop() : void
WiiFactory
EWatchNativeFactory
EWatchJavaFactory
eWatch Module
1
1
<<interface>>
GestureInputDeviceImplementor
+ getBuffer() : NSMutableArray
+ setBufferSize(size : int) : void
+ start() : void
+ stop() : void
<<creates>>
EWatchJavaAdaptor
+ initWithPort(port : NSString) : id
EWatchNativeImplementor
WiiImplementor
<<creates>>
<<creates>>

Figure 6.8: Class diagram of gesture input device subsystem

The filter subsystem (shown in figure 6.9) is responsible for preprocessing input data provided by the gesture input device subsystem. It is centered around the *Filter* interface which defines two methods, one for actually filtering data and one for resetting the filter. Currently, only the *MatlabFilterAdaptor* is implemented which accesses MATLAB functions exported as C libraries (compare section 6.2). Native filter implementations in Objectiv-C may be realized in the future to improve efficiency for a non-prototype system. An instance of a subclass of the *FilterStrategy* interface defines filters and their order to be used. *MatlabFilterStrategy* for example passes incoming data first to the smoothing, then to the calibration, and then to the vector quantization function.

6.6.3 Recognizer Subsystem

The gesture recognizer subsystem (shown in figure 6.9) handles input data which have been preprocessed by the filter subsystem before in order to recognize gestures. When a gesture is recognized, notifications are sent to Pinocchio which in turn adjusts tempo and volume accordingly. Currently, the *MatlabViterbiAdaptor* is the only implementation of the *GestureRecognizer* interface; it accesses the Viterbi algorithm implemented as MATLAB function using an exported C library.

Both, the filter as well as the gesture recognizer subsystem, make use of the *MatlabLibraryProxy* class which realizes a singleton pattern to make sure that the MATLAB Component Runtime is only initialized once, and that it is terminated when the host application itself terminates. In addition, *GestureInputDeviceSettings* holds several global settings for the filters and algorithms used in the two subsystems.

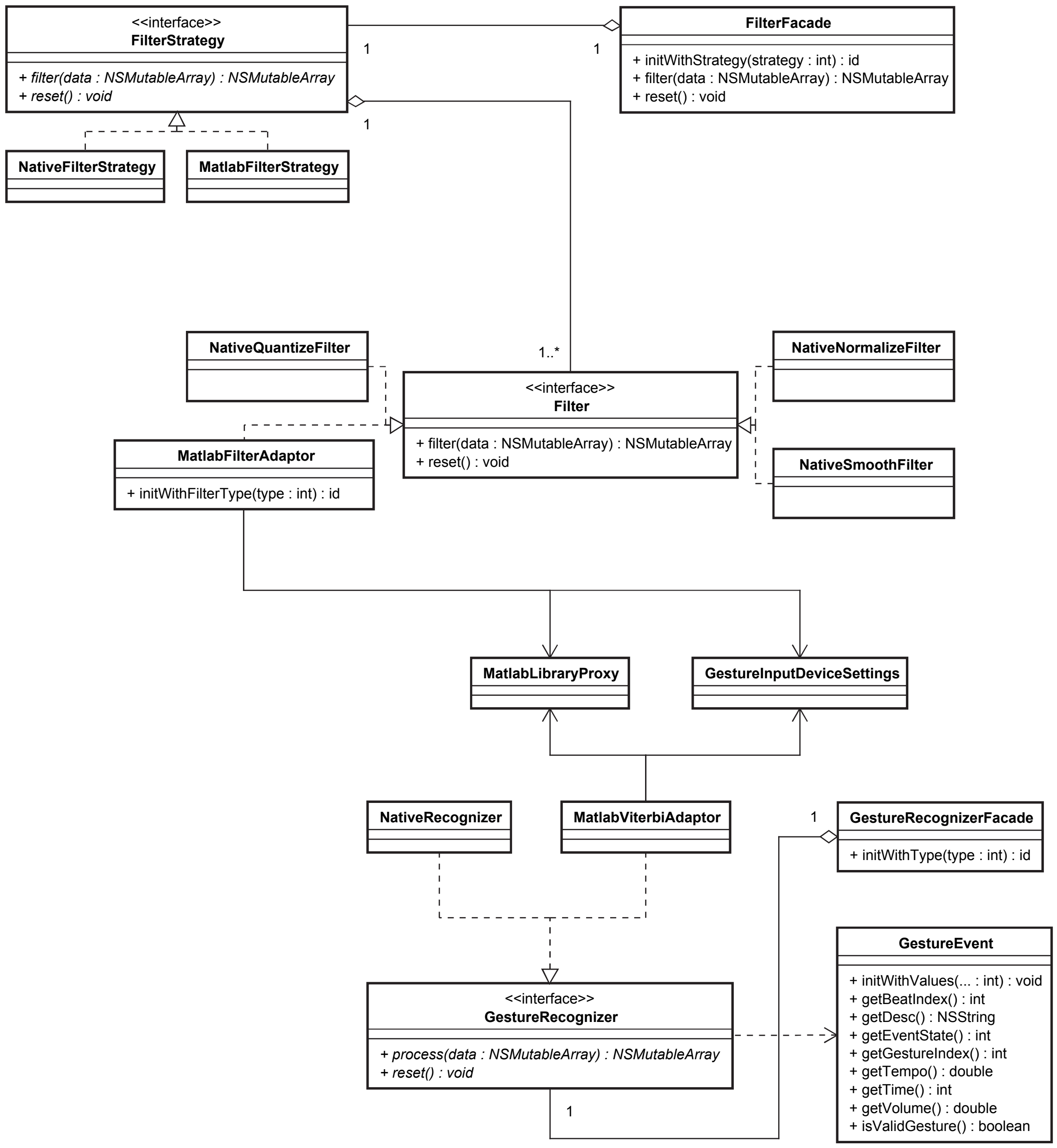


Figure 6.9: Class diagram of filter and recognizer subsystems

In this chapter, I discussed the implementation of modules and algorithms that have been introduced in the previous chapter. Before actually dealing with gesture data coming from the eWatch, I implemented a testbed using mouse gestures to evaluate initial versions of the recognition algorithms. The implementations of the eWatch module, responsible for communicating with the eWatch, and the recognition algorithms, realizing the core functionality of recognizing gestures, are both used by the evaluation environment as well as the recognition subsystem. While the recognition subsystem is implemented in Objective-C and responsible for the integration into Pinocchio, the evaluation environment is implemented in MATLAB and able to use prerecorded gesture test data to perform cross-validations in order to optimize parameters settings. Results of such an evaluations are described in the following chapter.

7 Evaluation

In this chapter, I shall provide an evaluation of the gesture recognition system developed and motivate decisions with regard to preprocessing and recognition algorithm settings, as introduced in chapter 5. These decisions are based on empirical evaluations. As described above, I used the evaluation environment introduced in the previous chapter to collect gesture data from different users performing different gestures. After describing basic conditions and methods in section 7.1, I shall motivate the evaluations performed and present their results in section 7.2. In section 7.3, I shall conclude this chapter with a discussion of implications and limitations that appeared during this evaluation.

7.1 Method

Evaluation Results Structure Due to the high number of test data recordings and parameter values that can vary, the evaluation environment uses a hierarchical data structure to store test results. Before introducing this structure, I shall define several terms that will be used in the following discussion:

- *Test data recording:* A two minute acceleration data recording containing data for one single gesture which was repeatedly performed by the same user at a fixed tempo. Additionally, a test data recording contains beat annotations inserted as described in section 6.4.
- *Test data recording set:* Set of g test data recordings, g being the number of gestures the system is able to recognize, i.e. three in all examples presented so far, namely 2-, 3-, and 4-beat gestures. A test data recording set contains exactly one test data recording for each gesture.
- *Setting:* Set of parameter values that potentially influence the recognition performance. For example, the number of states to be used in the HMM is one of the recognizer's parameters.
- *Setting set:* Set of settings, usually with the values of one single parameter varying[17] from setting element to setting element. For example, the first element of the setting set may have a value of ten for the number of HMM states, the second may have a value of eleven, and so on.

Figure 7.1 depicts the hierarchy of different tests performed during an evaluation. I shall introduce the different test levels in the following paragraphs:

- *Parameter evaluation:* The goal of a parameter evaluation is to find optimal values for a parameter with respect to recognition performance and the test data available. A parameter evaluation consists of k test suites, which all use the same setting set but different test data recording sets. For example, The first test suite may use test data for training and recognition coming from the same user (user-dependent test), while the second test suite may use test data for training coming from a different user than the test data for recognition (user-independent test).
- *Test suite:* A test suite is defined for a setting set containing l settings and multiple test data recording sets. The evaluation environment allows the user to specify which of the test data recording sets to be used for training and which for recognition. The number of test data recording sets assigned for recognition, m, determines the number of test runs one level below. Note that it is possible to assign a single test data recording set to both, training and

[17] Varying more than one parameter from setting to setting makes it difficult to identify the source for a potential change in recognition performance.

recognition. In such a case, a cross-validation is performed. The evaluation environment goes through all elements contained in the specified setting set and performs a setting test for each of them.

- *Setting test:* A fixed setting is used in a setting test. The evaluation environment performs a test run for every test data recording set assigned for recognition one level above, i.e. it performs *m* test runs. Note that test data used for recognition is never used for training during the same test.
- *Single test:* A single test is performed for a single test data recording, i.e. test data containing the performance of one gesture. The setting to be used is determined by the according setting test.

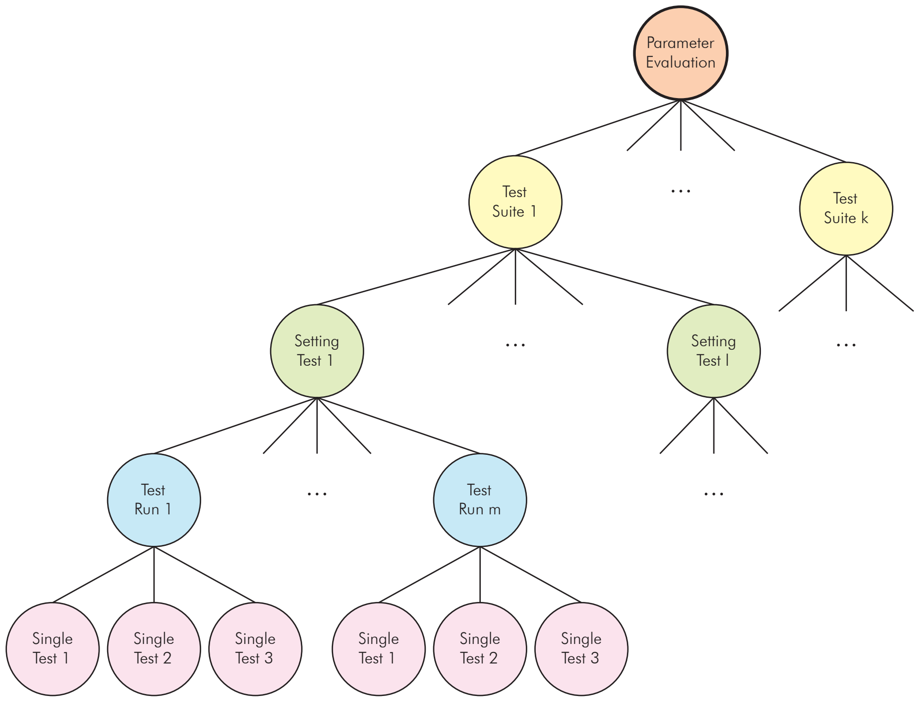


Figure 7.1: Evaluation results structure

Test Data Recordings In contrast to the initial two user studies with orchestra conductors, I am using test data collected from four different users for evaluating the gesture recognizer. All the users are graduate students of electrical or computer engineering between the age of twenty-four and twenty-six. This choice is due to the rather limited availability of conductors for user studies. One of the four users who contributed to the test data is female, three are male. Neither of them has a background knowledge in musical conducting. As discussed in section 1.1, Pinocchio is targeted at children without prior conducting knowledge as well.

Before recording gesture data, the users were shown how to perform the requested conducting gestures. After practicing for approximately one minute per gesture to be performed, data were recorded. As described in section 6.4, conducting gestures are performed following a metronome which is set to a fixed tempo per recording session. One recording session lasts for two minutes and consists of repeated and continuous performances of a specific conducting gesture. Using recordings longer than that results in a decrease of the user's concentration, hence in a lower accuracy of beat placement (compare 8.2).

All test data recordings contain regions with random hand movement, namely at the beginning and at the end of each recording, i.e. when the user prepares to start and before the recording is stopped. These regions are about twenty seconds in length per recording on an average and pose a challenge to the recognizer's threshold model. As can be seen from the results described below, the gesture recognizer is able to successfully reject random movements.

