# Conducting Gesture Recognition, Analysis and Performance System

Paul Kolesnik, Dept. of Music Technology, Faculty of Music
McGill University, Montreal
June 19, 2004

A thesis submitted to McGill University in partial fulfillment
of the requirements of the degree of Master of Arts

*Your file Votre référence*
*ISBN: 0-494-06516-8*
*Our file Notre référence*
*ISBN: 0-494-06516-8*

# Abstract

A number of conducting gesture analysis and performance systems have been developed over the years. However, most of the previous projects either primarily concentrated on tracking tempo and amplitude indicating gestures, or implemented individual mapping techniques for expressive gestures that varied from research to research. There is a clear need for a uniform process that could be applied toward analysis of both indicative and expressive gestures. The proposed system provides a set of tools that contain extensive functionality for identification, classification and performance with conducting gestures. Gesture recognition procedure is designed on the basis of Hidden Markov Model (HMM) process. A set of HMM tools are developed for Max/MSP software. Training and recognition procedures are applied toward both right hand beat- and amplitude- indicative gestures, and left hand expressive gestures. Continuous recognition of right-hand gestures is incorporated into a real-time gesture analysis and performance system in Eyesweb and Max/MSP/Jitter environments.

Un nombre de systèmes d'analyse et d'exécution avec des gestes d'un chef

d'orchestre ont été développés au cours des annes. Pourtant, la plupart de projets précédents se sont principalement concentrés sur des gestes indiquant le tempo et l'amplitude, ou ont mis en application des techniques individuelles pour les gestes expressifs qui ont changé de la recherche à la recherche. Il y a un besoin clair d'une procedure uniforme qui pourrait être appliqué vers l'analyse des gestes indicatifs et expressifs. Le système proposé fournit un ensemble d'outils qui contiennent la fonctionnalité étendue pour l'identification, la classification et l'exécution avec des gestes d'un chef d'orchestre. La procedure d'identification de gestes est concu sur la base du processus du Modèle Caché de Markov (HMM). Un ensemble d'outils de HMM sont développés pour le logiciel de Max/MSP. Des procédures de formation et d'identification sont appliqués vers les gestes indicatifs de main droit aussi que vers les gestes expressifs de main gauche. L'identification continuelle des gestes droits est incorporée à l'analyse de gestes et à un système d'exécution en temps réel dans des environnements d'Eyesweb et de Max/MSP/Jitter.

# Acknowledgements

I would like to express my gratitude to:

Marcelo Wanderley, the supervisor of the thesis, for his constant support and encouragement, for guiding the research in the right direction and for much appreciated advice during the preparation of the thesis.

Philippe Depalle, for providing me with valuable help in some of the technical aspects of the project.

Teresa Marrin, Declan Murphy and Tommi Ilmonen for allowing me to include the photographs of their systems in my thesis.

Alain Terriault, for letting me use the office facilities for studying and research, and for allowing me to have an ideal balance of work and research during the past two years .

To Madhumita Banerjee and Yuri Filipov for their friendship, support and for encouraging me to continue with the project.

To my mother, Lana Lysogor, for her belief in me, for contributing her conducting expertise to the projects, for enduring the long hours of numerous recordings, and for not giving up on the project during the initial experimentation stages.

To my grandmother Valentyna, my brothers Levko and Christian for always being there for me and for helping me in every possible way.

To Veronica for being so caring, understanding and patient with me, and for reminding me of what is truly important in life.

# Contents

# Chapter 1

# Introduction

The role of gestures in human-computer interaction is a fascinating area of research, which demonstrates that much of the communicative intent comes through ancillary movements that either coexist with language (McNeill 1992), or even replace it completely, as in the case of sign language (Wachsmuth and Frohlich 1998). In music, the best known and most researched gestural communication mode is conducting.

Conducting can be viewed as a way of controlling high-level aspects of performance of multiple instruments with one's physical gestures but without direct contact with the instruments themselves. In a conductor-musician interactive environment, visual information perceived by musicians serves as the means of conveying the musical gestures that are created by the conductors physical gestures.

In its nature, conducting is a highly individualistic form of art that offers a broad range of expressiveness, similarly to and arguably even more extensive than the expressive possibilities of playing a musical instrument. A talented conductor uses his or her entire body to convey the directions

for the orchestra. Whereas it is impossible to identify and classify all of the individualistic elements responsible for the effect of an expressive conducting performance, the traditional school of orchestral conducting has developed a well-defined and structured grammar of basic conducting gestures that is shared by majority of professional conductors and taught to student conductors. A good description of basic conducting technique can be found in (Rudolph 1994).

## 1.1 Expressive Conducting Gestures

The two principal functions of an orchestral conductor are to indicate the timing information for the beats of the score in order to synchronize the performance of the musical instruments, and to provide the gestures to indicate his or her artistic interpretation of the performance. The second function introduces a degree of variation and personal interpretation in the musical performance, and is represented by a number of gestures with a high degree of expressivity. Those expressive gestures provide an interesting area for research and are the main topic of this thesis.

## 1.2 Classification of Conducting Gestures

The technique of traditional conducting provides an excellent classification of conducting gestures. The proposed classification was derived from several well-known sources dealing with basic conducting methodology (Long 1971) (Malko 1950) (Ross 1976) (Rudolph 1994).

### 1.2.1 Classification by type

The most natural way of classification of conducting gestures is to group them by their intended effect on the performance. The basic grammar of conducting can be represented through the following hierarchy of elements:

Gesture

- Time-Beating
  - Regular beat
    - *Neutral Legato*
    - *Expressive Legato*
    - *Light Staccato*
    - *Full Staccato*
    - *Marcato*
    - *Tenuto*
  - Subdivided beat
  - Beat Transitions
- Expressive / Artistic
  - Dynamics
    - *Crescendo*
    - *Diminuendo*
    - *Sforzando*
  - Phrasing
    - *Sustained notes*

- *Melodic line*

– Sound Extraction

- *Staccato*

- *Tenuto*

– Cues (Entrances)

– Releases/Cutoffs and Rests

– Syncopation/Accent

– Holds (Fermata)

- Combinations of gestures (consecutive)

- Combinations of gestures (simultaneous)

- Individual/free technique expressive gestures

### 1.2.2 Classification by localization

The hierarchy of gestures shows that the greatest division of conducting gestures exists between time-beating gestures, almost always performed exclusively with the right hand, and expressive artistic gestures, many of which are performed with the left hand. However, the division of gesture execution between right and left hands is not that straightforward.

Whereas it is a common mistake of inexperienced conductors to use both hands for time-beating gestures, professional conductors perform tempo indications exclusively with the right hand—with rare exceptions, such as when a very large ensemble is conducted, time-beating gestures might be performed with both arms in parallel, or in an extremely unlikely case when the right hand is responsible for a very complex expressive indication, the left hand

might take over the time-beating for a short period of time. For practical purposes, it can be assumed that tempo gestures are always performed either by the right hand, or both hands where the left hand doubles the movements of the right hand, and therefore right hand following alone is sufficient to extract the time-beating information at all times.

Whereas time-indicating gestures are essentially of continuous and periodic nature, one gesture following another and transition points indicating the beat arrivals, expressive gestures can be characterized as possibly periodic or non-perioodic, with their occurence depending on conductor's interpretation and position in the musical score. Left hand is largely responsible for expressive gestures—however, right hand contributes to their performance as well.

Very often, combinations of gestures occur in both hands. In one possible case, two or more expressive gestures performed by the same hand are combined into a sequence of movements, which in itself can be viewed as a *complex expressive gesture*—such as a *diminuendo-cutoff* or *sforzando* gestures in the left hand. Another possible combination of gestures takes place when an expressive gesture is incorporated into another (often a time-beating) gesture, as in the case of indications of variations in dynamics and expressive styles that occur simultaneously with tempo indications in the right hand.

Artistic expressive gestures can be conveyed through the movement of the right hand, left hand, or auxiliary indications, such as head movement, facial expression, eye direction, breathing and posture. Whereas in many previous works an effort has been made to track those auxiliary elements and gestures performed with a baton, the current work is focused on recognition

of expressive gestures with right and left hands without a baton. According to (Rudolph 1994), using or not using a baton for orchestral conducting are both acceptable practices, and some orchestral conductors (such as Pierre Boulez, for example) do not use a baton during their performance. Whereas the use of baton contributes to more clarity and precision in beat-indicating gestures, not using the baton allows for a more active participation of the right hand in expressive gesture indications. Since this thesis focuses on recognition of expressive gestures, and due to technical limitations of input devices used in the project, it was chosen not to track the baton.

## 1.3 Computer-based conducting recognition systems

Over the years, researchers proposed a number of computer-based systems to understand the characteristics of conducting gestures. However, design of identification and recognition procedures for a range of expressive gestures has been one of the main issues in the field of computer-based conducting gesture recognition. There is a significant gap between the amount of expressive gestures produced by a conductor to control an orchestra, and the amount of gestural data identifiable and recognizable by computer systems. Most of the early designed systems concentrated on extraction of the right hand or baton-in-right-hand gestures—temporal beat transition points to control the playback of a prerecorded score and the amplitude of movement to control the playback volume—while not taking into account a wide range of expressive gestures produced by the left and right hands which contains meaningful information for the orchestra. Those of the later systems that did

implement identification and recognition procedures based on an extensive set of expressive articulation parameters used individual mapping techniques which varied from research to research. A uniform procedure is needed that would be able to identify and recognize not only beat-tracking and amplitude gestures, but also both right and left hand expressive gestures using a high level recognition technique.

Similarly to an orchestral musician who is expected to know the meaning of conducting gestures in the context of both general technique and idiosyncratic style of the conductor in order to recognize and apply them to a musical score, any computer gesture recognition system is required to contain some knowledge of the gestures it is expected to identify. More specifically, such a system should include three basic elements—a vocabulary of gestures to be recognized, a procedure to identify transitions between gestures from an incoming stream of data retrieved from the conductors movements, and a set of rules that can be used by the system to recognize the gestures.

The main interest of using computer recognition for conducting gestures involves gaining a better understanding of dynamics of conducting gestures in terms of general conducting technique, different schools of conducting and individual conducting styles. Applications of conducting gesture recognition include development of educational programs for student and professional conductors, conducting performance systems, and research tools in the area of analysis and classifications of conducting gestures.

## 1.4 Thesis Overview

This thesis is based on the development of a Conducting Gesture Analysis, Performance and Recognition System. Gesture Analysis refers to the component of the system that deals with processing of the input information in order to track the movement of the right and left hands, extract their respective positional data streams, and identify elementary right-hand gesture elements—such as beat transition points and amplitude of the beat-indicating movements. Gesture Performance refers to the part of the system that maps the elementary features to changes in output of the prerecorded audio (and optionally, video) score during a realtime interactive performance with the system. Gesture Recognition, which is the main area of research presented in the thesis, is the component of the system that uses high-level training and recognition procedures to process the positional information extracted with Gesture Analysis, and provides a set of positional research tools that are used for recognition of characteristic features of left- and right-hand expressive conducting gestures.

Chapter 2 of the thesis will provide a chronological overview of up-to-date research done in the area of conducting recognition and classify the developed systems in terms of their design and performance characteristics. Chapter 3 will focus on the implementation of the Gesture Analysis and Performance components of the system. Chapter 4 will discuss the application of Hidden Markov Model (HMM) in Max/MSP environment as a Gesture Recognition element of the system. Chapter 5 will describe the research based on recognition of expressive conducting gestures that was carried out with the system. Chapter 6 will discuss results of the experiments and will summa-

rize the advancements presented in the thesis. Appendix A will present a general overview of the Hidden Markov Model techniques that were used in the project.

# Chapter 2

# Overview of Previous Systems

Many performance, educational and research systems have been designed over the years in the areas of computer-based conducting gesture recognition. Those systems experimented with a number of different approaches towards conducting gesture analysis, recognition and mapping to music synthesis.

## 2.1 Early Systems

### Groove system

The first system that implemented a user-controlled realtime music synthesis was the GROOVE (General Real-time Output Operations on Voltage-controlled Equipment) system, designed by Mathews and Moore in 1970 (Mathews and Moore 1970). The system used a 24-note keyboard, four rotary knobs, and 3-D joystick as input devices to control a synthesizer that generated sounds through an interface for analog devices and 14 DAC convertors. The GROOVE was the first project to introduce the idea of the expressive control of a user over the performance of a computer music synthesis program, in a way similar to a conductor having a degree of expressive

control over performance of the orchestra.

### Conductor Program

In 1976, Mathews implemented a *Conductor Program* (Mathews 1976). The application was based on the GROOVE system, and was designed to improve user control over computer performance, which introduced a higher degree of expressivity and improvisation in music synthesis. Articulation and other expressive effects were stored in a score file, and could be applied to music during realtime performance.

### Microcomputer-based Conducting System

In 1980, Buxton et al. designed a computer-based conductor following performance system (Buxton, Reeves, Fedorkov, Smith, and Baecker 1980). The system tracked the 2-dimensional position of a cursor (a mouse-type input device) on a tablet and input from buttons on the cursor, as well as external switches and sliders. Tracked parameters included pitch shift, tempo, amplitude, timbre and articulation parameters. Preprocessed score, which consisted of sets of instructions for the synthesizer, was used to produce the output of the system. Similarly to the *Groove* and *Conductor Program* projects, this system was not based on recognition of actual conducting gestures, but rather worked with simulation of those gestures in 2-dimensional space using a number of input devices (tablet, cursor, switches and slides).

## 2.2 MIDI-based systems

### Conductor Follower

Following Buxtons research, a *Conductor Follower* system was designed by Haflich and Burns in 1983 (Haflich and Burns 1983). The system used Polaroid ultrasonic rangefinder units and a wand-shaped device with a corner reflector on its tip to produce reflections that were analyzed by a computer. Position of the wand device was tracked in 2D space and used to extract beat points. The analyzed information was applied to control tempo and dynamics of the synthesized sound. The significance of the *Conductor Follower* project was that it was the first system to extract and analyze a range of real conducting gestures (and not their simulations) in space—unlike previous systems that used such input devices as a joystick and knobs (Mathews and Moore 1970) or a tablet (Buxton, Reeves, Fedorkov, Smith, and Baecker 1980), this system used a wand-shaped device which was similar to an actual orchestral conducting baton.