Seventy-two sets of test data were collected in total, resulting in approximately 144 min of gesture data. These sets cover conducting gestures for three different measures—2-, 3-, and 4-beat—and tempos of 50, 70, 90, 110, and 130 bpm. Due to the mere number of different possible settings and their according value domains, it is impossible to evaluate all possible combinations of settings. Thus, I had to focus on a selection of settings that appeared to be beneficial to test, i.e. that seemed to be of high relevance to the decisions to be made. Moreover, the order of tested parameters is important, since after a specific parameter value is found to produce good results, it will be used in subsequent test runs for the evaluation of different parameters. I chose the presented order with respect to importance of parameters as reported in the literature, and also stepped back for several settings in order to test different combinations.

Metrics The recognition process has four possible outcomes. Either a gesture is recognized correctly—further discussed below—, it was deleted (the recognizer did ignore the gesture), it got substituted (the recognizer detected a gesture but misclassified it), or it was inserted (the recognizer detected a gesture where there was none). In the course of the evaluations presented in this chapter I shall use the following metrics in order to evaluate the recognition performance with regard to varying settings.

- Detection ratio $= \frac{\text{\# of correctly recognized gestures}}{\text{\# of input gestures}}$
- Reliability $= \frac{\text{\# of correctly recognized gestures}}{\text{\# of input gestures} + \text{\# of insertion errors}}$
- Latency $=$ mean of offsets of correctly recognized gestures

As can be seen from these definitions, the detection ratio provides the number of correctly recognized gesture in relation to the number of expected input gestures. However, this metric alone is not sufficient, since it does not reflect the number of inserted gestures, i.e. the number of false positives produced by the recognizer. Therefore, the reliability is used in addition to the detection ratio.

The decision weather a recognized gesture was recognized correctly is dependent on two conditions. First, it must be recognized at the correct point in time; this condition is related to the gesture segmentation or spotting problem discussed in section 5.3. Second, it must be classified correctly, as discussed in section 5.2. While it is straightforward to verify conformance with the second condition, it is necessary to introduce a threshold in order to be able to verify conformance with the first condition. Given a sampling rate of 100 Hz and the fact that a human conductor is not able to accurately perform conducting beats at a resolution of 10 ms, I defined a time window around the expected event of an input gesture; as long as a (correctly classified) gesture is spotted within this window, it is considered to be correctly recognized. A window size of 100 samples centered around the current sample is used in all tests presented in this chapter, resulting in a maximum allowed displacement of 500 ms. However, the average latency in most of the results is many times slower.

The third metric introduced above, latency, is thus the mean of absolute offsets of correctly recognized gestures. That is, if a gesture is recognized correctly, with respect to the two conditions discussed above, the absolute value of displacement between the recognized gesture event and the expected gesture event contributes to the average latency. Recognition performance in general refers to a combination of these three metrics: the higher the detection ratio and the reliability are, and the lower latency is, the better is the recognition performance.

Whenever a mean value for one of these three metrics with respect to a specific setting is given, it has been calculated in the following way (compare figure 7.1): Results for single tests are averaged for each test run. Results for test runs are averaged for each setting test, in turn. It is apparent that it does not make sense to average over setting tests since the whole point of this evaluation is to find out which setting is best suited. If more than one test suite is used during a parameter evaluation however, setting tests are averaged across test suites in order to determine optimal values independent from test data used, using a weighted average according to the number of test data recording sets used for recognition. For example, results from setting test one of test suite one are combined with results from setting test one of test suite two, and so on.

Looking at the standard deviation at different levels of the evaluation hierarchy yields clues about different aspects with regard to the recognition and the test data used. A high standard deviation of single tests in the bottom level for an individual test run is an indication that the recognition performance varies a lot for different gestures, e.g. the system may be better in recognizing a 2-beat gesture than a 3-beat gesture. A high standard deviation at the level of test runs is an indication that the test data used yields varying results, e.g. one test data recording sets performs better than another set, using the same setting. Finally, the standard deviation's meaning for setting tests across test suites depends on the choice of test data for individual test suites. For example, if test suite one tests user-dependent recognition while test suite two tests user-independent recognition, a high standard deviation is an indication that a setting does not perform equally well for different training/recognition constellations.

7.2 Results

In this section, I shall present a selection of evaluation results that are particularly relevant to design decisions made for the gesture recognizer. Since it is not always obvious at the beginning what parameter values to test, and how to design a parameter evaluation, I conducted additional evaluations besides those presented in this section; they either lead to one of the tests presented here, or turned out not to be relevant for the decisions to be made.

Varying numbers of test sets are used for evaluating different parameter settings. This is due to the unequal availability of test data with certain characteristics for different users; it is always noted how many data sets were used for a test suite.

7.2.1 Number of States

The choice of an adequate number of states to be used in order to represent a gesture in an HMM is not apparent. Therefore, I tested various different state settings, keeping in mind that different test data characteristics, with respect to user and tempo, may behave differently. The goal is to find a state setting which is equally suitable for any of the characteristics represented in the test data.

I tested state settings with an equal number of states for each gesture to be recognized on the one hand, and state settings with the number of states corresponding to the gesture's length on the other hand. In the presented plots, the first 56 values along the x axis, i.e. $1 \leq x \leq 56$, correspond to equally distributed number of states, starting at allocating 5 states to each HMM and ending with allocating 60 states to each HMM. For $56 < x \leq 72$, a different number of states is assigned for each gesture. That is, i states are assigned to the HMM representing a 2-beat conducting gesture, $2i$ states are used for a 3-beat, and $3i$ for a 4-beat conducting gestures, with $5 \leq i \leq 20$. Table 7.1 summarizes the state setting used.

State Setting i	**Number of States**
1	(5, 5, 5)
⋮	⋮
56	(60, 60, 60)
57	(5, 10, 15)
⋮	⋮
72	(20, 40, 60)

Table 7.1: State settings used in evaluation

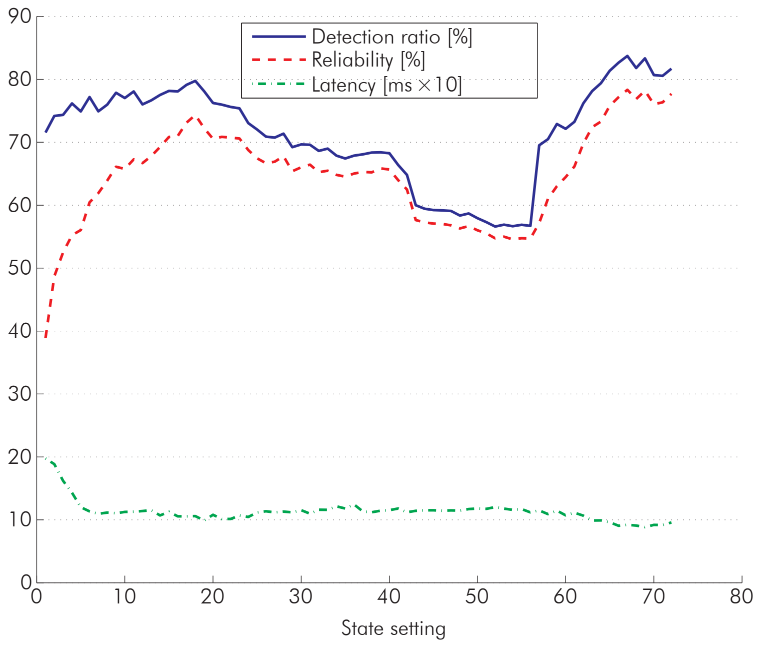


Figure 7.2: Results of first state parameter evaluation

I used three parameter evaluations: the first parameter evaluation tests user-dependent recognition, with test data from the same user who also provided training data, the second parameter evaluation tests user-independent recognition using test data from a different user, and the third parameter evaluation uses mixed training and recognition data coming from different users, respectively. Sixteen quantization labels were used and no smoothing was applied. A summary of this evaluation setup is presented in table 7.2. Due to the unequal availability of gesture data, different numbers of data sets are used depending on the test run's characteristics; this is referred to as total number of sets in the table. Figure 7.2 depicts the results of the first parameter evaluation, performing user-dependent recognition.

Table 7.3 shows the state evaluation's results. As the recognition performance for parameter evaluations two and three—testing user-independent recognition—is not sufficient, I shall focus on the user-dependent recognition in the following. User-independent recognition is discussed further in section 7.3.

Leaving out test suites which use recordings of 50 bpm and 130 bpm, the maximal recognition rate is increased from 83.72% to 88.54% with a standard deviation of 5.79% (15.98% before) and the maximal reliability is increased from 78.35% to 82.91% with a standard deviation of 5.42% (15.96% before), both for a state setting of (14, 28, 42). As can be seen from these results, extreme tempos influence the recognition performance negatively. Leaving out the 50 bpm test suite only,

Parameter Evaluation #	**Test Suite #**	**Training Data**		**Recognition Data**	
		from	**at tempo* [bpm]**	**from**	**at tempo* [bpm]**
1	1	User A	90 (3)	User A	50 (1)
	2				70 (2)
	3				90 (3)
	4				110 (2)
	5				130 (1)
	6		70 (1)		50 (1)
	7		90 (1)		70 (1)
	8		110(1)		90 (1)
	9				110 (1)
	10				130 (1)
2	1	User A	90 (3)	User B	90 (2)
	2			User C	90 (2)
	3			User D	90 (2)
3	1	User A-D	90 (4)	User A-D	90 (4)
	2				

Table 7.2: State evaluation setup
* total number of sets specified in brackets

the recognition performance increases further (maximal recognition rate: 88.87% with a standard deviation of 5.04%; maximal reliability: 83.40% with a standard deviation of 5.03%). These results suggest that accelerations caused by conducting gestures at a tempo of 50 bpm are too low in order to enable a reliable recognition using the method presented here. I shall discuss this issue further in section 7.3.

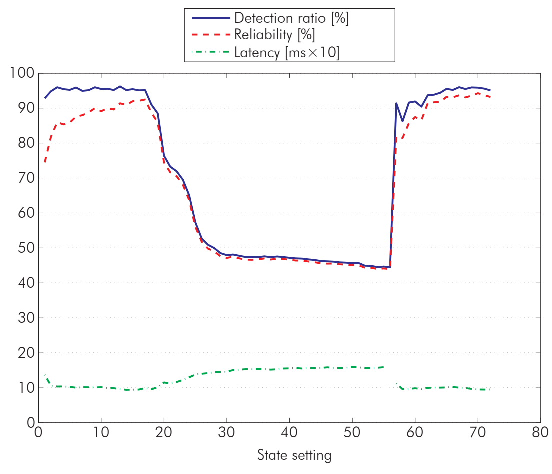
A value of 13.14% (15.75%) as average over all standard deviations of the detection ratio (reliability) at the single tests level[18]—indicating how the recognition performance varies for different conducting gestures—suggests that certain conducting gestures are recognized better than others. Figure 7.3 shows the first parameter evaluation's result (leaving out 50 bpm test data recording sets) split by conducting gesture. As can be seen from these plots, the 3-beat gesture performs worse than the 2-beat or 4-beat gesture. The 4-beat gesture is recognized better for a greater variety of state settings.

The best recognition performance for the user-dependent evaluation was achieved when assigning 14 states to the HMM representing a 2-beat gesture, 28 states to the HMM representing a 3-beat gesture, and 48 states to the HMM representing a 4-beat gesture. This result does not guarantee that such a state setting still yields the best recognition performance if other parameter values are changed. However, since it is not feasible to test all possible combinations of parameter values, I shall use this state setting in the further evaluation.

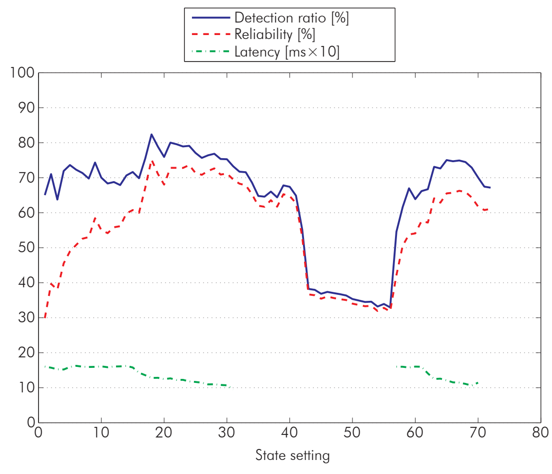
7.2.2 Number of Labels

In the evaluation so far, I used sixteen labels for vector quantization (compare section 5.2.2). Tests conducted earlier in the gesture recognizer's development phase suggested that using fewer than sixteen labels essentially decreases recognition performance. Using more labels increases memory and computational costs, since the HMM handles more observation symbols, resulting in larger

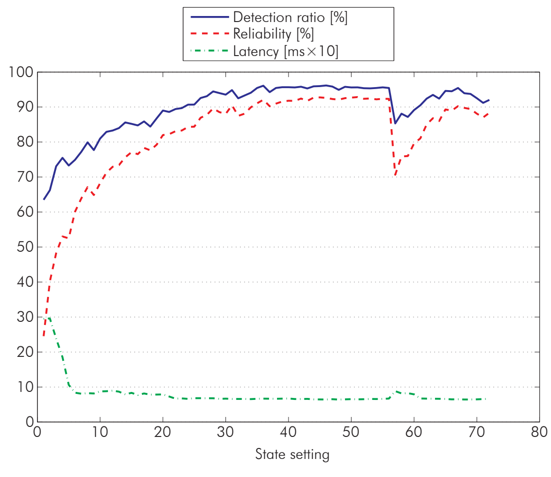
[18]The presented values were determined in the first parameter evaluation, leaving out 50 bpm test data recording sets.



(a) 2-beat



(b) 3-beat



(c) 4-beat

Figure 7.3: Results of first state parameter evaluation, partitioned by conducting gesture

Parameter Evaluation #	Max. Recognition Rate [%]	Max. Reliability [%]	Recognition Data [ms]
1	83.72 (15.98) for (14, 28, 42) states	78.35 (15.96) for (14, 28, 42) states	90.7 (39.2) for (14, 28, 42) states
2	53.94 (9.13) for (9, 9, 9) states*	37.38 (3.25) for (13, 13, 13) states*	**
3	71.90 (2.64) for (5, 5, 5) states*	60.22 (0.77) for (14, 28, 42) states*	**

Table 7.3: State evaluation results
* Considering the high fluctuation in recognition performance and the low number of states (numbers this low did not yield suitable result in the first parameter evaluation, which showed a more stable behavior in general) these results are likely to be random; i.e. recognition does not work for these settings and test data selection.
** Not specified because of insufficient recognition performance.

matrices for representing the HMM. In addition, it is likely that more training data are needed than before.

In order to find out if increasing the number of labels also increases the recognition performance, I performed a parameter evaluation using two test suites for user-dependent recognition, as well as a parameter evaluation using two test suites for user-independent recognition, both times testing sixteen versus thirty-two quantization labels. While detection ratio and reliability stayed constant for the user-dependent evaluation, detection ratio decreased by 7.55 percentage points and reliability by 5.5 percentage points for the user-independent evaluation. Therefore, I shall use sixteen quantization labels in the further evaluation.

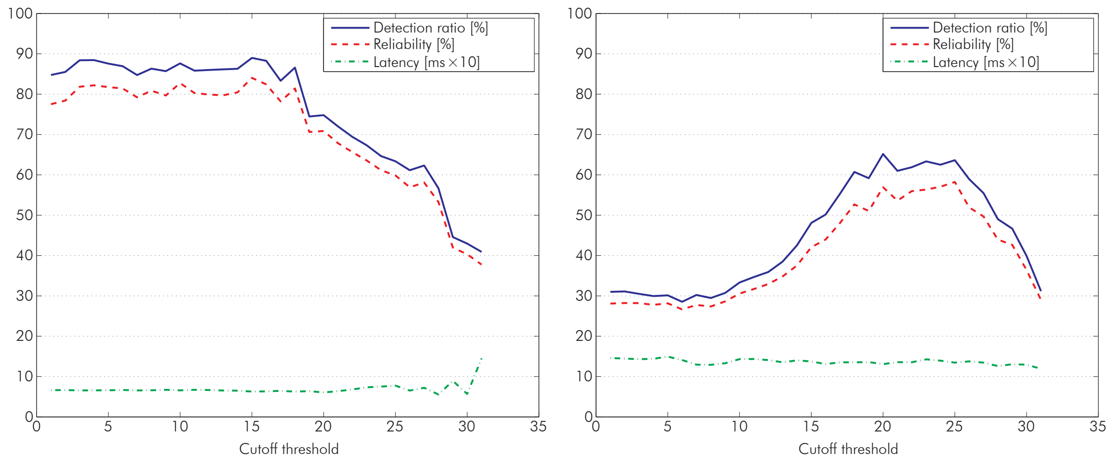
7.2.3 Cutoff Threshold

Parameter Evaluation #	Test Suite #	Training Data		Recognition Data	
		from	at tempo* [bpm]	from	at tempo* [bpm]
1	1	User A	90 (3)	User A	90 (1)
	2				110 (2)
2	1	User A	90 (3)	User B	90 (2)
	2			User C	90 (2)

Table 7.4: Cutoff threshold evaluation setup
* total number of sets specified in brackets

The vector quantization algorithm (compare section 5.2.2) allows to specify a cutoff threshold th_c. That is, if the acceleration vector's norm is smaller than th_c it is set to zero in order to minimize perturbation due to noise. Table 7.4 gives a summary of the evaluation setup used to analyze the cutoff threshold's influence on recognition performance. Again, two parameter evaluations are defined—one user-dependent and one user-independent—to evaluate values $0 \leq th_c \leq 30$.

As can be seen from the evaluation's results in figure 7.4, the user-dependent recognition performance is not particularly affected by threshold values up to approximately 18; values greater



(a) user-dependent (b) user-independent

Figure 7.4: Results of cutoff threshold evaluation

than 18 result in a rapid drop of recognition performance, though. The best recognition performance is achieved for $th_c = 15$ with a detection ratio of 88.99% and a reliability of 84.04%. However, the user-independent recognition performance is clearly affected by lower cutoff thresholds; it increases steadily for threshold values up to approximately 20.

Since a cutoff threshold value of $th_c = 15$ does not hurt the user-dependent recognition performance while at the same time increasing the user-independent recognition performance, I shall use this threshold value in the further evaluation.

Additionally, I was interested in how the cutoff threshold affects the recognition of slowly performed conducting gestures, which yielded a fairly disappointing recognition performance before. Therefore, I also evaluated the cutoff threshold with a user-dependent test suite, using training data at 70, 90, and 110 bpm and recognition data at 50 bpm. As can be seen from the results shown in figure 7.5, increasing the cutoff threshold also increases the recognition performance, up to a certain threshold value. This finding confirmed my decision to use a value of $th_c = 15$.

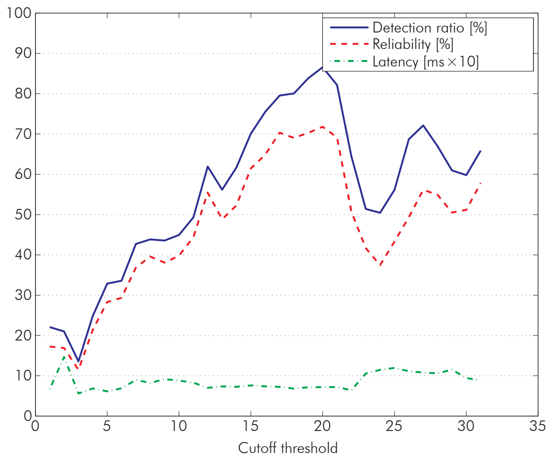


Figure 7.5: Results of cutoff threshold evaluation for 50 bpm recognition

Parameter Evaluation #	**Test Suite #**	**Training Data**		**Recognition Data**	
		from	**at tempo*** **[bpm]**	**from**	**at tempo*** **[bpm]**
1	1	User A	90 (3)	User A	90 (3)
	2		70 (1)		110 (1)
			90 (1)		
			110 (1)		
2	1	User A	90 (3)	User B	90 (2)

Table 7.5: Smoothing evaluation setup
* total number of sets specified in brackets

7.2.4 Smoothing

The last parameter I tested in the context of this evaluation is smoothing (compare 5.2.2), using a moving average algorithm with window size n; table 7.5 shows the training and recognition data used. I tested window sizes of $0 \leq n \leq 50$.

Figure 7.6 depicts the results of this parameter evaluation. As can be seen from the plot, smoothing does not have a great effect on user-dependent recognition performance up to values of $n = 28$. For greater values, the recognition performance drops. The results of the user-independent evaluation show more fluctuation; the recognition performance seems to be slightly higher for values of $11 \leq n \leq 25$.

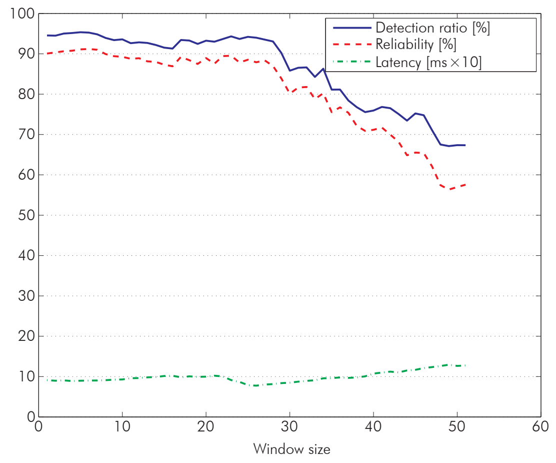
It is interesting to notice that the latency is not particularly affected by smoothing; even for values of $n = 20$—introducing a latency of 200 ms—there is almost no change in the resulting latency. This behavior may be explained with the test data's periodic nature. Since test data recordings contain continuously performed conducting gestures, and since smoothing is also applied to test data before training, it becomes apparent that not the actual gestures are trained anymore, but a shifted version of them. Thus, gestures are in fact recognized with a very low latency—since the recognizer was trained on shifted data. This approach becomes problematic if the shifting introduced due to smoothing becomes too large, or if isolated gestures are supposed to be recognized.

Since smoothing does not improve recognition performance, and since it introduces latency which may cause problems for isolated gesture recognition, it is not used in this conducting gesture recognition system.

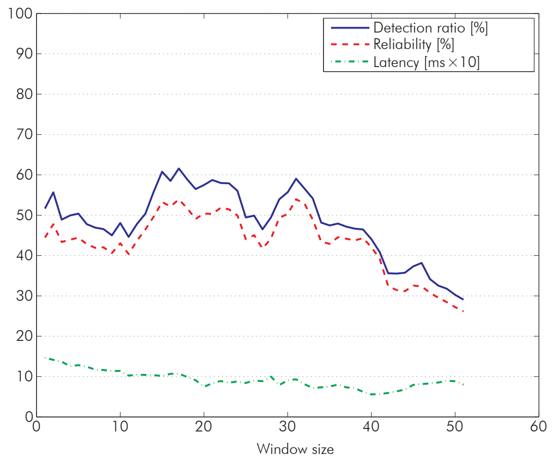
7.3 Discussion

Table 7.6 summarizes the parameter values chosen for the conducting gesture recognizer based on the evaluations' results presented in the previous section. As already stated earlier, the order I used to evaluate the different parameters potentially influenced this result; choosing a certain parameter for subsequent evaluations alters their basic conditions. However, since it is not feasible to test any possible combination of parameter values, I chose the parameter's order by the magnitude of their potential impact, guided by the reviewed literature

Furthermore, due to limitations in time and availability of test subjects, I was not able to use the same number of test data recordings during all of the evaluation steps. However, when analyzing results of similar test sets, it turns out that their recognition performance is similar as well. An evaluation using 130 bpm data for recognition yields almost the same results compared to using 110 bpm data for recognition, for example. Therefore I am confident that the results presented are suitable for the recognition system proposed.