### Mechanical Baton and Radio Baton

In 1989, Max Matthews designed a device called *Mechanical Baton* (Mathews 1989). The baton, called a *Daton*, hit a metal plate which sent positional information to a PC Intel computer with a Roland 401 MIDI card. The computer made necessary corrections to a prerecorded pitch/duration score and sent out MIDI information to Yamaha synthesizer. The score variables that were affected by incoming information were tempo, loudness and balance of voices. In 1991, *Daton* was improved into a *Radio Baton* system, which used two batons that were moved above a metal plate (Mathews 1991). Positional

input data used by the system was determined by radio frequency signals emitted from the batons.

### MIDI Baton System

In 1989, a *MIDI Baton* system was developed by Keane and Gross (Keane and Gross 1989). The system used a baton controller mechanism and a footswitch as input devices to control a sequencer that would generate the sound. The baton contained a metal ball attached to a spring wire inside a brass tube. Change in acceleration of the baton caused the contact between the ball and the tube, which in its turn created an electrical signal. Beat information was extracted directly from electrical signals with some minor adjustments. The footswitch was used to send *start/pause/restart* commands to the sequencer. An important contribution of the project was development of system control over lead/lag time between the conductor and the system, degree of response to tempo variations, and minimum time between triggering messages to the sequencer. Later advancements of the project included a *MIDI Baton II* (Keane, Smecca, and Wood 1990) and *MIDI Baton III* (Keane and Wood 1991) systems.

### Computer Music System and Gesticulation System

Also in 1989, Morita et al. designed a *Computer Music System that Follows a Human Conductor*, the first project to use a CCD camera as an input device (Morita, Otheru, and Hashimoto 1989). Feature extraction hardware used with the camera followed a white glove or a baton marker of a conductors right hand. The system allowed for tempo and intensity control of a pre-recorded MIDI score. In 1990, the baton-motion understanding system was

combined with a gesture recognition system that tracked trajectory, velocity and acceleration information of a conductor's left hand with the *Dataglove* input device (Morita, Otheru, and Hashimoto 1990). The collected information was mapped to control dynamics and articulation effects in the score. Another important component of the system was a self-evaluation system that recorded results for next performance, which was a first attempt to use a learning algorithm based on collected data in a conductor gesture recognition device.

### Light Baton

Another device that used a CCD camera, called *Light Baton*, was designed in 1992 by Bertini and Carosi (Bertini and Carosi 1992). The system was primarily intended to be used for synchronization of live musical performances with prerecorded MIDI scores on a computer, with conductor's gestures serving as a connection between the human performer and the computer. A special conducting baton with a lamp on its tip was used to send light signals to the CCD camera. The light position was analyzed by an image acquisition board, and the playback of the prerecorded score was adjusted through control of tempo and amplitude.

### Adaptive Conductor Follower

A system called *Adaptive Conductor Follower* was developed in 1992 by Lee et al. (Lee, Garnett, and Wessel 1992), and was expanded into a *Conductor Follower* system by Brecht and Garnett in 1995 (Brecht and Garnett 1995). The system used Buchla Lightning baton and Mattel Power Glove to collect positional information, which was then processed by classification and esti-

mation algorithms in Max environment. Three possible evaluation methods were implemented at the beat analysis stage—a simple historical updating algorithm based on previous beat information, a more detailed updating algorithm with 6-point position update for each beat, and an algorithm based on neural networks which used 6-point position probability evaluation algorithm. Tempo and dynamics of the prerecorded MIDI score were modified based on the input information. One of the most important achievements of the system was that it produced the first successful attempt to use Artificial Neural Networks for recognition purposes.

### Ensemble Member and Conducted Computer / Extraction of Conducting Gestures in 3D space

In 1996, Tobey and Fujinaga designed a conductor follower system that used two Buchla Lightning batons in Max environment (Tobey and Fujinaga 1996). The system was based on Tobey's previous research (Tobey 1995) and was the first system to collect and analyze positional information in 3D space. Its capabilities included tempo (rubato) control, dynamics control, beat pattern recognition, beat style recognition, accentuation control and timbral balances.

### Digital Baton

Also in 1996, a *Digital Baton* system was implemented by Marrin and Paradiso (Marrin and Paradiso 1997). The handle of the *Digital Baton* input device contained pressure and acceleration sensors, and the tip of the baton held an infrared LED which was tracked by a camera with a position-sensitive photodiode. Processed information included adjustments of tempo, dynam-

ics and articulation effects for the prerecorded score. An in-depth analysis of conducting gestures was integrated in the design of the system.

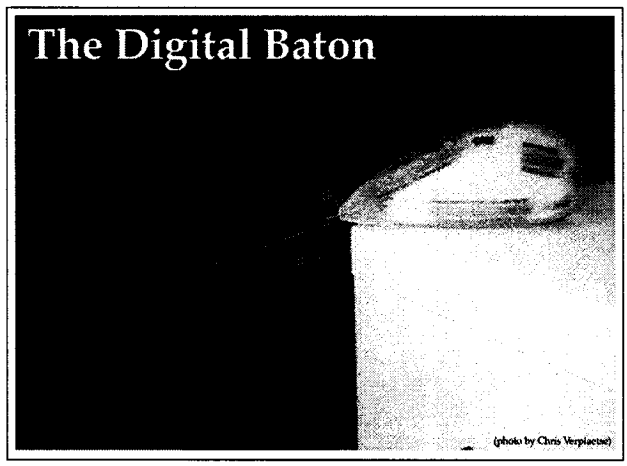


Figure 2.1: A picture of the *Digital Baton* device.

### Multi-Modal Conducting Simulator

A device called *Multi-Modal Conducting Simulator* was designed in 1998 by Usa and Mochida (Usa and Mochida 1998a), (Usa and Mochida 1998b). The system used two 2D acceleration sensors, an eye camera and a still video image to collect the 'cue-in' information, and a breathing sensor to control dynamics at beginnings of musical phrases. The *Conducting Simulator* was the first system to use Hidden Markov Models, a high-level statistical observation analysis tool, to determine right hand conducting gesture patterns, as well as the first system to implement eye tracking as one of the input indications.

### Conductor's Jacket

In 1998, Marrin and Picard created the *Conductor's Jacket* system that took on an alternative approach towards the source of gestural input information (Marrin and Picard 1998), (Marrin 2000). Whereas all of the previous researchers in the field had been mainly concerned with tracking positional coordinates of the conductor's hands, Marrin and Pickard constructed a system to analyze muscle tension response as the main indication of intended gestural messages.

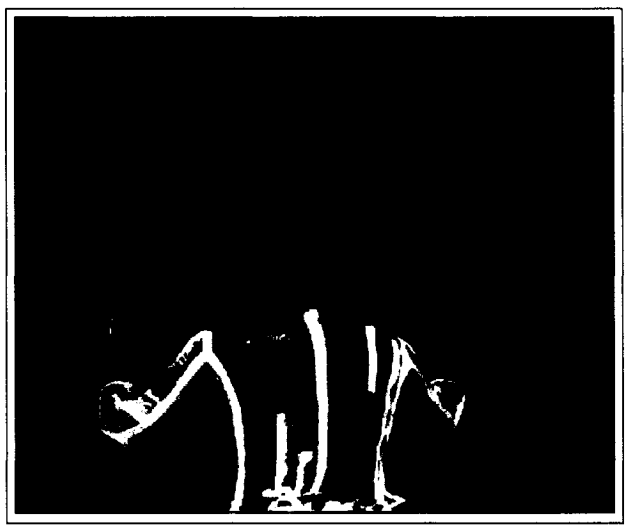

Figure 2.2: Keith Lockhart conducting an orchestra with the *Conductor's Jacket* system (photograph by Rich Fletcher).

The *Conductor's Jacket* consisted of four muscle tension electromyogram (EMG) sensors, respiration monitor, heart rate monitor, temperature sensor and Galvanic skin response sensor. The input information was passed on to two networked computers analyzing system. The processed MIDI informa-

tion was then transferred to MIDI-controllable sound production equipment. The variables that were adjusted by the captured gestural information were volume, tempo, balance, accents, dynamics and a number of articulation effects.

### Conductor Following with Artificial Neural Networks

In 1999, Ilmonen and Takala used Artificial Neural Networks scheme to create a conducting recognition system (Ilmonen and Takala 1999). A data suit with FastTrack motion tracking devices was used to collect positional information—which was the 1st time that high-precision 3D motion tracking device was implemented in a conductor follower system.

Figure 2.3: Tommi Ilmonen conducting with the *Virtual Orchestra* system

The system was designed based on a modular approach, with three basic

modules responsible for their corresponding layers: input, analyzing software and synthesizing software. Tempo and articulation were the main variables affected by the input information. In addition, the system used a computer-generated 3D graphical image of an orchestra controlled by the user's gestures, which was developed on the basis of the Virtual Orchestra system by DIVA (Digital Virtual Acoustics Group) (Takala 1997).

### Virtual Dance and Music

The *Virtual Dance and Music* system was designed in 2000 by Segen, Majumder and Gluckman (Segen, Mujumder, and Gluckman 2000). The system contained three main components: a gesture recognition system which extracted the tempo information by using two synchronized cameras, a dance sequencer which adjusted sets of video frames to gestural information, and a music sequencer which was represented by a MIDI synthesizer with tempo control. The main accent was put on prediction of tempo factor in the MIDI score using the polynomials, and output produced by the system consisted of synchronized dance and music. The *Virtual Dance and Music* system was performance-oriented and focused primarily on synchronization of dance and media through elementary tempo-indicating movement rather than on recognition and analysis of a wide range of conducting gestures.

## 2.3 Audio-based systems

### Personal Orchestra

In 2002, Borchers et al. designed a system called *Personal Orchestra* (Borchers, Samminger, and Muhlhauser 2002). Similarly to earlier designs of conduc-

tor following devices, the main control variables of the device were tempo, volume and instrumentation. However, one of the main innovations of the system was that it used prerecorded audio (and video) material from a real life performance by Vienna Philarmonic Orchestra, as opposed to a computer-generated MIDI score used in previous systems. Whereas this contributed to a more realistic sounding output, it also created a number of problems at design stage due to complexity of audio compression-expansion in real time. A solution to the problem included pitch-shifting the audio file at a range of intervals (plus-minus an octave) prior to the use of the system and playing the resulting audio files at their corresponding speeds to conserve the pitch continuity. For example, a file that resulted in transposing the original file up an octave, or doubled frequency, would be played at half speed to produce the same pitch components as the original audio file, A dynamic crossfade between the files was then implemented to produce audio output at desired speeds. Video time stretching did not produce similar problems since it could be easily implemented by simply repeating or dropping video frames.

Buchla Lightning sticks were the input device used by the system that transmitted 2-dimensional coordinates to a movement analysis program (written in Java) which interpreted the maximum downward coordinates of the right hand as beat indicators and vertical coordinates of the left hand as amplitude indicators. The program then applied changes to control variables, and sent audio output to the speakers and video output to a projector. Since the system was designed for an exhibition in Vienna Music Museum, it also included a general graphical user interface with a video content intended to make user-computer interaction process more natural.

Personal Orchestra Project was not intended for being used by a conductor—rather, similarly to the *Virtual Dance and Music* project, it was performance-oriented and designed as an interactive exhibit for a general user with little or no conducting experience. Therefore, it did not contain the complexity of gesture recognition functionality introduced by some of the earlier systems. However, its audio stretching algorithm was used in later systems which made a significant contribution to research in conducting gesture recognition.

### Conducting Audio Files via Computer Vision

In 2003, Murphy et al. presented a conducting gesture recognition system which was able to control the tempo of an audio file playback through standard conducting movements (Murphy, Andersen, and Jensen 2003). The system incorporated three major components – gesture tracking, audio beat estimation and audio time scaling.

Gesture tracking was done by using a set of Murphy's *Computer Vision* techniques. A direct view camera (by itself or in combination with a profile view camera) was used to track either the conductor's baton or his/her right hand. Baton tracking involved a more complex tracking technique with seek/track modes whereas hand tracking was done with a more straightforward Lucas-Kanade feature tracking algorithm (included in the motion capture library of the Eyesweb software used by the system (Camurri, Coletta, Peri, Ricchetti, Ricci, Trocca, and Volpe 2000)). Recognition of conductor's beat indications was performed based on visual input, and time-stamped MIDI messages were sent to indicate the points of beat occurances.

Audio beat estimation was done through extracting parameters from a

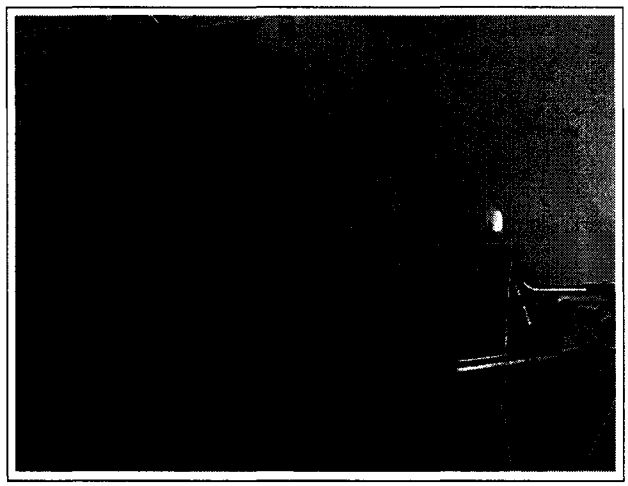

Figure 2.4: Declan Murphy with the *Computer Vision* system

selected audio file. In particular, HFC (high frequency component) parameter, calculated as sum of the high frequency spectral magnitude above 4KHz weighted by the frequencies squared, was extracted. Since HFC data by itself was too noisy to be used directly for beat detection, it was passed on to a beat probability vector algorithm which was modeled based on an assumption that a note onset is more likely to occur at a time interval roughly equal to the last time interval between previous note onsets (or equal to a multiple of that interval).

Two alternative methods were implemented for audio time scaling, based on the choice of the beat coupling method to be used. The first method, known as the event based approach, worked on the assumption of the low system latency and attempted to synchronize the beats at highest interval rates. In that case, audio information was either cut or extended in order to compensate for the beat interval change. Another method, which was

designed for conditions of larger latency, is known as a delayed approach. It attempted to synchronize the audio playback with conductor's indications not for the current, but rather for the subsequent beat. This results in a "lagging" effect at stages when a conductor attempts to change the tempo, but also gives more time for tempo adjustments. In that situation, a phase vocoder was used instead of cutting or repeating audio segments, as in the *Personal Orchestra* system. This was the first time real-time audio manipulation using vocoder techniques was implemented in a conducting gesture recognition system, which resulted in a better sound quality. The time scaling calculation algorithm that was used for the delayed approach was similar to the one used in the *Personal Orchestra* system (Borchers, Samminger, and Muhlhauser 2002).

## 2.4 Overview Summary Table

This section presents a summarized description of the system described in this chapter in table format.

| Year | Name | Authors | Input Device | Tracked Parameters | Tracking Rate / Range | Ctl Vars | Output | Software/Hardware | Purpose / Note |
|---|---|---|---|---|---|---|---|---|---|
| 1980 | Microcomputer-based Conducting System | Buxton | - | - | - | - | Prerecorded MIDI score | - | research |
| 1983 | Conductor Follower | Haflich, Burns | Two ultrasonic rangefinder units (used in Polaroid cameras), wand device | Arm position in 2d space | 10-20 Hz, 5-foot range | tempo, dynamics | Prerecorded MIDI score | - | research |
| 1989 | Mechanical Baton | Matthews | Daton, joystick, knobs | Daton position when hitting a metal plate (2d space), User input through knobs | Limited by metal plate | tempo, dynamics, voice balance | Prerecorded MIDI score (triggers) | A/D data translation card, Roland MIDI Card, Yamaha 816 synth | research / 1st effective baton conducting system for computers |
| 1991 | Radio Baton | Matthews | 2 radio frequency batons joystick, knobs | Position of the two batons above metal plate (2d) | Space above metal plate | tempo, dynamics, voice balance | Prerecorded MIDI score (triggers) | A/D data translation card, Roland MIDI Card, Yamaha 816 synth | Research / more precise than Mechanical Baton |
| 1989 | MIDI Baton System | Keane, Gross | Baton controller (metal ball attached to a spring wire inside a brass tube), footswitch | Acceleration of the baton controller Footswitch signal to start/stop score | 0.83 Khz | tempo | Prerecorded MIDI score | Garfield Time Commander (synchronization unit), MIDI synth | Research / Oversensitivity problem with the controller (solved by filling the tube with fluid) |
| 1989 | Computer Music System that follows a Human Conductor | Morita, Hashimoto, Otheru | CCD Camera, white glove (right hand) / baton marker | Right hand: Arm / baton position in 2d space | 30 Hz | Tempo, dynamics | Prerecorded MIDI score | Computer vision system, MIDI control unit | Research / First system to use a CCD camera |
| 1991 | Gesticulation System | Morita, Watanabe, | CCD Camera, white glove (right hand) / | Right hand: Arm / baton position in 2d | 30 Hz | Right hand: tempo, | Prerecorded MIDI score | Computer vision system (right hand) | Research / First system to |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Hashimoto, Otheru, Harada | baton marker<br>Dataglove (left hand) with optical-fiber and magnetic position sensors | space<br>Left hand: trajectory, velocity and acceleration | | dynamics<br>Left hand: MPX [Music Performance Expression] – dynamics, effects, etc. | | Gesture Understanding System (left hand)<br>Performance Communication System<br>MIDI control unit | contain a self-evaluation algorithm which recorded results for next performance |
| 1992 | Light Baton | Bertini, Carosi | CCD camera,<br>Baton containing a lamp | Baton position in 2d space | 25 Hz | tempo, dynamics | Prerecorded MIDI score | Turbo Pascal 5.5 software, VISCA Vision Card | Research |
| 1992, / 1995 | Adaptive Conductor Follower Conductor Follower | Lee, Garnett, Wessel / Brecht, Garnett | Buchla Lightning Baton, Mattel Power Glove | Baton position in 2d space | - | tempo, dynamics | Prerecorded MIDI score | Max and Maxnet programming environment, three alternative analysis algorithms | Research /<br>First system to use neural networks for beat analysis |
| 1995/ 1996 | Ensemble Member and Conducted Computer / Extraction of Conducting Gestures in 3D space | Tobey / Tobey, Fujinaga | Two Buchla Lightning Batons | Baton and left hand position in 3d space | - | tempo, dynamics, beat pattern, beat style, accentuation, timbre | Prerecorded MIDI score | - | Research, performance /<br>First system to use 3d tracking |
| 1996 | Digital Baton | Marrin | Digital Baton | Pressure, acceleration, intensity, position sensors; LED light that is captured by external photodiode (2d space) | 1 KHz for position sensors (low intensity),<br>20 Hz for all other sensors | tempo, dynamics, articulation effects | Prerecorded MIDI score | Max patches | Research, performance, educational /<br>System distinguished by a variety of sensors combined in a single baton |
| 1998 | Multi-Modal Conducting Simulator | Usa, Mochida | Eye camera (used together with a still image | Right hand acceleration/position, breath | - | Tempo, dynamics, Articulation | Prerecorded MIDI score | HMM [Hidden Markov Model ] Movement | Research, performance, educational / first |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | of an orchestra), 2 acceleration sensors, breathing sensor, | intensity, eye gaze direction | | effects, phrasing (corresponds to breath intensity) | | Recognition system | system to use HMMs |
| 1999 | Conductor Following with Artificial Neural Networks | Ilmomen, Takala | Datasuit with 6dof sensors (replaced with accelerometers in demo for cost reasons) | 3D positional information for the body | - | Tempo, articulation | Prerecorded MIDI score, 3D graphics (orchestra) | ANN [Artificial Neural Network ] analysis system, FastTrak 6DOF sensor system | Research, performance / First system to use high-precision 6dof positional sensors |
| 1998 / 2000 | Conductor's Jacket / Inside the Conductor's Jacket | Marrin, Picard / Marrin | Jacket: 4 EMG, 1 respiration, 1 heart rate, 1 temperature, 1 Galvanic skin response sensors | Main: muscle tension Secondary: breath intensity, heart rate, skin response | 330 Hz – 4 KHz | Tempo, dynamics, articulation, accents, meter, pulse, vibrato, number of voices, harmonic colorations, etc | Prerecorded MIDI score | Two networked computers analyzing system: *Labview *Visual Dev studio *Rogus MIDI library | Research, performance, educational / No positional sensors, info received through muscle tension, extensive analysis of gestural information |
| 2000 | Virtual Dance and Music Conducted by a Human Conductor | Segen, Majumder, Gluckman | 2 cameras | Right arm positional information (3d space) | 30 fps | tempo | Synchronized prerecorded MIDI and video | Gesture recognition system, dance sequencer and music sequencer | Performance / Primarily designed for synchronization of music and dance during performance |
| 2002 | Personal Orchestra | Borchers, Samminger, Muhlhauser | Buchla Lightning baton | Right arm positional information (2d space) | 30 fps | tempo, dynamics, instrumentation | Synchronized prerecorded audio and video | Java server / client, Quicktime | Research, exhibition, performance / First system to use audio time-stretch algorithm |
| 2003 | Computer Vision | Murphy, Andersen Jensen | Video cameras (direct/profile) | Right arm positional information (3d space) | 30 fps | tempo | Prerecorded audio, 3D graphics(baton) | Computer Vision, EyesWeb, Mixxx | Research |

# Chapter 3

# Gesture Analysis and Performance

This chapter deals with the overall design of the Gesture Analysis and Performance components of the described system. Whereas the Gesture Analysis part of the system is implemented as a link between the user input and processing of extracted positional data stream and is also used together with Gesture Recognition system, the main purpose of the Performance component of the system is mapping of elementary conducting gestures to changes in prerecorded audio/video output during a realtime interactive performance.

## 3.1 Gesture Analysis

Gesture Analysis part of the system was developed as a set of tools to be applied to hand movement tracking, extraction of positional data stream, and identification of beat-indicating transition points and beat amplitude in right-hand gestures.

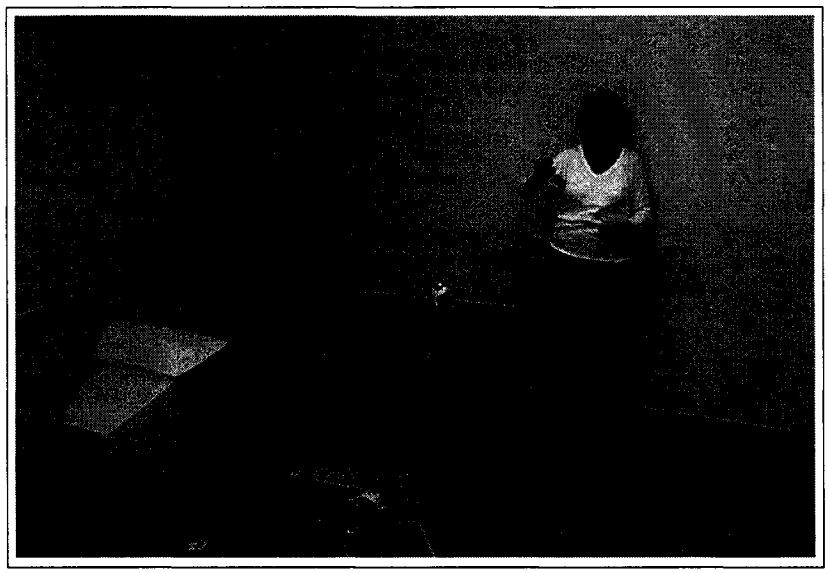

Figure 3.1: Lana Lysogor, a doctoral conducting student, performing with the Gesture Analysis and Performance components of the system.

### 3.1.1 Image input

Gesture image tracking is done with two inexpensive Logitech QuickCam Messenger USB cameras, placed in front and profile view of the user. The cameras are used to follow the image of the user's hand movements, with the user wearing a color glove on the right hand (or/and left hand) to facilitate the tracking process. Whereas for practical reasons the USB camera input was chosen as a low-cost solution, the described system is built to also be compatible with higher precision six-degrees-of-freedom (6DOF) positional trackers for use in further research.

### 3.1.2 Image Processing with Eyesweb Software

The front and profile view images obtained by the cameras are used for feature extraction by Eyesweb, an image processing software that was developed at DIST, University of Genoa, Italy (Camurri et al. 2000) and has been used in numerous interactive multimedia installations, including the *Computer Vision* system (Murphy, Andersen, and Jensen 2003). Image acquisition and processing is handled in the Eyesweb patch using blob colour tracking techniques. At the initialization stage, the intended tracking regions of the color glove is selected by the user clicking with a mouse on the incoming image regions. An internal color blob tracking object in Eyesweb, which is the core of the image processing patch, then extracts two-dimensional positional coordinates of the center of the tracked region and passes their values to Eyesweb network data transfer objects. As feedback to the user, the patch displays the full incoming image from the camera as well as the resulting image that contains the tracked color regions only. The incoming image is also recorded by the patch in a video file, which could be later used for offline processing of the user's gestures.

### 3.1.3 OSC network

The Eyesweb patch sends out its output stream to Max/MSP software via Open Sound Control (OSC) network (Wright 1998). The network provides a connection between the Eyesweb patch installed on a PC computer, and Max/MSP/Jitter software that is run in a Mac OsX environment. The connection between the two computers is established directly through a crossover network cable. Four positional streams, two per hand (horizontal and ver-

tical), are sent out on four separate OSC channels. Eyesweb network OSC objects are used as servers, and Max/MSP external OSC objects are used as clients.

### 3.1.4 Positional Data Filtering

One of the problems associated with the incoming data stream was that the values were coming into the Max/MSP patch from the Eyesweb environment at irregular intervals. A *pk.resample* external object was written to resample those values into a regular flow data stream, according to a specified resampling rate (no value interpolation was implemented). Another object called *pk.velacc* was written to calculate velocity values for horizontal and vertical movement, as well as vector velocity and acceleration of the movement.

In the course of experimenting with the system, it was also discovered that the positional information stream received by the Max patch contained a substantial amount of jitter, caused both by slight errors in hand movement tracking by Eyesweb objects and slight variations in the natural hand movement that are not perceived by the human eye but are nonetheless present in human movment. Therefore, some kind of filtering needed to be implemented in the system. A low pass filtering technique, used for cleaning of the positional data in *Motion Analysis and Mapping to Music* project (Bevilacqua, Ridenour, and Cuccia 2002), was applied to the current problem. The filter, designed through a set of internal objects in Max/MSP, calculates a running average using a number of input points, which results in unwanted jitter data being filtered out of the system. Since the frame rate of the input cameras is 25 fps (40Hz), which is substantially lower than the rate of the Vicon motion

capture system used in the *Motion Analysis and Mapping to Music* project, a 6-point (instead of a 10-point, as in referred project) running average filter was implemented to avoid excessive filtering and latency in the system. The filtered data stream is sent to the beat/amplitude extraction subpatch. Both the filtered and unfiltered data streams are sent as graphic points to four separate displays in the Max patch, and there a clear visible improvement in the case of filtered information.

### 3.1.5 Recording of Positional Information

As an optional feature, a separate Max patch was designed to be responsible for recording of positional data stream. In the record mode, the data is stored in a table file and included time stamp information and horizontal and vertical positional value streams for both hands. In playback mode, the recorded positional information can be played in order to control the performance of the score by itself, without extracting it from the video recordings in the Eyesweb patch.