(a) user-dependent



(b) user-independent

Figure 7.6: Results of smoothing evaluation

Parameter	Value
Number of states	(14, 28, 42)
Number of labels	16
Cutoff value	15
Smoothing	None

Table 7.6: Summary of decisions made based on evaluation

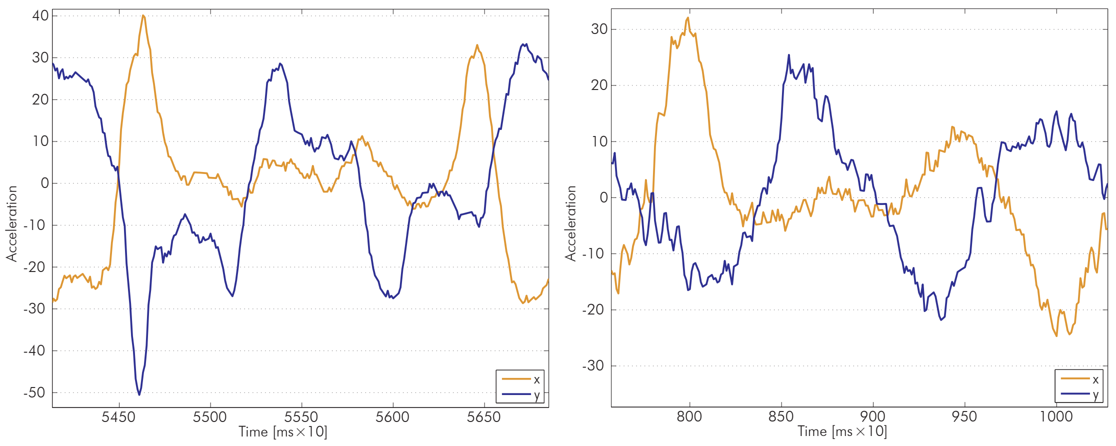
The maximal detection rate achieved for user-dependent recognition applying the chosen parameter values is 94.56%, the maximal reliability is 90.07%, and the mean latency is 91.14 ms (compare figure 7.6). For a further discussions on approaches to improve the recognition performance, using beat prediction for example, refer to chapter 9. In the following, I shall discuss several limitations of the conducting system that became apparent during the evaluation.

User-Independence As can be seen from the evaluation results presented in this chapter, the user-independent recognition performance is not satisfactory: detection rates lie around 50% while the reliability is even lower.

It is expected that the recognition using gesture data from a user who did not contribute to the data that were used to train the recognizer's HMM performs worse than the recognition using gesture data from the same user. However, since the achieved recognition performs seems to be exceptionally slow, I analyzed the test data recordings and compared data coming from different users.

As can be seen from figure 7.7, it seems that the level of noise varies strongly among different users. A possible reason for this shortcoming may be found in the way the eWatch is currently worn, namely using a second eWatch to enable an upright position of the first eWatch responsible for data collection (compare section 4.3). This way of mounting is less stiff and makes the eWatch jiggle more—depending on how firm the wrist band is tight to the user's wrist. Since different users have different preferences with regard to how firm they close the wrist band, the eWatch jiggles with varying strength, thus resulting in a different noise profile.

The integration of a 3D accelerometers into the eWatch is currently in progress and is expected to be completed by the end of 2007. A 3D accelerometer would not only enable the recognition of



(a) user A (b) user C

Figure 7.7: Different levels of noise in acceleration data of a 4-beat conducting gesture at 90 bpm

more gesture variations due to the additional dimension, but would also solve the mounting problem mentioned above, potentially resulting in a better user-independent recognition performance.

Tempo While the recognition of conducting gestures at high tempos up to 130 bpm did not affect the recognition performance, slow conducting gestures at 50 bpm were not recognized well by the system (compare section 7.2.1), potentially due to the acceleration's low amplitude. Since the introduction of a cutoff threshold, i.e. the reduction of noise for low acceleration amplitudes, helped to improve the recognition performance—the detection ratio increased by 48.01 percentage points (compare figure 7.5)—a reduction of noise, coming along with wearing the eWatch as it is supposed to be worn as discussed in the previous section, is likely to improve the recognition performance for slow conducting gestures.

Additional Gestures The proposed system is currently limited to the recognition of three different conducting gestures. This is sufficient for a large variety of different pieces of classical music, especially since tempo and volume are extracted as well. While it is straightforward to include additional gestures by means of simply training them, a higher number of gestures increases the number of states which in turn increases the number of computational steps required for performing the recognition. Moreover, confusion between different gestures becomes more likely, depending on the gesture's similarity.

In this chapter, I introduced a method for the systematic evaluation of the conducting recognition system using test recordings from several users. After defining the evaluations to be performed I presented the results and concluded with a discussion of the system's limitations. In the next chapter, I shall give a summary of the ideas and contributions of this thesis.

8 Conclusions

In this chapter, I shall summarize the work presented so far and discuss strengths as well as limitations of the proposed gesture recognition approach, while comparing the achieved results to prior systems.

In this thesis, I described the development of a gesture-base input device, focusing on musical conducting gestures as example application. The eWatch, a wearable sensor platform with a wrist-watch form factor, transmits acceleration data in two dimension at a sampling frequency of 100 Hz via Bluetooth to the host computer, running the recognition system which is embedded into Pinocchio, a virtual symphony orchestra. By using the eWatch as an input device, the system is able to continuously recognize different conducting gestures, representing different measures, and extract the conveyed information about tempo and volume to control the virtual orchestra.

The gesture recognition system consists of several modules for preprocessing the raw acceleration data received from the eWatch. After calibrating, i.e. eliminating the gravity's effect, the data are quantized into labels, using the acceleration's direction. The task of gesture recognition consists of two subtasks, namely gesture spotting and gesture classification. In the proposed approach, a discrete hidden Markov model (HMM) is used to perform gesture recognition in a single stage. After training models for the gestures to be recognized separately, these single models are integrated into a combined model. A threshold model—responsible for rejecting random movement—is added to the combined model as well.

The combined model is a regular hidden Markov model in turn that allows the application of standard algorithms. In order realize a continuous recognition of gestures, the Viterbi algorithm is adopted to handle continuous data streams as input. After a gesture is recognized, its conveyed tempo and volume information is extracted and sent to the host system which adjusts the virtual orchestra's performance accordingly.

8.1 Strengths

In discussing this approach's strengths, I shall stick to the functional and non-functional requirements defined above in section 4.1. The approach as presented is able to find and classify common conducting gestures (as described by [Rudolf, 1950]) which are continuously performed by the user with his or her arm. The recognized gestures are interpreted, i.e. tempo and volume are extracted. Moreover, it is possible to recognize arbitrary gestures by means of providing according training data. Although the system is capable of recognizing gestures performed by different users, the respective user has to provide training data first in order to enable a reliable recognition.

The eWatch is worn around the wrist, a common, socially accepted place. It uses accelerometers to capture the user's gestures. In contrast to vision-based approaches, accelerometers do not rely on special requirements with respect to lighting conditions. To compensate this shortcoming of camera-based systems, a special baton is often used that either holds a light emitter or is augmented with a color marker. Accelerometers do not necessarily have to be built into the baton as can be seen from the system described in this thesis. Therefore, the user is free to chose any baton he or she likes or is accustomed to[19].

Due to their small size, accelerometers can be used in an unobtrusive way. While vision-based systems may be even less obtrusive, accelerometers do not restrict the user's range of action. Using cameras, the user has to stay withing the viewing frustum which is much smaller than the area a wireless transmitter for acceleration data can cover. Furthermore, accelerometers—as well as the other modules used to build the eWatch—are low cost components.

Using a threshold model to reject non-meaningful gestures, the recognition system presented here is fairly robust; it reaches a reliability of more than 90% and a recognition rate of more than

[19][Rudolf, 1950] indicates that a conductor choses his or her baton thoroughly with respect to several properties.

94% for user-dependent recognition, with an average latency of around 90 ms (not meeting the defined responsiveness requirement, compare section 8.2).

In the following, I shall compare these results to those achieved by prior conducting systems for user-dependent conducting gesture recognition. While reporting on an accuracy of 100%, the system proposed by [Sawada et al., 1995] has a latency of 500 ms – 1000 ms at the same time. [Usa and Mochida, 1998b] achieve an accuracy of 99.74%, using score information and beat prediction in addition to HMM, however. Although the approach presented by [Kolesnik, 2004] recognizes gestures with an accuracy of 92.2%, he states that this solution is not feasible for realtime. [Garnett et al., 2001] achieve an accuracy of 85% and [Grüll, 2005, Lee et al., 2006a] an accuracy of more than 90%, the latter with a latency of up to 675 ms, however.

Meeting the interface requirement defined above, the gesture recognition module is integrated into Pinocchio enabling the user to conduct the virtual orchestra in realtime and providing a graphical user interface for adjusting recognition parameters. No formal evaluation on availability or reliability was conducted. However, the eWatch has been used in two user studies, more than two hours of additional gesture data were collected, and the Pinocchio system with the embedded gesture recognition module was tested without issues—regarding availability or reliability—to become apparent. The eWatch' battery lasts for several hours under the conditions used here.

8.2 Limitations

Besides of the strengths discussed in the previous section, there are several limitations to be mentioned as well. Due to the fact that the eWatch houses 2D accelerometers only, the recognition system is not able to detect gestures that mainly use the third dimension. Conducting gestures may also include movements of the fingers to convey certain information. Since the eWatch is worn around the wrist, this information is not captured. In addition, the eWatch's loose mounting—as a result of the workaround discussed in section 4.3—accounts for an increased amount of noise, which is a potential cause for the insufficient user-independent recognition performance (compare section 7.3). Moreover, recognition performance decreases with tempo. At 50 bpm, the detection rate drops to approximately 70% with a reliabiliy of around 62%.

The system's evaluation was limited to the usage of three gestures. While arbitrary gestures can be trained, not every gesture is equally well suited for recognition. In addition, gestures must be sufficiently different in order to be distinguishable. The accuracy of beat annotations for the collected user data relies on the user's performance. If a user does not follow the metronome's tick accurately, the resulting annotations will not match the gestures represented in the acceleration data.

Although the system is able to perform gesture recognition in realtime using a MacBook equipped with a 2.16 GHz Intel Core 2 Duo processor and 1 GB of RAM, the lowest possible latency—determined in offline evaluation—is around 90 ms on an average.

This chapter provided a summary of the presented approach with regard to strengths and limitations. In the next chapter, I shall pick up some of those limitations and discuss potential improvements in future work.

9 Future Work

In this chapter, I shall outline potential directions of future research, beginning with several ideas closely related to the approaches and implementation presented here and concluding with a more general outlook on using sensors for human computer interaction in a multimodal environment.

9.1 Input Device

The eWatch's limitation of featuring only 2D accelerometers required a workaround, which was identified as a potential cause for the decrease in user-independent compared to user-dependent recognition performance (compare section 8.2). A new version of the eWatch—currently under development at Carnegie Mellon University and Intel Research—will be equipped with 3D accelerometers. Using the third dimension enables the system to capture more information about the performed gestures. Variations between gestures, that were not recognizable using two-dimensional acceleration data only, may be detected in the future—allowing for a greater variety of gestures that can be recognized.

Besides accelerometers, the eWatch also features light, audio, and temperature sensors which are currently not used in the recognition process (the light sensor is used for synchronization only). Incorporating these sensors into the recognition process might result in features that help to increase the recognition performance.

Although users were able to wear the eWatch during conducting for an extensive amount of time (compare section 3.2), its usability would improve with a reduction in size and weight. Several of the eWatch's components are not used during gesture recognition, such as the LCD or the CPU. By removing these components, a smaller and lighter input device can be realized.

Using the eWatch's processing power to recognize gestures onboard is another possible direction of future research—keeping the CPU as a component in this case is necessary. Such an implementation would result in a lower amount of data to be transferred to the host computer, but would result in a shorter battery life due to the increased computation at the same time. However, having a gesture recognizer on the eWatch enables new areas of applications, since the eWatch can interpret gestures and send according commands independently.

9.2 Gesture Recognition

Prediction Although a recognition rate of 94.56% and a reliability of 90.07% was achieved for user-dependent recognition, further improvement is desirable (when conducting for two minutes using a 2-beat gesture at 90 bpm, there are still about five gestures not recognized correctly on an average). Besides of improving the machine learning approach itself, the use of prediction methods is subject to further investigation. Applying a tempo prediction, the expected time of the next beat may be used as an additional clue, since the tempo is not likely to vary rapidly. Several other orchestra systems are using such a technique (compare section 2.2).

Additionally, context information with regard to the piece of music being conducted may be used. [Starner et al., 1994, Starner and Pentland, 1995] use a grammar when recognizing cursive handwriting and American Sign Language to guide the recognition process, for example. A similar mechanism, i.e. applying information contained in the musical score, may be incorporated in future versions.

Single Beat Detection The detection of single beats, as described in section 5.3.2, shows promising results. Further development and evaluation is required, however, in order to realize a reliable recognition at the level of single beats. In addition, Pinocchio needs to be adopted in order to support the synchronization between the conductor's beats and the beats in the music.

Training Currently, training the recognizer, i.e. enabling it to recognize certain gestures, is done manually using the evaluation environment implemented in MATLAB. The file containing the resulting hidden Markov model is then loaded by the gesture recognition module in Pinocchio. Further implementations may contain a training module which is accessible through a graphical user interface in order to enable the training of new gestures by the end user. Such a module would prompt the user to perform a certain conducting gesture following the ticks of a metronome. Additionally, it could provide feedback on the quality of the gesture with regard to its suitability for the gesture recognizer.