### 3.1.6 Beat Transition and Amplitude Extraction

One of the main responsibilities of the Gesture Analysis component of the system is to extract beat amplitude and beat transition points from the right-hand conducting gestures based on maxima and minima of their absolute positional values. Those values are extracted from the positional coordinate streams that are received from the Eyesweb patch. Beat transition point extraction was done by a *pk.beatrecognize* external object, which detected a beat every time there was a transition from the downward to the upward

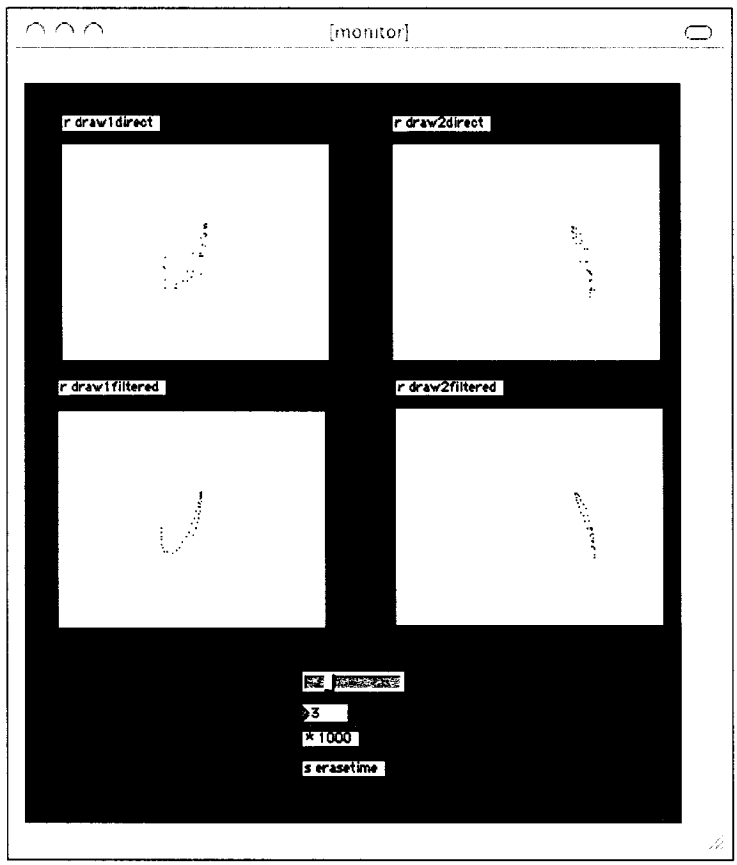


Figure 3.2: Positional data displays in the Gesture Analysis Max/MSP patch: front signal raw (upper left) and filtered (lower left), profile signal raw (upper right) and filtered (lower right).

vertical movement of the right hand.

In order to avoid false beat detections which would occur due to noise in positional data stream caused by minor fluctuations in positional values (that were greatly reduced by the running average filter), a *pk.filter* object was implemented. The object accepted an argument which set a minimal time interval in milliseconds required to occur between consecutive beat detections in order for the detected beat to pass through the object. The time interval value 50 ms was found to be appropriate to most situations and was used in all of the experiments with the system.

At the beginning of the conducting performance, it is natural for a conductor to conduct a subbeat before the music entrance at the first beat. In order to avoid the frequent problem of the system mistaking the subbeat for the first beat, another filter was implemented with a set of internal Max objects. In it, an incoming 'bang' message (signifying a beat detected by the previous objects in the chain) would only be let through if the difference between the maximum vertical position value that occurred during the beat and the very first positional value recorded at the beginning of the beat (which corresponds to the minimum vertical value of the beat) is greater than the threshold value specified by the user. In order words, the beat is only detected if the right hand travels in the upward direction before the direction is changed from downward to upward. This filter successfully eliminates the first subbeat problem.

Right- and left-hand amplitude extraction was implemented with a set of internal Max objects. For right-hand beat amplitude, the main control variable was the vector length of the tracked path of the right hand in space

that would be recorded between consecutive beats. Since different beats vary in form, and therefore also in length of the positional path they travel, each individual beat is only compared to its corresponding beat from the last measure. Each time a beat occurs, the calculated vector path value for the new beat is divided by the path value of the previous beat, and the result is multiplied by the result of the same calculation done for the previous beat (the original value of the result would be set to 1). However, one of the requirements for the system to work is that the four beats of the first conducted measure are be conducted in a uniform fashion in terms of dynamics, since their values would serve as a reference to the beats in all of the consecutive measures. Alternatively, the values for the vector path lengths can be prerecorded prior to the use of the system and be used as reference values.

Left-hand amplitude level is calculated by direct mapping of the vertical value of the left hand on the video screen to the amplitude level of the score, with the option of maximum and minimum volume level values corresponding to the highest and lowest positions of the hand. The new left-hand amplitude value is then averaged with the extracted right-hand amplitude, and the resulting value is sent to the audio output level of the patch.

## 3.2 Gesture Performance

Gesture Performance element of the system is primarily responsible for mapping the identified beat transition points and beat amplitude values to modifications in playback speed and volume of the audio score that is being conducted by the user.

### 3.2.1 Calculation of speed modification

To compute adjustments between user's beat indications and audio playback speed, audio stretching calculation algorithm was used. The developed algorithm was similar to the one first implemented in (Borchers, Samminger, and Muhlhauser 2002) where it was developed on the basis of techniques that were originally used to synchronize computer clocks over the network. It can be summarized in the following procedure:

Step 1. Every time a new beat is detected, calculate the value of the speed with which the user has conducted the last measure:

$$v_1 = \frac{b_s - b_s'}{b_u - b_u'}, \tag{3.1}$$

where $b_s$ is the original position of the beat in the score, $b_s'$ is the original position of the previous beat, $b_u$ is the time of the incoming detected beat, and $b_u'$ is the time of the previous detected beat.

Step 2. On the basis of value of $v_1$, calculate the value of adjustment speed:

$$v_2 = \frac{v_1 \cdot \Delta t}{t_u + \Delta t - b_s}, \tag{3.2}$$

where $\Delta t$ is the user-specified catchup position interval for the actual score position to become synchronized with the desired score position in the future and $t_u$ is the position until which the score was conducted by the user.

Step 3. Throughout the execution, a runtime loop computes the value of $t_u$ as:

$$t_u = t_u' + v_{cur} \cdot i, \tag{3.3}$$

where $t'_u$ is the previous value of $t_u$ before the actual computation, $v_{cur}$ is the current speed (initially set to 1) and $i$ is the interval between the loop executions in ms (in this case, $i$ was equal to 1 ms). If the condition $t_{bu} > \Delta t$ is true ($t_{bu}$ being the position to which the score was conducted at the current speed since the beginning of the current beat), then $v_{cur}$ was set to $v_1$, otherwise $v_{cur} = v_2$. This way, if the synchronization of the position on the score and position of the user's beats occurs before the end of the current beat, the current speed is set to the speed at which the user is conducting, as there is no need for further adjustments until the next detected beat.

In the described system, the calculations were implemented by the external *pk.coexcalc* object. The object accepted *bang* messages in its left inlet and the values of corresponding beat positions on the audio score in the right inlet. It then calculated the time compression / expansion ration and sent it continuously out the left outlet. The current calculated position on the audio score is sent out of the right outlet of the object.

### 3.2.2 Recording of Beat Values

In order to prepare an audio score for performance with the described system, a beat recording patch is used at the preprocessing stage. Beat indications are tapped in by the user during the playback of the audio file, and their relative time values are recorded in a separate table file. During performance, the table values are then supplied to the *pk.coexcalc* object which uses them as reference beat values for tempo adjustment calculation. One of the much needed future improvements of the system would be an implementation of a

realtime beat recognition object.

| Object Name | Component | Stage | Function |
|---|---|---|---|
| pk.resample | Analysis | Input | resample the incoming stream of data according to a user-specified rate |
| pk.diff | Analysis | Data Analysis | calculate running difference between incoming data values |
| pk.velacc | Analysis | Data Analysis | calculate horizontal, vertical and vector velocity and acceleration based on horizontal and vertical position values |
| pk.beatrecognize | Analysis | Beat Tracking | detect beat transition based on direction changes and velocity of vertical positional data |
| pk.filterbangs | Analysis | Data Filter | only allow a single occurance of detected beats through within a user-specified interval |
| pk.coexcalc | Performance | Data Mapping | calculate the current speed required to catch up with position in the audio file based on incoming beat transitions |

Table 3.1: External objects written for Analysis and Performance system components.

### 3.2.3 Video output

As an optional feature, a prerecorded video score can be used as visual feedback to the user simultaneously with the corresponding audio score. Video score tempo modification is done by internal Jitter environment objects. Since video stretching/ compression can be easily done by dropping/repeating the frames without visible artifacts (given that the frame

rate is high enough), there were no problems associated with its implementation in the system. A video (and audio) recording of McGill Symphony orchestra rehearsal[1] was used with the system. Since the main goal of the system is directed towards recognition of expressive gestures, video playback is not part of the final system implementation. However, it has been developed to be used as feedback to the user and the audience during interactive performances with the system—it could be displayed through an overhead projector or a monitor on the wall in front of the user to simulate the presence of an orchestra.

### 3.2.4 Time Compression/Expansion Using the Phase Vocoder

The resulting time compression/expansion value was sent to audio/video output patch. For actual audio adjustment, several techniques were tried out. Initially, an external *tap.shift* object was used, which was part of the Tap Tools package developed by Tim Placew for Max/MSP environment. Whereas the *tap.shift* object is able to change audio speed in realtime, it produces audible artifacts caused by speed adjustments that deteriorate the quality of the output sound. The technique that provides the best performance is an internal object implementation of phase vocoder techniques in Max/MSP. Due to spectral prebuffering of the audio score used by this method, the patch is able to make smoother tempo adjustments.

[1]Recorded in November 2003 at Pollack Hall, McGill Strathcona Music Building.

### Overview of Phase Vocoder Technique

Phase vocoder can be defined as a channelized analysis-resynthesis tool that, through a number of techniques, measures and stores spectral signal data in different frequency bands, and then uses those values to modify and recreate the signal in time domain. Most of the phase vocoder systems use Short Time Fourier Transfrom (STFT) for analysis and resynthesis of the signal. The concept of a phase vocoder was introduced in 1939 by Dudley in his "channel vocoder" system (Dudley 1939) as a voice coding tool, and then extended by Flanagan and Golden (Flanagan and Golden 1966) and became known by its current name of "phase vocoder" through their work. A number of further accomplishments and improvements were made in design and implementation of phase vocoder technique (eg. (Depalle and Poirot 1991), (Fischman 1997), (Laroche 1998), (Laroche and Dolson 1997)).

The two most popular applications of a phase vocoder are time scaling (used in the described system) and pitch-scaling. Time scaling refers to changing the length of a signal while keeping the original frequency, which is done by interpolating/decimating the signal in the frequency domain before resynthesis. Pitch scaling, or changing the signal's frequency without affecting its length, is implemented through a combination of oversampling/undersampling the signal in time domain prior to analysis stage and time-scaling the signal to its original length.

### Phase Vocoder Implementation in Max/MSP

A phase vocoder system was implemented as a patcher in Max/Msp environment. The Short Time Fourier Transform (STFT) analysis and resynthesis

stages are implemented by using internal MSP pfft~ object. During the pre-processing stage of the phase vocoding process, the input signal from the time buffer is processed by STFT and recorded into a spectral buffer. Windowing and the overlap ratio are the two important factors which affect the quality of the signal at the resynthesis stage. During the performance stage, the signal stored in spectral buffers is readjusted and resynthesized by the pfft~ object to provide a time-domain output of a different duration. One of the main drawbacks of using an internal implementation of the Phase Vocoder in Max/MSP is linked to the realtime nature of the MSP environment: it does not provide the tools to transform a loaded audio buffer into a spectral buffer instantaneously—instead, the file has to be played back by the patch from beginning to end prior to the actual performance in order to be loaded into the buffer. An alternative solution of storing the spectral buffer in a separate file itself does not provide good results, due to the fact that the signal loaded in the spectral buffer is not normalized (which is usually done during the inverse FFT stage), and gets altered if saved into a file and then reloaded into the buffer. Another required improvement of the Performance element of the system would involve implementation of an external Phase Vocoder object in MSP that would be able to load the signal into the spectral buffer in non-realtime.

# Chapter 4

# Gesture Recognition

Recognition of isolated and continuous gestures implemented in the system is based on Hidden Markov Model procedure. This statistical observation sequence analysis process, widely known for its use in speech recognition, has been also used in score following and sign language gesture recognition systems, and has been applied to right-hand beat conducting recognition in *Multi-Modal Conducting Simulator* (Usa and Mochida 1998a), (Usa and Mochida 1998b).

Whereas there already exist several objects for Max environment that implement some of the HMM aspects, their functionality is specific to the goals of the projects they were developed for and does not provide the full scope of HMM capabilities. The main goal of the current implementation was to design an object that would provide the full functionality of a discrete HMM model (training, finding optimal sequence of states and recognition), and for the object to be general and straightforward enough not to be limited to this project, but rather be available for general use in Max/MSP software. The resulting source code of the object could be easily adjusted to be used

as a class in a different environment.

This chapter discusses the details of implementing the HMM model and its supporting components as a set of external objects in Max/MSP (known as the HMM Max/MSP Package), which was done using all of the described HMM techniques. Whereas there exist a number of alternative methods of calculation of the HMM procedures described in the chapter, the formulae presented here were collected from different sources as an optimal set of computations for practical implementation of HMM functionality. A detailed description of HMM techniques is provided in Appendix A.

## 4.1 HMM Package in Max/MSP

Since none of the existing external HMM objects written for Max environment provided the functionality required for the system, a set of external HMM objects was implemented in Max.

### 4.1.1 HMM External Object in Max

The object was written as a representation of a discrete HMM model and served as an implementation of its three principal features—learning, finding an optimal sequence of states and recognition.

The number of states, labels and the type of the object (0 for ergodic, 1 for left-to-right) are specified during its initialization by typing it as arguments in the object box, in the order they are described here. The fourth argument accepted by the object box is the maximum number for the array that records the incoming labels the object can accept at a time. Those characteristics can only be changed by reinitializing the object by changing its arguments in the

box (or importing another recorded model into the current one), since they are directly responsible for the amount of memory allocation for the arrays used by the object. If no arguments are supplied after the object name, it defaults to a 5-state 10-label ergodic model with 200 as the maximum size for the internal array of numbers recorded from the incoming stream.

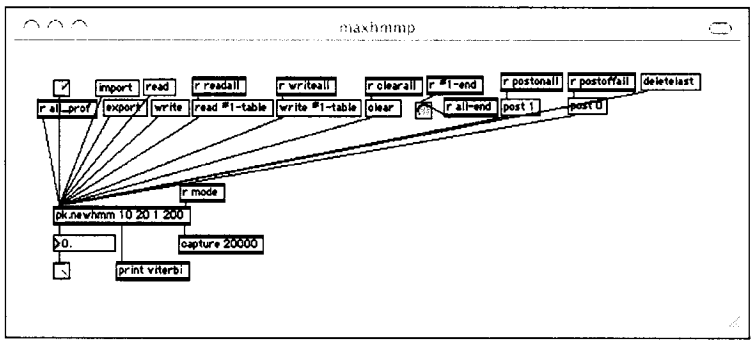


Figure 4.1: A 10-state 20-label left-to-right HMM object in Max/MSP.

The current mode of the object can be changed by sending an input to its right inlet—0 for training the model, 1 for recognition mode, and 2 for the Viterbi functionality. By default, the object is initialized in the training mode.

The object accepts an incoming stream of numbers in its left inlet and records them in an internal array in the order they are received. When a *bang* message is received in the left inlet, the labels stored in the internal array are passed to one of the object's algorithms—training, recognition or Viterbi, based on what mode it is currently in), and the internal array is cleared in order to be ready to accept the next stream of numbers. For the training algorithm, multiple scaled reestimation formulae are used to calculate the new model parameters, and the array of numbers is then stored in the object

so that it can be used during the next training procedure together with all of the previously recorded and the new incoming training array. For the recognition algorithm, probability is calculated using scaled forward and backward probability computations (ref), and the result—which will always be in the range of $-\infty < P(O|\lambda) \leq 0$, since it is a logarithm of the actual probability—is passed on to the leftmost outlet of the object. For the Viterbi algorithm, the scaled Viterbi computations are used to generate the most probable sequence of states based on the incoming array, which is then passed as a list to the middle outlet of the object.

A *deletelast* message to the left inlet of the object deletes the last training array from its memory, and retrains the model with all of the training observations that were recorded previously to the one that was deleted.

The object stores the information about the model which can be then viewed, imported or exported as a list, and written to or read from a file. The information is stored in the following format:

| number of states | number of labels | type | number of trainings | initial array | transition matrix | state label output matrix | training observations |
|---|---|---|---|---|---|---|---|

Table 4.1: Storing format of an HMM model.

Number of states, number of labels, type and number of trainings are each represented by a single integer number. State transition and state label output matrices are represented as arrays of numbers, with each of the rows of the matrices placed in left-to-right order. There is no separation needed between the two matrices, since their dimensions are specified by the 'number of states' and 'number of labels' values—for example, a 5-state 10-label model

will have a 5X5 state transition matrix represented by an array of 25 values, and a 5X10 output matrix represented by an array of 50 values. Training observations are stored after the two matrices, with each observation being in the following format:

| number of labels in this observation | observation array | -1(to indicate the end of the current observation) |
|---|---|---|

Table 4.2: Storing format of an individual training observation.

Double-clicking on the object box opens a text editor which displays the current model information. A *read* message to the left inlet of the object opens a 'read file' dialogue so that a previously recorded model can be loaded into a current object. A *read <filename>* message results in reading the file from the same directory the current Max patch with the object is in. Similarly, *write* message opens a *write file* dialogue, and *write <filename>* writes the specified file in the same directory as the Max patch. The model information can be also imported and exported as a list within the patch—an *export* message sends out of the current model data through the rightmost outlet of the object in a list format, and an *import <list>* message (where *<list>*) is a list containing model information) loads the new model information into the object. Therefore, it is possible to pass model information between several objects in the same patch by sending an *export* message to one object, appending the word *import* to the list that gets generated as the output, and sending it to another HMM object in the patch, which could be useful in some applications.

The *post 1* and *post 0* messages turn on and off the additional information

about the training, recognition and Viterbi computations to be printed in the Max window.

### 4.1.2 Other objects of the HMM Package

Two supporting external objects were written in addition to the main HMM object for the current system. The *orient2d* is responsible for calculation of positional orientation—it accepts a relative change in horizontal positional data in its left input and relative change in vertical positional data in the right output, calculates the positional orientation based on those values, and outputs the result from the right output in degrees (in the range of 0-359) and out the left output in radians $(0 - 2\pi)$.

The *code2d* object is responsible for the elementary vector quantization of the positional data stream. It divides the incoming positional data stream with a larger number of possible values (0-359 in this case) in a number of sectors, the number being determined by the desired number of labels for the HMM model object, and assigns that label to each incoming number within that sector. For example, a *code2d* object with 20 and 360 as its respective label size and maximum incoming data size, divides the range of 0-360 in 20 sectors, assigns the labels of 1 to 20 to each respective sector, and outputs a corresponding label for each incoming orientation value.

For future projects using the HMM object in Max/MSP (such as speech or music data recognition), or for advancements in the current project that would require a more complex quantization technique, other external objects will have to be written to produce the desired label stream that will serve as the input to HMM external objects.

## 4.2 Testing the HMM Objects

In order to verify the ability of the designed HMM package objects to correctly identify input gestures based on HMM training and recognition techniques, several tests were carried out. Left-to-right 5-state 10-label HMM models were used in all of the testing examples.

### 4.2.1 Symbol Recognition with a Mouse/Wacom Tablet

Recognition of English Alphabet symbols was the initial task for the system developed with the external HMM objects in order to test its performance. Absolute 2-D positional coordinates extracted from the movement of the mouse or a Wacom tablet were used to calculate the orientation values with a *orient2d* object. Resulting data stream was then passed to the *code2d* object that mapped the observation stream to a label data stream. Each of the HMM objects that were implemented in the system represented an isolated symbol to be recognized. At the learning stage, HMM objects were individually trained with 10 symbol examples. At the recognition stage, an observation stream representing a symbol was passed to all of the HMM objects, and the one producing the highest probability was considered as the recognized symbol. There were five observation examples of each symbol provided for recognition, and the system performed with a 92.5% recognition rate.

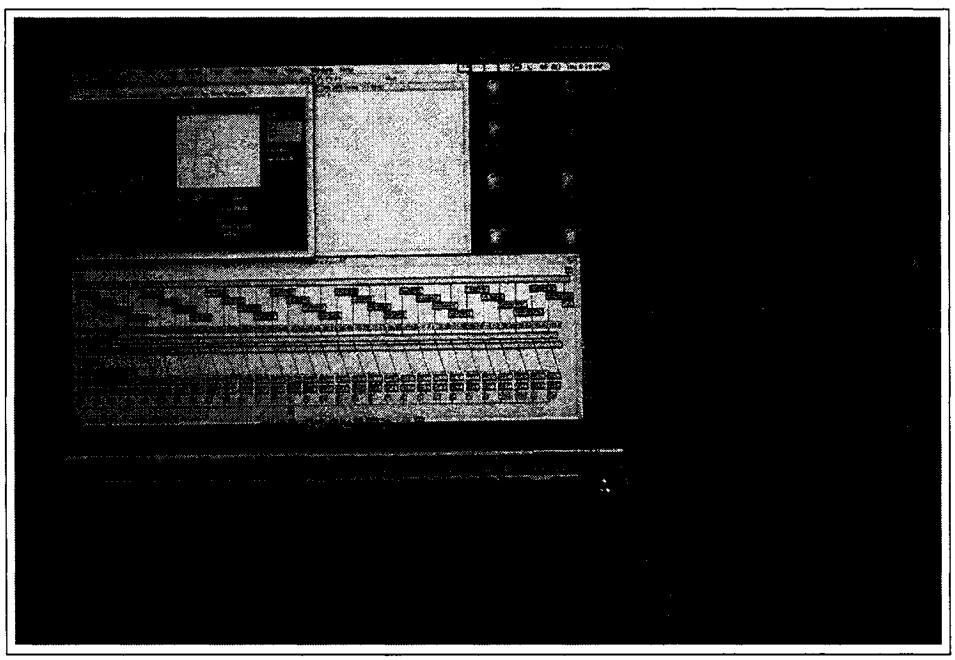

Figure 4.2: HMM symbol patch with a Wacom tablet input—recognition process.

### 4.2.2 Symbol Recognition with USB Cameras and Eyesweb

The procedure used by the symbol recognition system was then replicated using a single webcam to capture a 2-D positional user input. Gesture Analysis component of the system was used to extract and transfer the positional information from the video input to the recognition patch. Five English alphabet symbols (A,B,C,D,E) were used for training and recognition. As in the previous experiment, there were 10 training and 5 recognition sets per gesture.

Resulting recognition rates were lower than those obtained in the previous experiment. In particular, the capital symbol 'D' was repeatedly mistaken for a 'B', whereas all of the other symbols (that did not share positional similarities, as in the case of those two symbols) were correctly identified. This can be explained by the fact that Eyesweb has a faster recognition rate than the one used by the mouse tracking object, and the visual gesture symbolizing the symbol was performed during a longer period of time than writing it in with a mouse. Therefore, the left-to-right object did not contain enough states to represent all of the positional transitions, and considered the symbol 'D' as the upper part of the symbol 'B',whereas it did not contain enough available states to represent the lower part. On the basis of this observation, it was decided to use 10-state models for all of the HMM models during the actual conducting gesture recognition experiments.

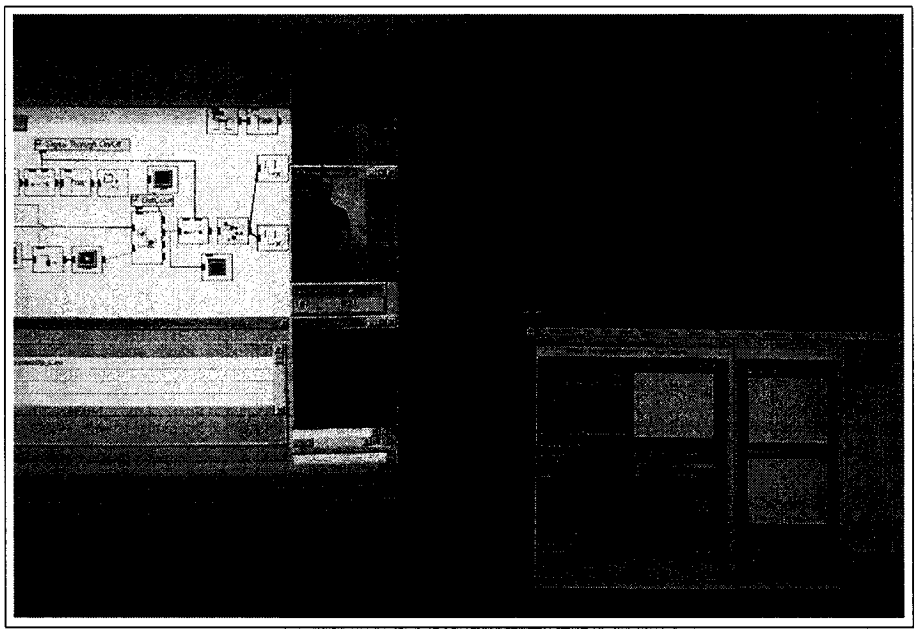

Figure 4.3: HMM symbol patch with Eyesweb input—training process.

# Chapter 5

# System Implementation and Results

This chapter describes a set of experiments conducted with the final system that was designed based on implementation of its individual components—analysis, performance and recognition—discussed in detail in previous chapters.

## 5.1 System Experiments with Conducting Gesture Recognition

All of the conducting gestures used for positional recordings were performed by a doctoral conducting student at the Music Faculty of McGill University. All of the conducting gesture recordings were done by the Gesture Analysis component of the system using the Eyesweb software with two USB cameras that were placed in front and profile view of the conductor. The recorded session files were later edited using the Adobe Premiere 6.5 software in order to prepare them for use with the recognition patches in Max/MSP (editing involved deleting unnecessary beginnings/endings of the files, and splitting

larger session files into training and recognition parts). For positional data retrieval and transfer, Gesture Analysis provided the required functionality, as in the case of the Analysis and Performance system. The HMM objects that were used in Max for all of the gesture recognition experiments were based on a 10-state 20-label left-to-right model. The choice of the model type was based on the fact that the left-to-right model is known to perform well in situations when the length of the incoming data stream for the same gesture may greatly vary from example to example, which is likely to occur in the case of conducting gestures.

### 5.1.1 Left hand Expressive Gestures

Five left-hand isolated expressive gestures were selected to be recognized—*crescendo*-cutoff, *diminuendo*-cutoff, *fermata*-click gesture, *accent* indication and expansion gesture. The set of gestures was intentionally chosen to contain both simple (accent, expansion) and complex(crescendo+cutoff, diminuendo+cutoff and fermata+click) gestures in order to test the system's ability to cope with both kinds of gestures simultaneously.

For each of the five gestures, 20 training sets and 10 recognition sets were recorded as two synchronized movie files for front and profile views, and then split into 30 individual file pairs using video editing software. In the recognition component of the system, five HMM object pairs were assigned to correspond to the gestures. Each HMM object pair was then individually trained with the 20 training video segments. Upon completion of the training process, 50 examples (10 examples per gesture) were presented for recognition to the entire set of the HMM objects. The recognition scores of the

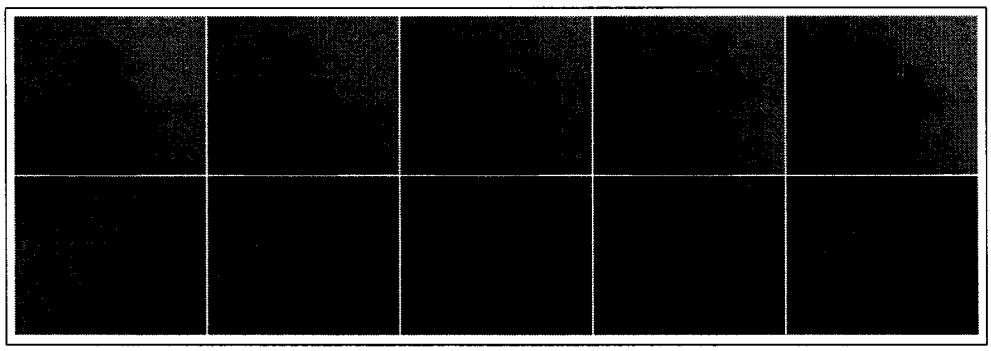

Figure 5.1: Front and profile camera view of the user training the recognition system with a left-hand *crescendo*-cutoff expressive gesture.

pairs of HMMs were combined (through simple addition of the logarithms of probabilities), and compared to find the maximum logarithm value, which indicated the gesture that was considered as the most likely to correspond to the incoming positional data stream.