In order to reduce the required amount of training data, noise can be added artificially to generate more training data sets. [Kela et al., 2006] use such an approach and report on an improvement of the recognition results for the presented gesture recognition system. Further information on the applicability of noise for increasing detectability can be found in [Kay, 2000].

Performance In order to maintain a high flexibility during the development and evaluation of the gesture recognition approach presented in this thesis, the system is implemented in MATLAB and integrated into Pinocchio by means of adopters. Results show that the system is capable of performing realtime gesture recognition with the virtual orchestra running on a Apple MacBook equipped with a 2.16 GHz Intel Core 2 Duo processor and 1 GB of RAM. However, a re-implementation of the recognition algorithms in Objective-C will reduce the recognizer's computational overhead.

In addition, the eWatch's firmware may be adopted to communicate more efficiently and save bandwidth. For debugging purposes, there is currently more data transmitted than just raw acceleration information.

9.3 Evaluation

User Study In this section, I shall discuss how the evaluation method may be improved in future work. Using a larger number of different users, who are performing gestures in a larger variety, is likely to result in a more representative result for the user-independent recognition performance. Due to time limitations, I was not able to use gesture data recorded with conductor Walter Morales in evaluating the system's recognition performance. Future research may use these data to analyze how the gesture recognition system performs when a professional musician conducts a piece of music.

Baton In the course of this thesis, gesture data for evaluation was collected from users who were not using a baton. As stated before, Walter Morales performed gestures with and without baton in the according user study. These data may be used to analyze differences and potential impacts on the recognition system. It is of interest if the acceleration profile produced by performing gestures the one or the other way is similar enough to allow a recognition when only one of the modes is used for training.

9.4 Pinocchio

In its current version, Pinocchio offers two conducting modes: The previously implemented camera-based baton recognition and the conducting gesture recognition using the eWatch which is topic of this thesis. However, these two modules are working independently. One goal of future work may include the integration of these two modules in order to compensate for module-specific shortcomings and increase the recognition performance. [Brashear et al., 2003] report on a performance increase in sign language recognition, using a hat-mounted camera in conjunction with accelerometers.

Moreover, such a sensor fusion approach may include further input devices, thus providing Pinocchio with a truly multimodal user interface. A gaze detection may be used to apply conducting gestures only to selected parts of the orchestra, addressing a specific group of musicians, for example.

9.5 Outlook

The virtual orchestra conducting system Pinocchio, with its potentially multimodal input capabilities, is an example for a system with complex user interaction. Applying the experience gained in the course of this research project, a general framework can be thought of which supports multiple input devices incorporating different types of sensors for a variety of interaction modalities. Using such a sensor fusion approach, the system is able to compensate for shortcomings that a single input device may have. Possible applications include an intelligent house, or a smart car dashboard. The combination of different input sources, capturing different aspects of the user's direct and indirect communication, may help the system to react more appropriately to the user's intention.

10 Acknowledgments

Letting the past seven months I was working on this thesis passing my mind, I realize that a lot of people helped me and contributed in different ways. In this last chapter, I would like to take the opportunity to thank them for their support.

In doing so, I would like to thank my advisors in Germany and the US, Prof. Bernd Brügge, Prof. Asim Smailagic, and Prof. Daniel P. Siewiorek, for giving me the opportunity, motivation, and freedom to work on this project, and Prof. Heinrich Hußmann for taking over the co-supervision. In addition, I would like to thank Prof. Roger B. Dannenberg, Daniel Meyer, and Walter Morales for their great support in conducting the user studies.

Furthermore, I would like to thank all my family and friends for their support, encouragement, and patience. For valuable advice, proofreading, and conducting, I would especially like to thank Joseph Bradley, Lucia Castellanos, Sophie Crass, Wolfgang Crass, Linda Estevez, Brian French, Stanislav Funiak, Max Heinig, Andreas Krause, Uwe Maurer, and Marko Skočibušić.

11 Appendix

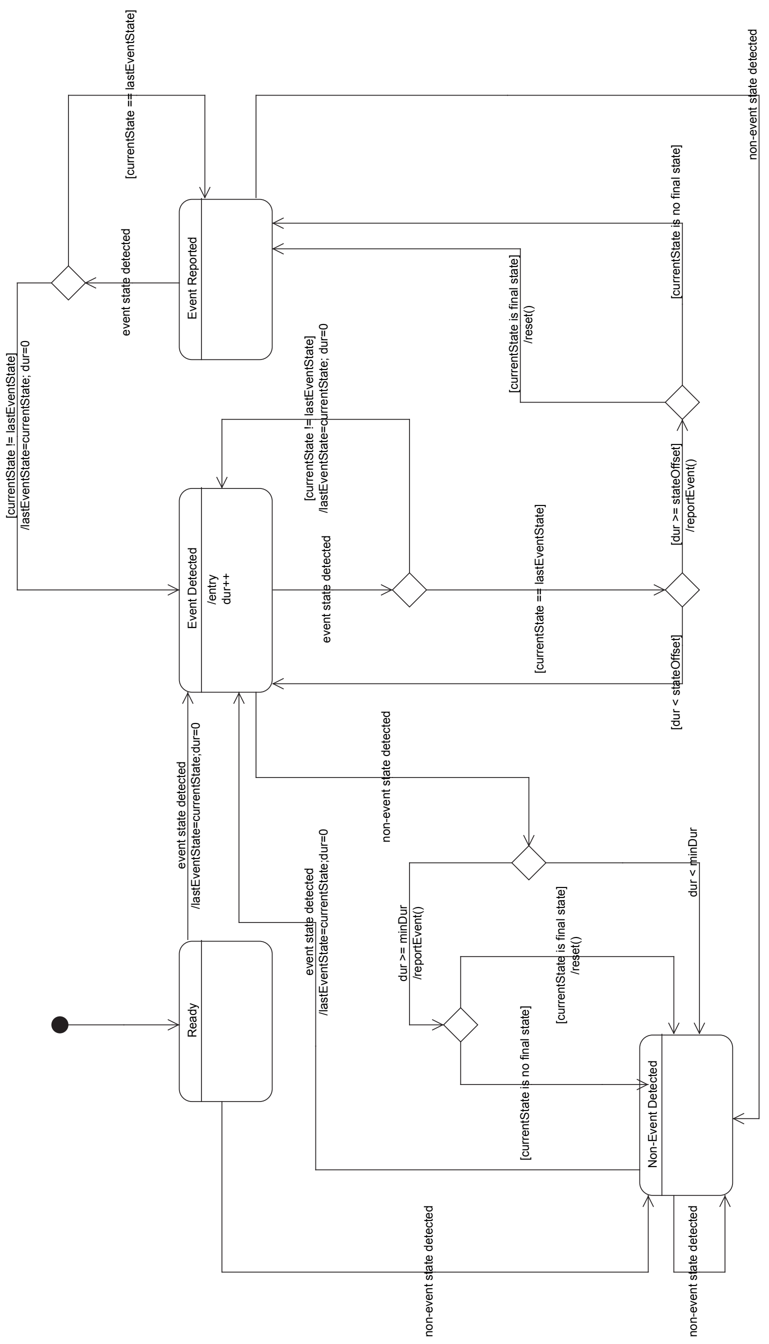
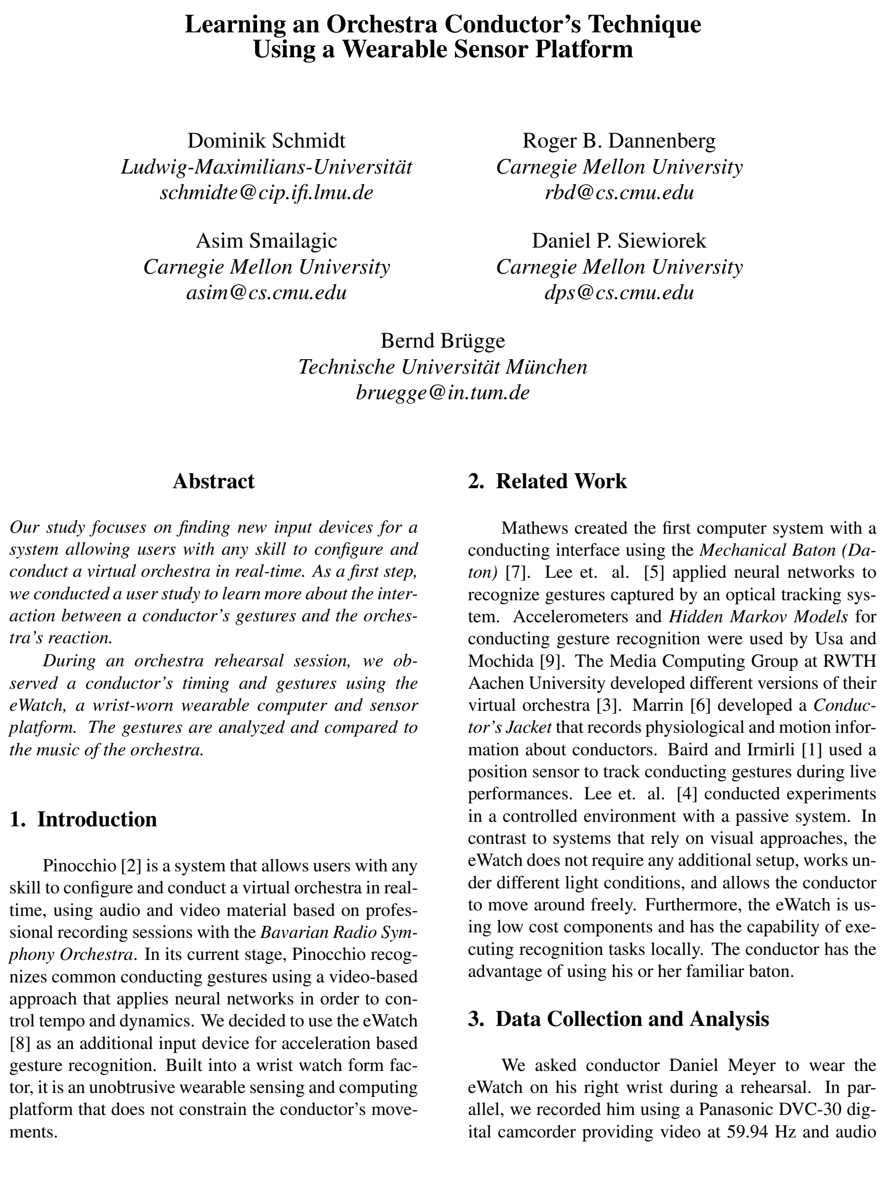

Ready
Event Detected
/entry
dur++
Event Reported
Non-Event Detected
event state detected
/lastEventState=currentState;dur=0
event state detected
/lastEventState=currentState;dur=0
event state detected
[currentState != lastEventState]
/lastEventState=currentState; dur=0
[currentState == lastEventState]
[currentState == lastEventState]
[currentState != lastEventState]
/lastEventState=currentState; dur=0
event state detected
[dur >= stateOffset]
/reportEvent()
[dur < stateOffset]
[currentState is final state]
/reset()
[currentState is no final state]
non-event state detected
non-event state detected
dur < minDur
dur >= minDur
/reportEvent()
[currentState is final state]
/reset()
[currentState is no final state]
non-event state detected
non-event state detected

Figure 11.1: State diagram for modified Viterbi algorithm



Learning an Orchestra Conductor's Technique Using a Wearable Sensor Platform

Dominik Schmidt
Ludwig-Maximilians-Universität
schmidte@cip.ifi.lmu.de

Roger B. Dannenberg
Carnegie Mellon University
rbd@cs.cmu.edu

Asim Smailagic
Carnegie Mellon University
asim@cs.cmu.edu

Daniel P. Siewiorek
Carnegie Mellon University
dps@cs.cmu.edu

Bernd Brügge
Technische Universität München
bruegge@in.tum.de

Abstract

Our study focuses on finding new input devices for a system allowing users with any skill to configure and conduct a virtual orchestra in real-time. As a first step, we conducted a user study to learn more about the interaction between a conductor's gestures and the orchestra's reaction.

During an orchestra rehearsal session, we observed a conductor's timing and gestures using the eWatch, a wrist-worn wearable computer and sensor platform. The gestures are analyzed and compared to the music of the orchestra.

1. Introduction

Pinocchio [2] is a system that allows users with any skill to configure and conduct a virtual orchestra in real-time, using audio and video material based on professional recording sessions with the *Bavarian Radio Symphony Orchestra*. In its current stage, Pinocchio recognizes common conducting gestures using a video-based approach that applies neural networks in order to control tempo and dynamics. We decided to use the eWatch [8] as an additional input device for acceleration based gesture recognition. Built into a wrist watch form factor, it is an unobtrusive wearable sensing and computing platform that does not constrain the conductor's movements.