Table 5.1 presents the results of the recognition experiment:

| Gesture | Trained (no.) | Tested (no.) | Recognized (no.) |
|---|---|---|---|
| Crescendo+Cutoff | 20 | 10 | 10 |
| Diminuendo+Cutoff | 20 | 10 | 10 |
| Fermata+Click | 20 | 10 | 9 |
| Accent | 20 | 10 | 10 |
| Expansion | 20 | 10 | 10 |
| Total | 100 | 50 | 49 (98% rate) |

Table 5.1: Left-hand expressive gesture recognition.

From the table, it can be seen that the gestures were recognized with a high degree of accuracy. In fact, those gestures presented a real challenge to the HMM models, since they shared many of the positional characteristics.

For example, the complex *fermata*-click gesture (which is described in detail in (Rudolph 1994)) is a combination of a *fermata* indication, followed a short 'click' or 'breath', followed by an entrance indication. The positional information received from the middle part of the *fermata*-click gesture is very similar to the simple accent gesture. Nonetheless, the HMM objects were able to distinguish the differences between the gestures. In the only incorrect recognition case, where a *fermata*-click gesture was identified as an accent, it actually *looked* more similar to an accent, and could be easily mistaken for an accent by an actual musician with knowledge of conducting gestures.

### 5.1.2 Expressive Styles of Right Hand Beat Indicating Gestures

For right-hand beat indicating gestures, three sets of beat patterns were chosen—the first set containing a four-beat expressive legato and four-beat light staccato patterns, the second with a three-beat expressive legato and three-beat light staccato, and the third set with a two-beat legato, two-beat marcato and two-beat staccato patterns. A separate HMM object pair was used to represent each beat gesture of the described patterns—so there were four HMM pairs for each pattern of the first set, three for each pattern of the second set, and two for the third set. For each beat pattern, 20 measures of continuous conducting was recorded for gesture training purposes and 10 measure of contunuous conducting for gesture recognition. In this case, instead of manually splitting the video files into individual segments for each beat of each pattern, the temporal segmentation process was performed by

using the beat identification part of the gesture analysis component of the system. A loop was performed during the execution where every time a beat was detected, the incoming positional data stream was rerouted to the HMM pair representing the next beat, and when the last beat of the beat pattern was reached, the first beat HMM pair was chosen for the next training stage.

| Gesture | Pattern | Beat (no.) | Trained (no.) | Tested (no.) | Recognized (Individual) | Recognized (Set) |
|---|---|---|---|---|---|---|
| 4-beat | expressive legato | 1st | 20 | 10 | 10 | 10 |
| | | 2nd | 20 | 10 | 10 | 10 |
| | | 3rd | 20 | 10 | 10 | 10 |
| | | 4th | 20 | 10 | 10 | 9 |
| | light staccato | 1st | 20 | 10 | 10 | 10 |
| | | 2nd | 20 | 10 | 10 | 10 |
| | | 3rd | 20 | 10 | 10 | 10 |
| | | 4th | 20 | 10 | 10 | 10 |
| 3-beat | expressive legato | 1st | 20 | 10 | 10 | 10 |
| | | 2nd | 20 | 10 | 10 | 10 |
| | | 3rd | 20 | 10 | 10 | 10 |
| | light staccato | 1st | 20 | 10 | 10 | 10 |
| | | 2nd | 20 | 10 | 10 | 10 |
| | | 3rd | 20 | 10 | 10 | 10 |
| 2-beat | neutral legato | 1st | 20 | 10 | 10 | 10 |
| | | 2nd | 20 | 10 | 10 | 10 |
| | marcato | 1st | 20 | 10 | 10 | 10 |
| | | 2nd | 20 | 10 | 10 | 10 |
| | light staccato | 1st | 20 | 10 | 10 | 10 |
| | | 2nd | 20 | 10 | 10 | 10 |
| Total | 7 | 20 | 400 | 200 | 200 (100% rate) | 199 (99.5% rate) |

Table 5.2: Recognition of Expressive Styles of Right-hand Beat-Indicating Gestures.

After all of the three sets of beat patterns (representing 7 different beat

patterns and 20 different beat model pairs) were trained, 10 measures for each beat pattern were presented to the system for gesture recognition. The recognition for each beat pattern was done twice—at first, by comparing the scores of the HMM pairs corresponding to different beats of the current beat pattern only, and then by comparing the scores of the models representing the beats of the entire set.

It is clear from the Table 5.1.2 that the system provided a robust recognition of right hand beat-indicating gestures, both in terms of correctly identifying the individual beat information within a single beat pattern, and distinguishing between different styles of conducting using the same meter value. Although the recognition system does provide an option of comparing the scores of HMMs for all of the sets and beat patterns, this comparison was not included as part of the experiment, since from the general knowledge of conducting it is obvious that there is not enough difference between the beats of the same conducting style (legato, light staccato) but different meter to differentiate between them. For example, the third beat of a 3-beat legato and the 4th beat of a 4-beat legato are visually and positionally indistinguishable. If such a functionality would be desired in the system, it would have to include information about the previous detected beat into the coding process for label assignment for the HMMs.

### 5.1.3 Embedded Right hand Expressive Gestures

Since the right hand is known to be mainly responsible for time-beating indications throughout the performance, the right hand expressive gestures have to be incorporated into the beating gestures—unlike the left hand that

has more freedom of movement and is not constrained with the baton (if it is being used). Therefore, in order to extract right hand expressive gestures, either transitions from one type of beat to another or different forms of the same basic type of beat should be analyzed.

The gestures were studied on examples of gradual *crescendo* and *diminuendo* indicated through gradual increase/decrease in the amplitude of individual beat indicating gestures. Three different cases were studied—a gradual crescendo (one measure span) on a four-beat legato, a gradual diminuendo (one measure span) on a four-beat legato, and no dynamics on a four-beat legato. In this case, each of the HMM model pairs represented the entire measure and not an individual beat in the measure as in the previous experiment, so that it would be possible to track changes in beat amplitude over the four beats. Therefore, this system variation contained three HMM pairs, one per beat pattern.

The gradual crescendo and gradual diminuendo patterns were not recorded as isolated gestures—rather, they were conducted in a continuous manner, in spans of 20 measures for training and 10 measures for recognition. Although in real-life performance, those gestures would indicate a transition from one dynamic level to another and would not occur in the repeated manner that they were recorded in, this way of recording the gestures allowed for automatic temporal segmentation and training with the gesture analysis component of the system. In this case, no rerouting of the input signal in realtime was needed (as it was done in the continuous right-hand gesture experiment), and a new training would occur every new measure (at the end of every fourth beat), and not at the end of every beat as in the

previous experiment.

| Gesture | Trained (no.) | Tested (no.) | Recognized (no.) |
|---|---|---|---|
| Gradual Crescendo (4-beat *legato*) | 20 | 10 | 10 |
| Gradual Diminuendo (4-beat *legato*) | 20 | 10 | 10 |
| No Dynamics (4-beat *legato*) | 20 | 10 | 10 |
| Total | 60 | 30 | 30 (100 % rate) |

Table 5.3: Right hand expressive gesture recognition.

Since in this case only three long patterns were analyzed, and their positional relative paths were clearly different by comparison to each other, it is not surprising that the system was able to distinguish the gesture transition patterns with a perfect recognition rate.

## 5.2 Combined Gesture Analysis, Recognition and Performance

All of the three components of the system were combined in order to test gesture recognition with realtime performance. An audio file of the 2nd movement of Mozart's 12th symphony was conducted by the user, and the HMM models that were trained as described in section 5.1.2. Initially, the results were incosistent and did not provide the recognition rates expected based on the previous experiments. Upon examination of the situation, it became clear that the main difference between gesture recognition based on prerecorded video files and realtime video inputs was that the positional data rate generated by the Eysweb positional tracking object was much lower in the latter case, due to the load of processing multiple video inputs in

realtime. Another factor that contributed to the problem was that whereas the HMM models were trained with the positional data filtered through a 6-point averaging low-pass system, the data was not filtered in the case of realtime performance to avoid latency issues, which were already present in the performance system—therefore, the data characteristics received by the HMM models during the recognition process was different from the positional streams that the models were trained with. A solution to this problem was tried where the 6-point averaging filter was added to the system and sent its information to HMM models, whereas the beat recognition process was operating on a separate path with the direct unfiltered data. Since there was a time difference between the beat identification and gesture recognition data due to the averaging filter, the detected beats were delayed by 240 ms (equivalent to 5 frames of positional data) before being sent to the recognition models to identify the beginning of the recognition process.

Whereas the input processing part of the patch was often unstable due to software/hardware limitations, the HMMs were able to recognize the incoming gesture data with a high degree of accuracy during those cases when an acceptable number of positional frames was received—the system performed with a 94.6% recognition rate over the 73 measures (and 294 beats of data), with comparison being done between the four possible gestures in the 4-beat legato pattern.

Those issues identified the fact that for realtime HMM recognition purposes, a higher-rate lower-latency input processing system is needed. Nonetheless, the experiment demostrated that HMMs can provide a high recognition rate during a realtime conducting performance.

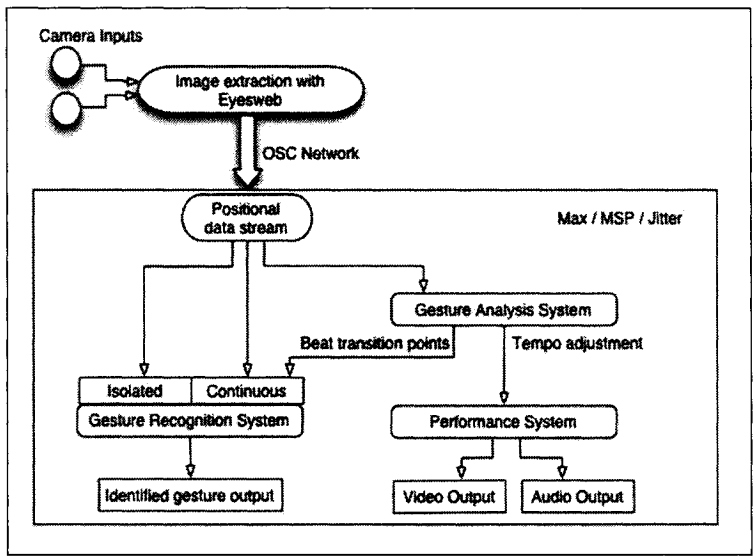


Figure 5.2: Schematic representation of the gesture analysis, recognition and performance system.

# Chapter 6

# Discussion

## 6.1 Evaluation of Results

One of the main drawbacks of the project was the inability of the camera input devices to provide information at the sufficient rate and resolution in order for the system to perform with a high level of precision. Using 6DOF magnetic sensors instead of cameras will resolve the issue in the future, and will enable the use of some additional features—such as tracking the movement of the conducting baton. Nonetheless, through experiments described in the previous chapter it was proven that HMM techniques can be successfully used for recognition of both time-beating and expressive gestures.

## 6.2 Conclusion

The purpose of the work was to develop a set of practical tools to be applied for future work and to test them to make sure they are appropriate for gesture recognition, rather than providing an in-depth study into classification of conducting gestures.

The main achievement of the work is development of an HMM-based procedure that can be applied to analysis and classification of expressive conducting gestures. In particular, HMM training and recognition processes were applied to analysis of both right hand beat indicator gestures and left hand expressive articulation gestures. This brings an improvement over existing systems, since whereas right hand movements had been analyzed with HMM and Artificial Neural Net techniques in the past, there has been no previous research involving high-level recognition and classification techniques applied to left hand expressive gestures.

The designed HMM package, which is available for free distribution online (*http://www.music.mcgill.ca/~pkoles/download.html*), is intended for use as a general tool in Max/MSP environment. It could be applied not only towards positional data classification but also towards any other process that involves pattern recognition—such as speech recognition, timbre recognition or score following. The resulting set of analysis, HMM-based recognition and performance tools will be directed towards future research in development of standardized classification of conducting gestures.

## 6.3 Future Work

One of the future goals of the project is to design a gesture recognition process that can be implemented in a continuous conducting movement environment in combination with the developed gesture analysis and performance system. Whereas the issue of temporal segmentation of a continuous gesture observation stream can be easily solved for right-hand beat indicating gestures through the use of information extracted by another process (such as

tracking positional maxima and minima of the gestures), there is no simple way of using a similar technique for expressive gestures, since there is no clear uniform indication of positional transitions between them. A solution to this problem lies in the capability of HMM process to automatically segment an entire observation stream into isolated gesture states. This technique involves training HMM models separately with isolated gestures, and then chaining the trained models together into a single network of states. The Viterbi algorithm can then be used on the entire observation stream, so that the temporal segmentation problem is simplified to computing the most probable path through the network.

Upon completion of the continuous process of gesture recognition, the eventual goal of the work will be to develop a classification library of conductors gestures for computer conducting gesture recognition systems. This part of the project will address the need for development of a uniform set of conducting gesture definitions in terms of their positional information and mappings to the music score. The proposed library will be based on the existing well-developed grammar of traditional conducting technique, and will be introduced as a standardized set of gesture definitions to be used for future research in the field of conducting gesture recognition. Positional 3-D recording of the library gestures will be done with Vicon 460 Motion Capture and Polhemus Liberty systems now available at McGill Input Devices and Music Interaction Laboratory.

# Appendix A

# Hidden Markov Model

This appendix provides a technical overview of the concept and functionality of a Hidden Markov Model [1]. Whereas there exist a number of alternative methods of calculation of the HMM procedures described in the appendix, the formulae presented here were collected from different sources as an optimal set of computations for practical implementation of HMM functionality.