2. Related Work

Mathews created the first computer system with a conducting interface using the *Mechanical Baton (Daton)* [7]. Lee et. al. [5] applied neural networks to recognize gestures captured by an optical tracking system. Accelerometers and *Hidden Markov Models* for conducting gesture recognition were used by Usa and Mochida [9]. The Media Computing Group at RWTH Aachen University developed different versions of their virtual orchestra [3]. Marrin [6] developed a *Conductor's Jacket* that records physiological and motion information about conductors. Baird and Irmirli [1] used a position sensor to track conducting gestures during live performances. Lee et. al. [4] conducted experiments in a controlled environment with a passive system. In contrast to systems that rely on visual approaches, the eWatch does not require any additional setup, works under different light conditions, and allows the conductor to move around freely. Furthermore, the eWatch is using low cost components and has the capability of executing recognition tasks locally. The conductor has the advantage of using his or her familiar baton.

3. Data Collection and Analysis

We asked conductor Daniel Meyer to wear the eWatch on his right wrist during a rehearsal. In parallel, we recorded him using a Panasonic DVC-30 digital camcorder providing video at 59.94 Hz and audio

Figure 11.2: Paper as accepted for IEEE ISWC 2007, page 1 of 2

at 48 kHz. The eWatch recorded light and acceleration data at 100 Hz. A LED emitting bright flashes that are clearly recognizable in the video track and in the light sensor data is used to synchronize eWatch and camcorder, thus accommodating for clock drift. Figure 1 depicts the system diagram with recording setup and analysis part.

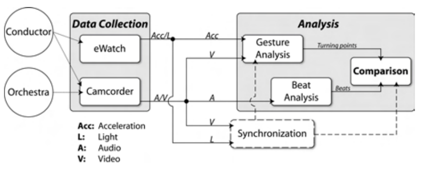


Figure 1. System diagram

A two minute long section from Brahm's Symphony No. 3 in F major, Allegro is selected for further analysis. Annotation of lower turning points in the conducting gestures (indicating the start of a measure) is done manually by observing the recorded video on a field-by-field basis. In Figure 2, velocity data of the x axis extracted from the eWatch's acceleration sensors is shown together with the annotations. The beat pattern is clearly recognizable.

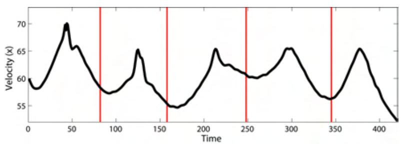


Figure 2. Velocity and gesture annotations

As a reliable audio beat extraction tool is not available, we marked beats manually by first tapping along with the music and later applying fine adjustment using an audio editor. Similar to the results presented by Lee et. al. [4] we found that the conductor conducts ahead of the beat, showing that these results also apply to a setting where the conductor can influence the performance. At a tempo of 72 bpm in the selected piece the conductor was ahead by 272 ms on average.

4. Conclusion and Future Work

The results obtained in this study affirm that the eWatch is a suitable and unobtrusive input device for this virtual orchestra system. Our goal is to build an user-independent and accurate recognition module that allows for the detection of arbitrary conducting gestures. The eWatch can be used as alternative or extension to already existent input capabilities. We are evaluating the use of *Hidden Markov Models* for the recognition of continuous gestures based on acceleration data. We wish to thank conductor Daniel Meyer for allowing us to record conducting gestures during a rehearsal.

5. Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. EEEC-540865 and Grant No. 0205266, the Defence Advanced Project Agency (DARPA) under Contract No. NBCHD030010, Pennsylvania Infrastructure Technology Alliance (PITA) — a collaboration among Commonwealth of Pennsylvania, Carnegie Mellon and Lehigh University, and a grant from Intel Corporation.

References

[1] B. Baird and O. Izmirli. Modeling the tempo coupling between an ensemble and the conductor. In *Proceedings of the International Computer Music Conference*, pages 163–166, 2001.

[2] B. Bruegge, C. Teschner, P. Lachenmaier, E. Fenzl, D. Schmidt, and S. Bierbaum. Pinocchio: Conducting a virtual symphony orchestra. In *Proceedings of the International Conference of Advances in Computer Entertainment Technology*, 2007.

[3] E. Lee, H. Kiel, S. Dedenbach, I. Grüll, T. Karrer, M. Wolf, and J. Borchers. In *CHI '06: CHI '06 Extended Abstracts on Human Factors in Computing Systems*.

[4] E. Lee, M. Wolf, and J. Borchers. Improving orchestral conducting systems in public spaces: examining the temporal characteristics and conceptual models of conducting gestures. In *CHI '05: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 731–740, 2005.

[5] M. Lee, G. Garnett, and D. Wessel. An adaptive conductor follower. In *Proceedings of the International Computer Music Conference*, pages 454–455, 1992.

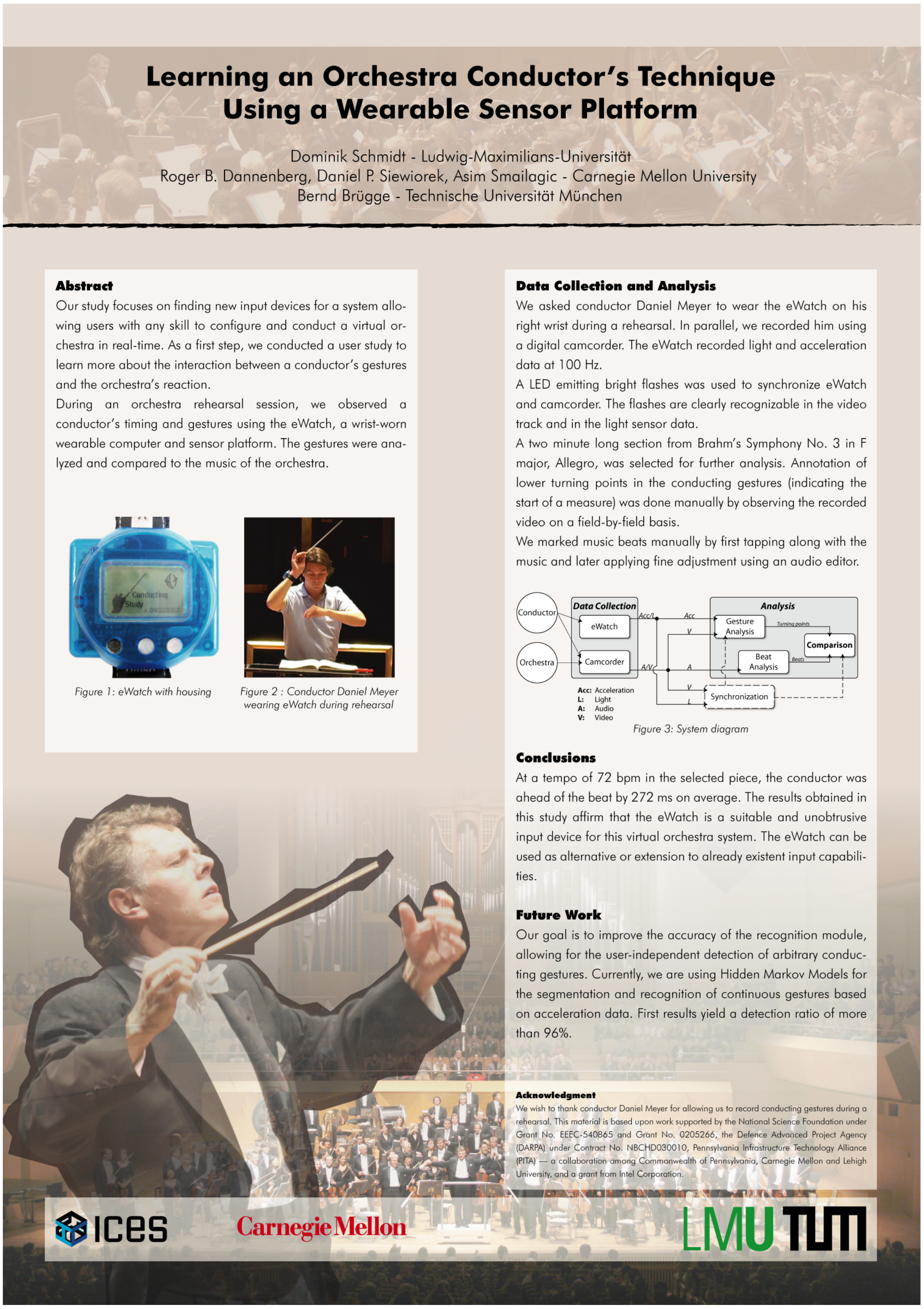
[6] T. Marrin and R. Picard. The "Conductor's Jacket": A device for recording expressive musical gestures. In *Proceedings of the International Computer Music Conference*, pages 215–219, 1998.

[7] M. V. Mathews. *The Conductor Program and Mechanical Baton*, pages 263–281. 1989.

[8] U. Maurer, A. Rowe, A. Smailagic, and D. P. Siewiorek. eWatch: A wearable sensor and notification platform. In *Wearable and Implantable Body Sensor Networks, 2006. BSN 2006.*, 2006.

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

Figure 11.3: Paper as accepted for IEEE ISWC 2007, page 2 of 2



Learning an Orchestra Conductor's Technique Using a Wearable Sensor Platform

Dominik Schmidt - Ludwig-Maximilians-Universität
Roger B. Dannenberg, Daniel P. Siewiorek, Asim Smailagic - Carnegie Mellon University
Bernd Brügge - Technische Universität München

Abstract

Our study focuses on finding new input devices for a system allowing users with any skill to configure and conduct a virtual orchestra in real-time. As a first step, we conducted a user study to learn more about the interaction between a conductor's gestures and the orchestra's reaction.
During an orchestra rehearsal session, we observed a conductor's timing and gestures using the eWatch, a wrist-worn wearable computer and sensor platform. The gestures were analyzed and compared to the music of the orchestra.

Figure 1: eWatch with housing

Figure 2 : Conductor Daniel Meyer wearing eWatch during rehearsal

Data Collection and Analysis

We asked conductor Daniel Meyer to wear the eWatch on his right wrist during a rehearsal. In parallel, we recorded him using a digital camcorder. The eWatch recorded light and acceleration data at 100 Hz.
A LED emitting bright flashes was used to synchronize eWatch and camcorder. The flashes are clearly recognizable in the video track and in the light sensor data.
A two minute long section from Brahm's Symphony No. 3 in F major, Allegro, was selected for further analysis. Annotation of lower turning points in the conducting gestures (indicating the start of a measure) was done manually by observing the recorded video on a field-by-field basis.
We marked music beats manually by first tapping along with the music and later applying fine adjustment using an audio editor.

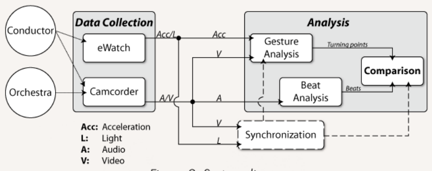


Figure 3: System diagram

Conclusions

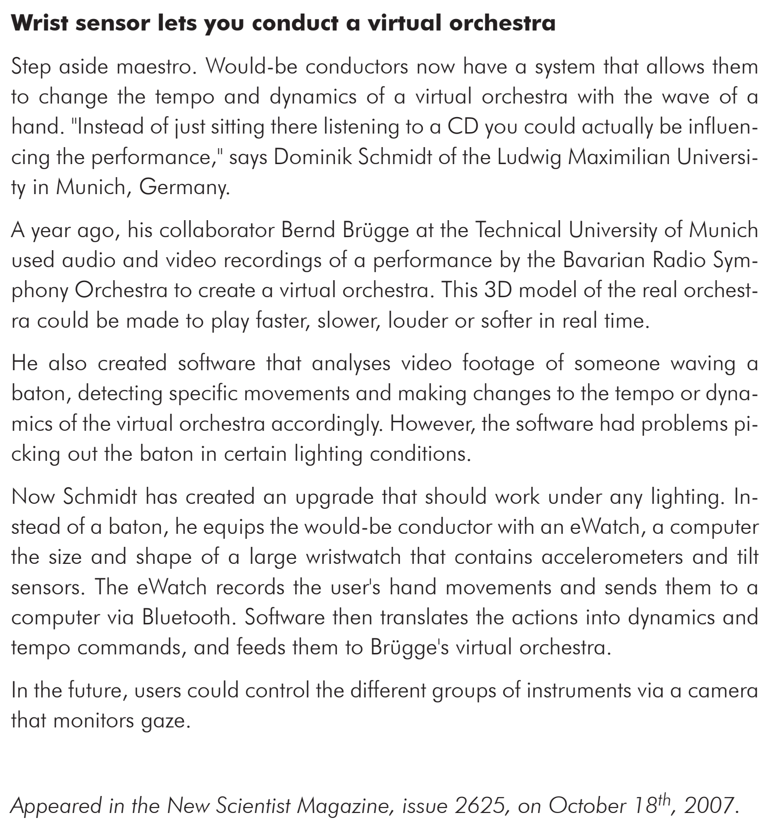
At a tempo of 72 bpm in the selected piece, the conductor was ahead of the beat by 272 ms on average. The results obtained in this study affirm that the eWatch is a suitable and unobtrusive input device for this virtual orchestra system. The eWatch can be used as alternative or extension to already existent input capabilities.

Future Work

Our goal is to improve the accuracy of the recognition module, allowing for the user-independent detection of arbitrary conducting gestures. Currently, we are using Hidden Markov Models for the segmentation and recognition of continuous gestures based on acceleration data. First results yield a detection ratio of more than 96%.

Acknowledgment
We wish to thank conductor Daniel Meyer for allowing us to record conducting gestures during a rehearsal. This material is based upon work supported by the National Science Foundation under Grant No. EEEC-540865 and Grant No. 0205266, the Defence Advanced Project Agency (DARPA) under Contract No. NBCHD030010, Pennsylvania Infrastructure Technology Alliance (PITA) — a collaboration among Commonwealth of Pennsylvania, Carnegie Mellon and Lehigh University, and a grant from Intel Corporation.

Figure 11.4: Poster as presented at IEEE ISWC 2007 in Boston, MA, USA on 10/11/2007



Wrist sensor lets you conduct a virtual orchestra

Step aside maestro. Would-be conductors now have a system that allows them to change the tempo and dynamics of a virtual orchestra with the wave of a hand. "Instead of just sitting there listening to a CD you could actually be influencing the performance," says Dominik Schmidt of the Ludwig Maximilian University in Munich, Germany.

A year ago, his collaborator Bernd Brügge at the Technical University of Munich used audio and video recordings of a performance by the Bavarian Radio Symphony Orchestra to create a virtual orchestra. This 3D model of the real orchestra could be made to play faster, slower, louder or softer in real time.

He also created software that analyses video footage of someone waving a baton, detecting specific movements and making changes to the tempo or dynamics of the virtual orchestra accordingly. However, the software had problems picking out the baton in certain lighting conditions.

Now Schmidt has created an upgrade that should work under any lighting. Instead of a baton, he equips the would-be conductor with an eWatch, a computer the size and shape of a large wristwatch that contains accelerometers and tilt sensors. The eWatch records the user's hand movements and sends them to a computer via Bluetooth. Software then translates the actions into dynamics and tempo commands, and feeds them to Brügge's virtual orchestra.

In the future, users could control the different groups of instruments via a camera that monitors gaze.

Appeared in the New Scientist Magazine, issue 2625, on October 18th, 2007.

Figure 11.5: Press coverage of poster presented at IEEE ISWC 2007

Content of Attached CD

CD 1

- *Document:* LaTeX source files, figures, PDF version of this document
- *Gesture Data:* eWatch recordings of conducting gestures from test users
- *Pictures:* Daniel Meyer wearing the eWatch, screenshots
- *References:* PDF version of papers cited (if available)

CD 2

- *Matlab:* scripts and functions, including implementations of recognition algorithms
- *Pinocchio:* Pinocchio system, including recognition subsystem

CD 3

- *Daniel Meyer*: video (recompressed) and eWatch data from user study

CD 4

- *Walter Morales*: video (recompressed) and eWatch data from user study

List of Tables

List of Figures

References

[Ailive, 2006] Ailive (2006). LiveMove white paper. Technical report.

[Apple, 2007] Apple (2007). http://developer.apple.com/documentation/cocoa/conceptual/legacy/javabridge/javabridge.html, last checked on 10/03/2007.

[AviSynth, 2007] AviSynth (2007). http://avisynth.org, last checked on 10/16/2007.

[Baird and Izmirli, 2001] Baird, B. and Izmirli, O. (2001). Modeling the tempo coupling between an ensemble and the conductor. In *Proceedings of the International Computer Music Conference*, pages 163–166.

[Bertini and Carosi, 1992] Bertini, G. and Carosi, P. (1992). Light baton: A system for conducting computer music performance. In *Proceedings of the International Computer Music Conference*, pages 73–76.

[Bianchi and Campbell, 2000] Bianchi, F. W. and Campbell, R. H. (2000). The virtual orchestra: Technical and creative issues. *Journal of Sound and Vibration*, 232(1):275–279.

[Bien and Kim, 1992] Bien, Z. and Kim, J. S. (1992). On-line analysis of music conductor's two-dimensional motion. In *IEEE International Conference on Fuzzy Systems*, pages 1047–1053.

[Bierbaum, 2007] Bierbaum, S. (2007). Investigation of multimodal interaction between a conductor and a virtual interactive orchestra. Master's thesis, Technische Universität München.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Boie et al., 1989] Boie, B., Mathews, M. V., and Schloss, A. (1989). The radio drum as a synthesizer controller. In *Proceedings of the International Computer Music Conference*, pages 42–45.

[Borchers et al., 2004] Borchers, J., Lee, E., Samminger, W., and Mühlhäuser, M. (2004). Personal orchestra: a real-time audio/video system for interactive conducting. *Multimedia Systems*, 9(5):458–465.

[Borchers et al., 2001] Borchers, J., Samminger, W., and Muser, M. (2001). Conducting a realistic electronic orchestra. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 161–162.

[Borchers, 1997] Borchers, J. O. (1997). Worldbeat: Designing a baton-based interface for an interactive music exhibit. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 131–138.

[Borchers et al., 2002] Borchers, J. O., Samminger, W., and Mühlhäuser, M. (2002). Engineering a realistic real-time conducting system for the audio/video rendering of a real orchestra. In *Proceedings. Fourth International Symposium on Multimedia Software Engineering*, pages 352–362.

[Boulanger and Mathews, 1997] Boulanger, R. and Mathews, M. V. (1997). The 1997 radio baton and improvisation modes. In *Proceedings of the International Computer Music Conference*, pages 395–398.

[Boulanger et al., 1990] Boulanger, R., Mathews, M. V., Vercoe, B., and Dannenberg, R. (1990). Conducting the midi orchestra, part 1: Interviews with Max Mathews, Barry Vercoe, and Roger Dannenberg. *Computer Music Journal*, 14(2):34–46.

[Brashear et al., 2003] Brashear, H., Starner, T., Lukowicz, P., and Junker, H. (2003). Using multiple sensors for mobile sign language recognition. pages 45–52.

[Brecht and Garnett, 1995] Brecht, B. and Garnett, G. E. (1995). Conductor follower. In *Proceedings of the International Computer Music Conference*, pages 185–186.

[Brügge and Dutoit, 2003] Brügge, B. and Dutoit, A. H. (2003). *Object-Oriented Software Engineering Using UML, Patterns, and Java.* Prentice Hall, Englewood Cliffs, NJ, second edition.

[Brügge et al., 2007] Brügge, B., Teschner, C., Lachenmaier, P., Fenzl, E., Schmidt, D., and Bierbaum, S. (2007). Pinocchio: conducting a virtual symphony orchestra. In *ACE '07: Proceedings of the international conference on Advances in computer entertainment technology*, pages 294–295, New York, NY, USA. ACM Press.

[Buxton et al., 1980] Buxton, W., Reeves, W., Fedorkow, G., Smith, K. C., and Baecker, R. (1980). A microcomputer-based conducting system. *Computer Music Journal*, 4(1):8–21.

[Campbell et al., 1996] Campbell, L. W., Becker, D. A., Azarbayejani, A., Bobick, A. F., and Pentland, A. (1996). Invariant features for 3-d gesture recognition. In *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*, pages 157–162.

[Chambers et al., 2002] Chambers, G. S., Venkatesh, S., West, G. A. W., and Bui, H. H. (2002). Hierarchical recognition of intentional human gestures for sports video annotation. volume 2, pages 1082–1085 vol.2.

[Darwiin, 2007] Darwiin (2007). http://sourceforge.net/projects/darwiin-remote/, last checked on 10/19/2007.

[Dillon et al., 2006] Dillon, R., Wong, G., and Ang, R. (2006). Virtual orchestra: An immersive computer game for fun and education. In *CyberGames '06: Proceedings of the 2006 international conference on Game research and development*, pages 215–218.

[Fenzl, 2007] Fenzl, E. (2007). Methods for multimodal input for the interaction with distributed systems using a virtual orchestra. Master's thesis, Technische Universität München.

[Garnett et al., 2001] Garnett, G. E., Jonnalagadda, M., Elezovic, Johnson, T., and Small, K. (2001). Technological advances for conducting a virtual ensemble. In *Proceedings of the International Computer Music Conference*, pages 167–169.

[Garnett et al., 1999] Garnett, G. E., Malvar-Ruiz, F., and Stoltzfus, F. (1999). Virtual conducting practice environment. In *Proceedings of the International Computer Music Conference*, pages 371–374.

[Grüll, 2005] Grüll, I. (2005). conga: A conducting gesture analysis framework. Master's thesis, Universität Ulm.

[Günter and Bunke, 2003] Günter, S. and Bunke, H. (2003). Optimizing the number of states, training iterations and gaussians in an hmm-based handwritten word recognizer. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 472–476 vol.1.

[He and Kundu, 1991] He, Y. and Kundu, A. (1991). 2-d shape classification using hidden markov model. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(11):1172–1184.

[HMMToolbox, 2007] HMMToolbox (2007). http://www.cs.ubc.ca/ murphyk/software/hmm/hmm.html, last checked on 10/04/2007.

[Hofmann et al., 1998] Hofmann, F. G., Heyer, P., and Hommel, G. (1998). *Velocity Profile Based Recognition of Dynamic Gestures with Discrete Hidden Markov Models.*

[Huang et al., 1990] Huang, X., Ariki, Y., and Jack, M. (1990). *Hidden Markov Models for Speech Recognition.* Columbia University Press, New York, NY, USA.

[Iba et al., 1999] Iba, S., Weghe, J. M. V., Paredis, C. J. J., and Khosla, P. K. (1999). An architecture for gesture-based control of mobile robots. In *Intelligent Robots and Systems, 1999. IROS '99. Proceedings. 1999 IEEE/RSJ International Conference on*, volume 2, pages 851–857 vol.2.

[Ilmonen, 1999] Ilmonen, T. (1999). Tracking conductor of an orchestra using artificial neural networks. Master's thesis, Helsinki University of Technology.

[Ilmonen and Jalkanen, 2000] Ilmonen, T. and Jalkanen, J. (2000). Accelerometer-based motion tracking for orchestra conductor following. In *Proceedings of 6th Eurographics Workshop*, pages 187–196.

[Ilmonen and Takala, 1999] Ilmonen, T. and Takala, T. (1999). Conductor following with artificial neural networks. In *Proceedings of the International Computer Music Conference*, pages 367–370.

[Kallio et al., 2003] Kallio, S., Kela, J., and Mantyjarvi, J. (2003). Online gesture recognition system for mobile interaction. volume 3, pages 2070–2076 vol.3.

[Kang et al., 2004] Kang, H., Lee, C. W., and Jung, K. (2004). Recognition-based gesture spotting in video games. *Pattern Recogn. Lett.*, 25(15):1701–1714.

[Kay, 2000] Kay, S. (2000). Can detectability be improved by adding noise? *Signal Processing Letters, IEEE*, 7(1):8–10.

[Keane and Gross, 1989] Keane, D. and Gross, P. (1989). The midi baton. In *Proceedings of the International Computer Music Conference*, pages 151–154.

[Keane et al., 1990] Keane, D., Smecca, G., and Wood, K. (1990). The midi baton ii. In *Proceedings of the International Computer Music Conference*, page Supplements.

[Keane and Wood, 1991] Keane, D. and Wood, K. (1991). The midi baton iii. In *Proceedings of the International Computer Music Conference*, pages 541–544.

[Keir et al., 2006] Keir, P., Payne, J., Elgoyhen, J., Horner, M., Naef, M., and Anderson, P. (2006). Gesture-recognition with non-referenced tracking. pages 151–158.

[Kela et al., 2006] Kela, J., Korpipää, P., Mäntyjärvi, J., Kallio, S., Savino, G., Jozzo, L., and Marca, D. (2006). Accelerometer-based gesture control for a design environment. *Personal Ubiquitous Comput.*, 10(5):285–299.