## A.1 Hidden Markov Model –Definition and Overview

Hidden Markov Model(HMM) is a structure that is used to statistically characterize the behavior of sequences of event observations. HMM is an extension of a model known as Markov Chains. Whereas Markov Chains deals with observation sequences that are fully accessible, HMM works with representation of so called "hidden" event which cannot be observed directly.

[1] A detailed overview of general HMM techniques can be found in (Rabiner and Huang 1986) and (Rabiner 1989). Scaling procedure and other practical issues mentioned in this chapter are described in (Deller, Hansen, and Proakis 2000), (Huang, Ariki, and Jack 1990) and (Lien 1998).

By definition, "An HMM is a double stochastic process with an underlying stochastic process which is not observable, but can only be observed through another set of stochastic process that produce the sequence of observed symbols" (Rabiner and Huang 1986). In other words, HMM is applied to an observable process which has been generated from another "hidden" process that is the main area of interest. The technique used to obtain an observable sequence from the hidden sequence is known as vector quantization.

### A.1.1 Introduction to HMM

The idea behind HMM is that any observable sequence can be represented as a sequence of states, with each state corresponding to a grouped portion of sequence values and containing its characteristic features in statistical form. HMM keeps track of what state is likely to be assigned the initial portion of the sequence, of what values are likely to occur in each state, and of what state-to-state transitions are likely to take place.

#### HMM Parameters

A Hidden Markov Model can be characterized through the following set of parameters:

$N$: number of states $(S_1, S_2, \ldots, S_N)$ of the model

$M$: number of labels $(Q_1, Q_2, \ldots, Q_M)$ that occur in the observation sequence (also referred to as codebook size)

$\Pi$: an array of initial state probabilities (size of the array is N):

$$\Pi = \{\pi_i\} = (\pi_1, \pi_2, \ldots, \pi_N) \tag{A.1}$$

$A$: an $N \times N$ matrix of state-to-state transitional probabilities:

$$\mathbf{A} = \{a_{ij}\} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix} \tag{A.2}$$

where $a_{ij}$ is a probability that a transition from state $S_i$ to state $S_j$ will take place.

$B$: an $N \times M$ matrix of state output probabilities:

$$\mathbf{B} = \{b_i(l)\} = \begin{pmatrix} b_1(1) & b_1(2) & \dots & b_1(M) \\ b_2(1) & b_2(2) & \dots & b_2(M) \\ \vdots & \vdots & \ddots & \vdots \\ b_N(1) & b_N(2) & \dots & b_N(M) \end{pmatrix} \tag{A.3}$$

where $b_i(l)$ is a probability that an occurrence of label $Q_l$will take place in state $S_i$.

The HMM model is then referred to as:

$$\lambda = (\Pi, A, B) \tag{A.4}$$

### Observation Parameters

An observation sequence used as an input array to HMM can be characterized by:

$T$: number of observations $(O_1, O_2, \dots, O_T)$ in the observed sequence O

$K$: number of multiple observation sequences $(O^1, O^2, \dots, O^K)$

### HMM Types

An HMM model can be of ergodic or left-to-right type. In an ergodic type, which is a general case of an HMM, the process is allowed to start and finish in any state, and all possible state-to-state transitions are allowed. In a left-to-right HMM type, the process is bound to start in the first state and finish in the last state, and the only transitions allowed at any given point are those of a state transition to itself or of a state transition to the consequent state.

## A.2 Three HMM problems

There are three main problems associated with application of HMM in sequence recognition process:

Problem 1: given an observation sequence $O$ and a Hidden Markov Model $\lambda$, calculate $P(O|\lambda)$ – the probability that the model would produce this observation sequence. It is also known as an HMM Recognition problem.

Problem 2: given an observation sequence $O$ and a Hidden Markov Model $\lambda$, calculate the optimal sequence of states $(I_1, I_2, \ldots, I_T)$ that would maximize the likelihood of $\lambda$ producing the observation. It is also referred to as HMM Uncovering Problem.

Problem 3: given an observation sequence $O$ (or a set of observation sequences $(O^1, O^2, \ldots, O^K)$) and a Hidden Markov Model $\lambda$, adjust the model parameters $\Pi, A, B$ so that probability of the model $P(O|\lambda)$ is maximized. This problem is also called HMM Training.

### A.2.1 Solution to Problem 1 – Forward-Backward Algorithm

A straightforward approach to the problem would be to go through all possible state sequences of length $T$, calculate probability of each sequence and then compute the result as a product of all of the sequence probabilities. This procedure would require $2T * N^T$ calculations – which is beyond computational capacity of any computer even for small values of states and observation times. A more efficient method used to find a solution to the problem is known as a Forward-Backward algorithm.

#### Forward Procedure

The algorithm uses a forward probability variable $\alpha$ that is defined as:

$$\alpha_t(i) = P(O_1, O_2, ..., O_t \mid i_t = i, \lambda) \tag{A.5}$$

i.e. the probability that the observation sequence is in state i at observation time t, given the model and the partial observation sequence $O_1 - O_t$. The following procedure is used to compute alphas for every state at all observation times:

1. For the first observation time $t = 1$,

$$\alpha_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \tag{A.6}$$

2. For all other observations $1 < t \leq T$ and $1 \leq i \leq N$,

$$\alpha_t(i) = [\sum_{j=1}^{N} \alpha_{t-1}(i) a_{ji}] b_i(O_t) \tag{A.7}$$

3. Probability of the model $P(O|\lambda)$ is then calculated as:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{A.8}$$

This procedure is based on the assumptions of 1st-order HMM that the state of a model at any observation time $t$ is dependent only on its state at the previous observation time $(t-1)$. Therefore, each of the alphas is calculated only based on information from the current and previous states. Based on the above equations, it is clear that computational cost of the procedure is now equal to $N^2T$ calculations.

**Backward Procedure**

Another method of calculating a probability of an HMM model is to use a backward probability variable $\beta$ which is defined as:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, ..., O_T \mid i_t = i, \lambda) \tag{A.9}$$

i.e. the probability of the partial observation sequence $O_t - O_T$ that starts in state i at time t, given the model $\lambda$. A similar iterative procedure is used to calculate betas of all states for every observation time but starting from the last observation:

1. For the last observation time $t = T$,

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \tag{A.10}$$

2. For all other observations $1 \leq t < T$ and $1 \leq i \leq N$,

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_i(O_{t+1}) \beta_{t+1}(j) \tag{A.11}$$

3. Probability of the model $P(O|\lambda)$ is then calculated as:

$$P(O|\lambda) = \sum_{i=1}^{N} \pi_i b_i(O_1)\beta_1(i) \tag{A.12}$$

The complexity of backward probability computation is also equal to $N^2T$ calculations since it uses the same sequential process of going through all states at each observation time as in case of forward procedure.

### A.2.2 Solution to Problem 2 – The Viterbi Algorithm

In order to compute the optimal sequence of states that would maximize the likelihood of an observation sequence $O$ being produced by a Hidden Markov Model $\lambda$, a procedure known as the Viterbi algorithm is used. An optimal sequence of states $Q = \{q_1, q_2, \ldots, q_t\}$ is found by computing optimal state probability values $\delta_t(i)$ for all states at each time step $t$, recording their indices as $\psi_t(i)$, and retrieving the optimal state indices $q_t(i)$ through a backtracking procedure. The maximum probability value at the final time step $T$ is equal to the overall model probability—therefore, the Viterbi algorithm can be used alternatively to the Forward-Backward procedure to calculate the value of $P(O|\lambda)$. The Viterbi algorithm can be summarized through the following sequence of steps:

1. Initialization.

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \tag{A.13}$$

$$\psi_1(i) = 0, \quad 1 \leq i \leq N \tag{A.14}$$

2. Recursion.

$$\delta_t(j) = \max_{1 \leq i \leq N}[\delta_{t-1}(i)a_{ij}]b_i(O_t), \quad 2 \leq t \leq T, \quad 1 \leq j \leq N \tag{A.15}$$

$$\psi_t(j) = \arg \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}], \quad 2 \le t \le T, \quad 1 \le j \le N \tag{A.16}$$

3. Termination.

$$P(O|\lambda) = \max_{1 \le i \le N} [\delta_T(i)] \tag{A.17}$$

$$q_T = \arg \max_{1 \le i \le N} [\delta_T(i)] \tag{A.18}$$

4. State sequence backtracking.

$$q_t = \psi_{t+1}(q_{t+1}), \quad t = T-1, T-2, \ldots, 1. \tag{A.19}$$

### A.2.3 Solution to Problem 3 – Baum-Welch Reestimation

A method known as Baum-Welch reestimation is used to maximize $P(O|\lambda)$ of an HMM. It uses probability variables $\epsilon_t(i,j)$, $\gamma_{i,j}$ and $\gamma_t(i)$ to reestimate $\pi$, $A$, $B$ parameters of the model.

The first variable $\epsilon_t(i,j)$ is defined as the probability that HMM process is in state $S_i$ at time $t$ and in state $S_j$ at time $(t+1)$, given the sequence $O_1, \ldots, O_t$. It is calculated as:

$$\epsilon_t(i,j) = \frac{P(q_t = S_i, q_{t+1} = S_j, O_1, \ldots, O_T)}{P(O|\lambda)} = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} \tag{A.20}$$

Another variable used by Baum-Welch reestimation procedure is $\gamma_t(i)$. It is defined as the sum of probabilities of transitions from state $S_i$ to $S_j$ for all times $t$ of observation sequence $O_1, \ldots, O_t$.

$$\gamma_{i,j} = \sum_{t=1}^{T-1} \epsilon_t(i,j) = \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} \tag{A.21}$$

Probability variable $\gamma_t(i)$ is defined as the probability that current observation label at time $t$ is assigned to state $S_i$ of the model.

$$\gamma_t(i) = \sum_{j=1}^{N} \epsilon_t(i,j) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} \tag{A.22}$$

Using definitions of $\epsilon_t(i,j)$, $\gamma_{i,j}$ and $\gamma_t(i)$, we can now describe Baum-Welch reestimation formulae for $\pi$, $A$, $B$:

1. Initial probability:

$$\bar{\pi}_i = \gamma_1(i) = \frac{\alpha_1(i)\beta_1(i)}{P(O|\lambda)} \tag{A.23}$$

   where $1 \leq i \leq N$.

2. Transition probability:

$$\bar{a}_{ij} = \frac{\gamma_{ij}}{\sum_{j=1}^{N} \gamma_{ij}} = \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(i)} \tag{A.24}$$

   where $1 \leq i \leq N$ and $1 \leq j \leq N$.

3. Output probability:

$$\bar{b}_i(l) = \frac{\sum_{t=1}^{T} \{t|O_t = v_t\}\gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} = \frac{\sum_{t=1}^{T} \{t|O_t = v_l\}\alpha_t(i)\beta_t(i)}{\sum_{t=1}^{T} \alpha_t(i)\beta_t(i)} \tag{A.25}$$

   where $1 \leq i \leq N$ and $1 \leq l \leq M$.

The formulae are simplified so that intermidiate variables $\epsilon_t(i,j)$, $\gamma_{i,j}$ and $\gamma_t(i)$ do not have to be calculated – nonetheless, they were included in the

presentation to support theoretical explanation of the reestimation procedure.

Baum-Welch training procedure is as follows:

Step 1: create an initial HMM model $\lambda_0$. For an ergodic model, random values are assigned to all of the model variables. For a left-to-right model, equal distribution values are assigned to all allowed initial and transition parameters as well as to all output parameters.

Step 2: Using formulae A.23, A.24 and A.25, create a new model $\lambda_1$ from $\lambda_0$.

Step 3: If $P(O|\lambda_1) > P(O|\lambda_0)$ or/and if the reestimation process has not yet run the specified number of times, continue. Otherwise, stop.

## A.3 Additional Issues Related to HMM Implementation

This section describes the practical issues that were encountered during the design stage of the HMM object. Computational solutions to the issues are provided.

### A.3.1 Scaling

Forward-Backward algorithm uses a recursive procedure to calculate values of $\alpha_t(i)$ and $\beta_t(i)$ for each observation time $t$. Because of the recursive nature of the process, those values will be calculated as a result of many multiplications including transition and output probability probabilities. Since those

probabilities $a_{ij}$ and $b_{il}$ will always be in the range $0 \leq (a_{ij}, b_{il}) \leq 1$, the values of $\alpha_t(i)$ and $\beta_t(i)$ will exponentially get closer and closer to 0 as the process is going through each observation time $t$. Therefore, for long observation sequences there is a danger that some of the $\alpha_t(i)$ and $\beta_t(i)$ values will be beyond computational precision range. A solution to this problem involves scaling all of the forward and backward probability values by a scaling coefficient which will keep them in the desired range and will cancel out at the end of the computation of reestimation parameters without interfering with final results.

### Scaled Forward Probabilities

Auxiliary variables $\tilde{\alpha}_t(i)$ and $\tilde{\alpha}_t(i)$ have to be defined for the forward scaling process:

1. For the first observation time $t = 1$,

$$\tilde{\alpha}_1(i) = \alpha_1(i) \tag{A.26}$$

2. For all other observations $1 < t \leq T$ and $1 \leq i \leq N$,

$$\tilde{\alpha}_t(i) = \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ji} b_i(O_t) \tag{A.27}$$

where

$$\hat{\alpha}_t(i) = c_t \tilde{\alpha}_t(i) \tag{A.28}$$

and scaling coefficient $c_t$ is defined as

$$c_t = \frac{1}{\sum_{i=1}^{N} \tilde{\alpha}_t(i)} \tag{A.29}$$

### Scaled Backward Probabilities

In a similar way, auxiliary variables $\tilde{\beta}_t(i)$ and $\tilde{\beta}_t(i)$ are used for backward probability calculations:

1. For the last observation time $t = T$,

$$\hat{\beta}_1(i) = \beta_T(i) \tag{A.30}$$

2. For all other observations $1 \leq t < T$ and $1 \leq i \leq N$,

$$\hat{\beta}_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \tilde{\beta}_{t+1}(j) \tag{A.31}$$

where

$$\tilde{\beta}_t(i) = c_t \hat{\beta}_t(i) \tag{A.32}$$

Scaling coefficients $c_t$ are not recalculated during the backward procedure – instead, $c_t$ coeffeicients from forward algorithm computations are used to adjust backward probability values.