[Kolesnik, 2004] Kolesnik, P. (2004). Conducting gesture recognition, analysis and performance system. Master's thesis, McGill University, Montreal.

[Kolesnik and Wanderley,] Kolesnik, P. and Wanderley, M. Recognition, analysis and performance with expressive conducting gestures.

[Krause et al., 2005] Krause, A., Ihmig, M., Rankin, E., Leong, D., Gupta, S., Siewiorek, D., Smailagic, A., Deisher, M., and Sengupta, U. (2005). Trading off prediction accuracy and power consumption for context-aware wearable computing. pages 20–26.

[Krause et al., 2003] Krause, A., Siewiorek, D. P., Smailagic, A., and Farringdon, J. (2003). Unsupervised, dynamic identification of physiological and activity context in wearable computing. pages 88–97.

[Lachenmaier, 2006] Lachenmaier, P. (2006). Entwurf und Implementierung eines Dirigierlernsystems für Kinder.

[Lee and Borchers, 2005] Lee, E. and Borchers, J. (2005). The role of time in engineering computer music systems. In *Proceedings of the 2005 conference on New interfaces for musical expression*, pages 204–207.

[Lee et al., 2006a] Lee, E., Grüll, U., Kiel, H., and Borchers, J. (2006a). conga: A framework for adaptive conducting gesture analysis. In *Proceedings of the 2006 conference on New interfaces for musical expression*, pages 260–265.

[Lee et al., 2006b] Lee, E., Karrer, T., and Borchers, J. (2006b). Toward a framework for interactive systems to conduct digital audio and video streams. *Computer Music Journal*, 30(1):21–36.

[Lee et al., 2006c] Lee, E., Kiel, H., Dedenbach, S., Grüll, Karrer, T., Wolf, M., and Borchers, J. (2006c). isymphony: An adaptive interactive orchestral conducting system for digital audio and video streams. In *CHI '06 extended abstracts on Human factors in computing systems*, pages 259–262.

[Lee et al., 2004] Lee, E., Marrin, T., and Borchers, J. (2004). You're the conductor: A realistic interactive conducting system for children. In *Proceedings of the 2004 conference on New interfaces for musical expression*, pages 68–73.

[Lee et al., 2005] Lee, E., Wolf, M., and Borchers, J. (2005). Improving orchestral conducting systems in public spaces: Examining the temporal characteristics and conceptual models of conducting gestures. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 731–740.

[Lee and Kim, 1999] Lee, H.-K. and Kim, J. H. (1999). An hmm-based threshold model approach for gesture recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(10):961–973.

[Lee et al., 1992a] Lee, M., Freed, A., and Wessel, D. (1992a). Neural networks for simultaneous classification and parameter estimation in musical instrument control. In *Proceedings of SPIE*, volume 1706, pages 244–255.

[Lee et al., 1992b] Lee, M., Garnett, G., and Wessel, D. (1992b). An adaptive conductor follower. In *Proceedings of the International Computer Music Conference*, pages 454–455.

[Lukowicz et al., 2003] Lukowicz, P., Ward, J., Junker, H., Stager, M., Troster, G., Atrash, A., and Starner, T. (2003). Recognizing workshop activity using body worn microphones and accelerometers.

[Mäntyjärvi et al., 2004] Mäntyjärvi, J., Kela, J., Korpipää;, P., and Kallio, S. (2004). Enabling fast and effortless customisation in accelerometer based gesture interaction. In *MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 25–31, New York, NY, USA. ACM Press.

[Mäntylä et al., 2000] Mäntylä, V. M., Mäntyjärvi, J., Seppänen, T., and Tuulari, E. (2000). Hand gesture recognition of a mobile device user. volume 1, pages 281–284 vol.1.

[Marrin, 1996] Marrin, T. (1996). Toward an understanding of musical gesture: Mapping expressive intention with the digital baton. Master's thesis, Massachusetts Institute of Technology.

[Marrin, 1997] Marrin, T. (1997). Possibilities for the digital baton as a general-purpose gestural interface. In *CHI '97 extended abstracts on Human factors in computing systems*, pages 311–312.

[Marrin, 2000] Marrin, T. (2000). *Inside the "Conductor's Jacket": Analysis, Interpretation and Musical Synthesis of Expressive Gesture*. PhD thesis, Massachusetts Institute of Technology.

[Marrin and Paradiso, 1997] Marrin, T. and Paradiso, J. (1997). The digital baton: a versatile performance instrument. In *Proceedings of the International Computer Music Conference*, pages 313–316.

[Marrin and Picard, 1998] Marrin, T. and Picard, R. (1998). The "conductor's jacket": A device for recording expressive musical gestures. In *Proceedings of the International Computer Music Conference*, pages 215–219.

[Mathews, 1976] Mathews, M. V. (1976). The conductor program. In *Proceedings of the International Computer Music Conference*.

[Mathews, 1979] Mathews, M. V. (1979). Sequential drum. *The Journal of the Acoustical Society of America*, 66:42+.

[Mathews, 1989] Mathews, M. V. (1989). *The Conductor Program and Mechanical Baton*, pages 263–281. MIT Press.

[Mathews, 1991] Mathews, M. V. (1991). The radio baton and conductor program, or: Pitch, the most important and least expressive part of music. *Computer Music Journal*, 15(4):37–46.

[Mathews and Abbott, 1980] Mathews, M. V. and Abbott, C. (1980). The sequential drum. *Computer Music Journal*, 4(4):45–59.

[Mathews and Moore, 1970] Mathews, M. V. and Moore, F. R. (1970). GROOVE - a program to compose, store, and edit functions of time. *Communications of the ACM*, 13(12):715–721.

[MatlabCompiler, 2007] MatlabCompiler (2007). http://www.mathworks.com/access/helpdesk/help/toolbox/compiler/, last checked on 10/15/2007.

[MatlabRequirements, 2007] MatlabRequirements (2007). http://www.mathworks.com/products/matlab/requirements.html, last checked on 10/03/2007.

[Maurer et al., 2006] Maurer, U., Rowe, A., Smailagic, A., and Siewiorek, D. P. (2006). eWatch: A wearable sensor and notification platform. pages 4 pp.+.

[Meyer, 2007] Meyer (2007). http://www.pittsburghyouthsymphony.org/bio_danmeyer.html, last checked on 10/16/2007.

[Min et al., 1997] Min, B.-W., Yoon, H.-S., Soh, J., Yang, Y.-M., and Ejima, T. (1997). Hand gesture recognition using hidden markov models. In *Systems, Man, and Cybernetics, 1997. 'Computational Cybernetics and Simulation'., 1997 IEEE International Conference on*, volume 5, pages 4232–4235 vol.5.

[Minnen et al., 2005] Minnen, D., Starner, T., Ward, J. A., Lukowicz, P., and Troster, G. (2005). Recognizing and discovering human actions from on-body sensor data. pages 1545–1548.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Science/Engineering/-Math.

[Morales, 2007] Morales (2007). http://www.edgewoodsymphony.org/about/director, last checked on 10/16/2007.

[Morguet and Lang, 1998] Morguet, P. and Lang, M. (1998). Spotting dynamic hand gestures in video image sequences using hidden markov models. In *Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on*, pages 193–197 vol.3.

[Morguet and Lang, 1999] Morguet, P. and Lang, M. (1999). Comparison of approaches to continuous hand gesture recognition for a visual dialog system.

[Morita et al., 1991] Morita, H., Hashimoto, S., and Ohteru, S. (1991). A computer music system that follows a human conductor. *Computer*, 24(7):44–53.

[Morita et al., 1989] Morita, H., Ohteru, S., and Hashimoto, S. (1989). Computer music system which follows a human conductor. In *Proceedings of the International Computer Music Conference*, pages 207–210.

[Morita et al., 1990] Morita, H., Watanabe, H., and Harada, T. (1990). Knowledge information processing in conducting computer music performer. In *Proceedings of the International Computer Music Conference*, pages 332–334.

[Murphy, 2004] Murphy, D. (2004). Live interpretation of conductors' beat patterns. In *Proceedings of the 13th Danish Conference on Pattern Recognition and Image Analysis*, pages 111–120.

[Murphy et al., 2004] Murphy, D., Andersen, T. H., and Jensen, K. (2004). *Conducting Audio Files via Computer Vision*, pages 529–540. Springer Berlin / Heidelberg.

[Netlab, 2007] Netlab (2007). http://www.ncrg.aston.ac.uk/netlab/, last checked on 10/04/2007.

[Paradiso, 1999] Paradiso, J. A. (1999). The brain opera technology: New instruments and gestural sensors for musical interaction and performance. *Journal of New Music Research*, 28(2):130–149.

[Pylvänäinen, 2005] Pylvänäinen, T. (2005). *Accelerometer Based Gesture Recognition Using Continuous HMMs*.

[Rabiner and Juang, 1986] Rabiner, L. and Juang, B. (1986). An introduction to hidden markov models. *ASSP Magazine, IEEE [see also IEEE Signal Processing Magazine]*, 3(1):4–16.

[Rabiner, 1989] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

[Rich, 1996] Rich, R. (1996). Buchla lightning ii. *Electronic Musician*, 12(8):118–122.

[Rigoll et al., 1998] Rigoll, G., Kosmala, A., and Eickeler, S. (1998). High performance real-time gesture recognition using hidden markov models. pages 69+.

[Rudolf, 1950] Rudolf, M. (1950). *The Grammar of Conducting - A Practical Study of Modern Baton Technique*. G. Schirmer, New York.

[Sawada and Hashimoto, 1997] Sawada, H. and Hashimoto, S. (1997). Gesture recognition using an acceleration sensor and its application to musical performance control. *Electronics and Communications in Japan, Part III*, 80(5):9–17.

[Sawada et al., 1995] Sawada, H., Ohkura, S., and Hashimoto, S. (1995). Gesture analysis using 3d acceleration sensor for music control. In *Proceedings of the International Computer Music Conference*, pages 257–260.

[Schertenleib et al., 2004] Schertenleib, S., Gutierrez, M., Vexo, F., and Thalmann, D. (2004). Conducting a virtual orchestra. *IEEE Multimedia*, 11(3):40–49.

[Schmidt et al., 2007] Schmidt, D., Dannenberg, R. B., Smailagic, A., Siewiorek, D. P., and Brügge, B. (2007). Learning an orchestra conductor's technique using a wearable sensor platform. In *Wearable Computers, 2007 11th IEEE International Symposium on*, pages 113–114.

[Segen et al., 2000] Segen, J., Gluckman, J., and Kumar, S. (2000). Visual interface for conducting virtual orchestra. In *Proceedings of the 15th International Conference on Pattern Recognition*, pages 276–279.

[Smailagic et al., 2005] Smailagic, A., Siewiorek, D. P., Maurer, U., Rowe, A., and Tang, K. P. (2005). eWatch: Context sensitive system design case study. pages 98–103.

[Starner et al., 1997] Starner, Weaver, J., and Pentland, A. (1997). A wearable computing based american sign language recognizer.

[Starner et al., 1994] Starner, T., Makhoul, J., Schwartz, R., and Chou, G. (1994). On-line cursive handwriting recognition using speech recognition methods. In *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*, volume v, pages V/125–V/128 vol.5.

[Starner and Pentland, 1995] Starner, T. and Pentland, A. (1995). Real-time american sign language recognition from video using hidden markov models. In *SCV95*.

[Takala, 1997] Takala, T. (1997). Virtual orchestra performance. In *SIGGRAPH '97: ACM SIGGRAPH 97 Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '97*, pages 81–81.

[Tobey, 1995] Tobey, F. (1995). The ensemble member and the conducted computer. In *Proceedings of the International Computer Music Conference*, pages 529–530.

[Tobey and Fujinaga, 1996] Tobey, F. and Fujinaga (1996). Extraction of conducting gestures in 3d space. In *Proceedings of the International Computer Music Conference*, pages 305–307.

[Usa and Mochida, 1998a] Usa, S. and Mochida, Y. (1998a). A conducting recognition system on the model of musicians' process. *Journal of the Acoustical Society of Japan*, pages 275–287.

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

[Wilson and Bobick, 1999] Wilson, A. D. and Bobick, A. F. (1999). Parametric hidden markov models for gesture recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(9):884–900.

[Yamato et al., 1992] Yamato, J., Ohya, J., and Ishii, K. (1992). Recognizing human action in time-sequential images using hidden markov model. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR '92., 1992 IEEE Computer Society Conference on*, pages 379–385.

[Yang and Xu, 1994] Yang, J. and Xu, Y. (1994). *Hidden Markov Model for Gesture Recognition*.