### Probability Calculation

The probability of the model $P(O|\lambda)$ can now be calculated using scaling coefficients $c_t$:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) = \frac{1}{\prod_{t=1}^{T} c_t} \tag{A.33}$$

However, in practice there will be a possibilty that the resulting probability value will also be smaller than computational precision range. A solution to this is to calculate $\log P(O|\lambda)$ instead:

$$\log(P(O|\lambda)) = -\sum_{t=1}^{T} log\ c_t = -\sum_{t=1}^{T} log \sum_{i=1}^{N} \tilde{\alpha}_t(i) \tag{A.34}$$

which produces a negative value in an acceptable range.

### Scaled Reestimation Parameters

In Baum-Welch reestimation formulae, forward-backward probabilities $\alpha_t(i)$ and $\beta_t(i)$ are replaced by scaled variables $\hat{\alpha}_t(i)$ and $\hat{\beta}_t(i)$:

1. Scaled initial probability

$$\bar{\pi}_i = \frac{\hat{\alpha}_1(i)\hat{\beta}_1(i)}{\sum_{i=1}^{N} \hat{\alpha}_T(i)} = \hat{\alpha}_1(i)\hat{\beta}_1(i)c_T \tag{A.35}$$

2. Scaled transition probability

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \hat{\alpha}_t(i)a_{ij}b_j(O_{t+1})\hat{\beta}_{t+1}(j)c_{t+1}}{\sum_{t=1}^{T-1} \hat{\alpha}_t(i)\hat{\beta}_t(i)} \tag{A.36}$$

3. Scaled output probability

$$\bar{b}_i(l) = \frac{\sum_{t=1}^{T} \{t|O_t = v_t\}\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{t=1}^{T} \hat{\alpha}_t(i)\hat{\beta}_t(i)} \tag{A.37}$$

Since scaling coefficients $c_t$ were used during calculation of all of the auxiliary variables used in the formulae, the results of the reestimation calculations are not affected, as the scaling coefficients get cancelled out.

### Scaled Viterbi Algorithm

In order to simplify the computation by replacing multiplications by additions, and to avoid number precision issues that were previously described for Forward-Backward and reestimation algorithms, logs of values are used in the case of Viterbi algorithm as well. The scaled Viterbi equations are:

1. Initialization.

$$\tilde{\delta}_1(i) = \log(\delta_1(i)) = \log(\pi_i) + \log(b_i(O_1)), \quad 1 \leq i \leq N \tag{A.38}$$

$$\tilde{\psi}_1(i) = 0, \quad 1 \leq i \leq N \tag{A.39}$$

2. Recursion

$$\tilde{\delta}_t(j) = \log(\delta_t(j)) = \max_{1\le i\le N}[\tilde{\delta}_{t-1}(i)+\log(a_{ij})]+\log(b_i(O_t)),\ 2 \le t \le T,\ 1 \le j \le N \tag{A.40}$$

$$\psi_t(j) = \arg\max_{1\le i\le N}[\tilde{\delta}_{t-1}(i) + a_{ij}],\ \ 2 \le t \le T,\ \ 1 \le j \le N \tag{A.41}$$

3. Termination.

$$\log(P(O|\lambda)) = \max_{1\le i\le N}[\tilde{\delta}_T(i)] \tag{A.42}$$

$$\tilde{q}_T = \arg\max_{1\le i\le N}[\tilde{\delta}_T(i)] \tag{A.43}$$

4. State sequence backtracking.

$$\tilde{q}_t = \tilde{\psi}_{t+1}(\tilde{q}_{t+1}),\ \ t = T-1, T-2, \ldots, 1. \tag{A.44}$$

### A.3.2 Multiple Observation Sequences

In the case where a model is being trained with $K$ observation sequences instead of a single sequence, the reestimation procedure can be used over all of the sequences at the same time.

#### HMM Probability

Probability of the model can be found as a product of individual probabilities calculated for all of the observation sequences:

$$P(O|\lambda) = \prod_{k=1}^{K} P(O^k|\lambda) \tag{A.45}$$

#### Reestimation Parameters

Initial probability value is calculated by weighing all of the $\bar{\pi}_i^k$ equally:

$$\bar{\pi}_i = \frac{1}{K}\sum_{k=1}^{K}\gamma_1^{(k)}(i) = \frac{1}{K}\sum_{k=1}^{K}\bar{\pi}_1^{(k)} \tag{A.46}$$

The weight of transition probabilities are dependant on the length of each observation sequence $O^k$:

$$\bar{a}_{ij} = \frac{\sum_{k=1}^{K} \gamma_{ij}^{(k)}}{\sum_{k=1}^{K} \sum_{j=1}^{N} \gamma_{ij}^{(k)}} = \frac{\sum_{k=1}^{K} \frac{1}{P(O^{(k)}|\lambda)} \sum_{t=1}^{T_k-1} \alpha_t^{(k)}(i) a_{ij} b_j(O_{t+1}^{(k)}) \beta_{t+1}^{(k)}(j)}{\sum_{k=1}^{K} \frac{1}{P(O^{(k)}|\lambda)} \sum_{t=1}^{T_k-1} \alpha_t^{(k)}(i) \beta_t^{(k)}(i)} \tag{A.47}$$

which is also the case for output probabilities:

$$\bar{b}_i(l) = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \{t|O_t = v_t\} \gamma_t^{(k)}(i)}{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \gamma_t^{(k)}(i)} = \frac{\sum_{k=1}^{K} \frac{1}{P(O^{(k)}|\lambda)} \sum_{t=1}^{T_k} \{t|O_t = v_t\} \alpha_t^{(k)}(i) \beta_t^{(k)}(i)}{\sum_{k=1}^{K} \frac{1}{P(O^{(k)}|\lambda)} \sum_{t=1}^{T_k} \alpha_t^{(k)}(i) \beta_t^{(k)}(i)} \tag{A.48}$$

### Scaled Reestimation Parameters

As in the case of a single observation sequence, the reestimation parameters for multiple observation sequences also have to be scaled. The scaled reestimation formulae are:

1.
$$\bar{\pi}_i = \frac{\sum_{k=1}^{K} \hat{\alpha}_1^{(k)}(i) \hat{\beta}_1^{(k)}(i)}{\sum_{k=1}^{K} \sum_{i=1}^{N} \hat{\alpha}_{T_k}^{(k)}(i)} = \sum_{k=1}^{K} \hat{\alpha}_1^{(k)}(i) \hat{\beta}_1^{(k)}(i) c_{T_k}^{(k)} \tag{A.49}$$

2.
$$\bar{a}_{ij} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^{(k)}(i) a_{ij} b_j(O_{t+1}^{(k)}) \hat{\beta}_{t+1}^{(k)}(j) c_{t+1}^{(k)}}{\sum_{k=1}^{K} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^{(k)}(i) \hat{\beta}_t^{(k)}(i)} \tag{A.50}$$

3.
$$\bar{b}_i(l) = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \{t|O_t = v_t\} \hat{\alpha}_t^{(k)}(i) \hat{\beta}_t^{(k)}(i)}{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \hat{\alpha}_t^{(k)}(i) \hat{\beta}_t^{(k)}(i)} \tag{A.51}$$

# List of Figures

# List of Tables

# Bibliography


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

Bevilacqua, F., J. Ridenour, and D. Cuccia (2002). 3D motion capture data: Motion analysis and mapping to music. In *Proceedings of the Workshop/Symposium on Sensing and Input for Media-centric Systems.*

Borchers, J., W. Samminger, and M. Muhlhauser (2002). Engineering a realistic real-time conducting system for the audio/video rendering of a real orchestra. In *Proceedings of the 4th International Symposium on Multimedia Software Engineering*, pp. 352–362.

Brecht, B. and G. Garnett (1995). Conductor Follower. In *Proceedings of the International Computer Music Conference*, pp. 185–186. International Computer Music Association.

Buxton, W., W. Reeves, G. Fedorkov, K. C. Smith, and R. Baecker (1980). A microprocessor-based conducting system. *Computer Music Journal 4*(1), 8–21.

Camurri, A., P. Coletta, M. Peri, M. Ricchetti, A. Ricci, R. Trocca, and G. Volpe (2000). A real-time platform for interactive performance. In *Proceedings of the International Computer Music Conference.* International Computer Music Association.

Deller, J. R., J. H. Hansen, and J. G. Proakis (2000). *Discrete-time Processing of Speech Signals.* New York: IEEE Press.

Depalle, P. and G. Poirot (1991). SVP: A modular system for analysis, processing and synthesis of sound signals. In *Proceedings of the International Computer Music Conference.* International Computer Music Association.

Dudley, H. (1939). *The Vocoder.* Reprinted in Schafer, R. and J. Markel (1979). *Speech Analysis.* IEEE Press.

Fischman, R. (1997). The phase vocoder: Theory and practice. *Organized SOund 2*(2), 127–145.

Flanagan, J. L. and R. M. Golden (1966). *Phase Vocoder.* Reprinted in Schafer, R. and J. Markel (1979). *Speech Analysis.* IEEE Press.

Haflich, F. and M. Burns (1983). Following a conductor: The engineering of an input device. In *Proceedings of the International Computer Music Conference.* International Computer Music Association.

Huang, X. D., Y. Ariki, and M. Jack (1990). *Hidden Markov Models for Speech Recognition.* New York: Columbia University Press.

Ilmonen, T. and T. Takala (1999). Conductor following with Artificial Neural Networks. In *Proceedings of the International Computer Music Conference*, pp. 367–370. International Computer Music Association.

Keane, D. and P. Gross (1989). The MIDI BATON. In *Proceedings of the International Computer Music Conference*, pp. 151–154. International Computer Music Association.

Keane, D., G. Smecca, and K. Wood (1990). The MIDI Baton II. In *Proceedings of the International Computer Music Conference*, pp. 151–154. International Computer Music Association.

Keane, D. and K. Wood (1991). The MIDI Baton III. In *Proceedings of the International Computer Music Conference*, pp. 541–544. International Computer Music Association.

Laroche, J. (1998). *Time and Pitch Scale Modifications of Audio Signals.* Kahrs, M. and K. Brandenburg, eds. *Applications of Digital Signal Processing to Audio and Acoustics.* Kluwer Academic Publishers.

Laroche, J. and M. Dolson (1997). About this phasiness business. In *Proceedings of the International Computer Music Conference.* International Computer Music Association.

Lee, M., G. Garnett, and D. Wessel (1992). An adaptive conductor follower. In *Proceedings of the International Computer Music Conference*, pp. 454–455. International Computer Music Association.

Lien, J. J. (1998). *Automatic Recognition of Facial Expressions Using Hidden Markov Models and Estimation of Expression Intensity.* Ph. D. thesis, The Robotics Institute, Carnegie Mellon University.

Long, R. G. (1971). *The Conductor's Workshop: A Workgroup on Instrumental Conducting.* Dubuque, Iowa: Wm. C. Brown Company Publishers.

Malko, N. (1950). *The Conductor and His Baton.* Denmark: Wilhelm Hansen, Copenhagen.

Marrin, T. (2000). *Inside the Conductors Jacket: Analysis, Interpretation and Musical Synthesis of Expressive Gesture.* Ph. D. thesis, Massachusetts Institute of Technology.

Marrin, T. and J. Paradiso (1997). The Digital Baton: A versatile performance instrument. In *Proceedings of the International Computer Music Conference*, pp. 313–316. International Computer Music Association.

Marrin, T. and R. Picard (1998). The Conductors Jacket: A device for recording expressive musical gestures. In *Proceedings of the International Computer Music Conference*, pp. 215–219. International Computer Music Association.

Mathews, M. (1989). *Current Directions in Computer Music Research.* MIT Press.

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

Mathews, M. V. (1976). The Conductor program. In *Proceedings of the International Computer Music Conference*, Cambridge, Massachusetts.

Mathews, M. V. (1991). The Radio Baton and the Conductor Program, or: Pitch—the most important and least expressive part of music. *Computer Music Journal 15*(4), 37–46.

McNeill, D. (1992). *Hand and Mind: What Gestures Reveal About Thought.* University of Chicago Press.

Morita, H., S. Otheru, and S. Hashimoto (1989). Computer music system that follows a human conductor. In *Proceedings of the International Computer Music Conference*, pp. 207–210. International Computer Music Association.

Morita, H., S. Otheru, and S. Hashimoto (1990). Knowledge information processing in conducting computer music performance. In *Proceedings of the International Computer Music Conference*, pp. 332–334. International Computer Music Association.

Murphy, D., T. H. Andersen, and K. Jensen (2003). Conducting audio files via Computer Vision. In *Proceedings of the 2003 International Gesture Workshop*, Genoa, Italy.

Rabiner, L. R. (1989). A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of IEEE 77*(2), 257–285.

Rabiner, L. R. and B. H. Huang (1986). An introduction to Hidden Markov Models. *IEEE Acoustics, Speech and Signal Processing Magazine 3*(1), 4–16.

Ross, A. A. (1976). *Techniques for Beginning Conductors.* Belmont, California: Wadsworth Publishing Company.

Rudolph, M. (1994). *The Grammar of Conducting: A comprehensive guide to baton technique and interpretation.* Toronto: Maxwell Macmillan Canada.

Segen, J., A. Mujumder, and J. Gluckman (2000). Virtual dance and music conducted by a human conductor. *Eurographics 19*(3).

Takala, T. (1997). Virtual Orchestra Performance. In *ACM SIGGRAPH 97 Visual Proceedings*, pp. 81. International Conference on Computer Graphics and Interactive Technologies.

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

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

Usa, S. and Y. Mochida (1998a). A conducting recognition system on the model of musicians process. *Journal of Acoustical Society of Japan 19*(4), 275–287.

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

Wachsmuth, I. and M. Frohlich (Eds.) (1998). *Gesture and Sign Language in Human-Computer Interaction: Proceedings of the International Gesture Workshop, Bielefeld, Germany, September 1997).* Springer-Verlag.

Wright (1998). Implementation and performance issues with OpenSound Control. In *Proceedings of the International Computer Music Conference*, pp. 224–227. International Computer Music Association.