19

Machine Perception of Natural Musical Conducting Gestures

Matthew Wayne Krom

B.A. *cum laude* Computer Science Cornell University, 1993

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning, in partial fulfillment of the requirements for the degree of Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1996

Author ______________________________
Program in Media Arts and Sciences,
May 10, 1996

Certified by ______________________________
Aaron Bobick
Assistant Professor of Computational Vision
Program in Media Arts and Sciences
Thesis Supervisor

Certified by ______________________________
Barry Vercoe
Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by ______________________________
Stephen A. Benton
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Machine Perception of Natural Musical Conducting Gestures

by

Matthew Wayne Krom

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning, on May 10, 1996, in partial fulfillment of the requirements for the degree of Master of Science

ABSTRACT

Musical conducting is a richly expressive mode of human expression. A recognition and conductor-following system is presented which interprets the motions of a human conductor during a musical performance. Gestural features are recognized using a rule-based recognition model, which is motivated by a model for musical expression. Information in the score is augmented by expressive and timing information gained through gesture recognition, and the resulting music is played through audio.

Perceptual aids, such as context-sensitivity of the gestures' meanings, and the context found in the musical score, assist the perception task. It is proposed that natural motion recognition is possible because key features of the recognition model remain consistently meaningful across individual conductors, allowing for the development of gestural models which accommodate a diverse range of users.

Thesis Supervisor: Aaron Bobick
Title: Assistant Professor of Computational Vision

Thesis Supervisor: Barry Vercoe
Title: Professor of Media Arts and Sciences

Machine Perception of Natural Musical Conducting Gestures

by

Matthew Wayne Krom

Reader ______________________________

Whitman Richards
Professor of Cognitive Science
Head, Media Arts and Sciences Program

Acknowledgments

The Vision and Modeling, and Machine Listening groups at the Media Lab have provided an atmosphere where each person brings his or her own brand of intelligence, and even genius. My time here has been brief, but stimulating.

Andy, Dave, Josh, and the Christmas Cactus have been excellent officemates.

Aaron Bobick has always been insightful and inspirational for his understanding of the principles of high-level computational vision, which I found to be refreshing when I first visited the lab as a prospective student.

Whitman Richards, through his class "Knowledge, Perception, and Cognition," has really influenced the way I think about the mind and perception. Incidentally, Whitman was my advisor's advisor.

I applaud the High Level Vision group (HLV) for our self-organized weekly meetings that have brought us together at least a little more. To HLV I leave osterley.media.mit.edu, procured for this project and aged six months.

Barry Vercoe taught me the down-to-earth basics of computer-generated audio. As one of the pioneers of computational musical accompaniment, Barry is an inspiration. I also thank Barry for his genuine interest in my thesis topic.

I thank Tod Machover, for giving me an excuse to learn Java (to produce musically expressive web pages) and for his class which worked to produce an Internet presence for the Brain Opera.

Thanks again to my readers for their contribution to this project. When people such as Aaron, Barry, and Whitman become interested in your project, there is real incentive!

I thank my parents Joe and Anita, without whom this would certainly not be possible. One of my only regrets of University on the East Coast has been the distance away from both of them and my sister Nicole.

Finally, I am thankful for April, who deserves some credit for everything I attempt. Thanks for being wonderful through our entire Media Lab experience.

Contents

List Of Figures

Chapter 1

Introduction

A system for conducting virtual music performers -- a conductor-following computer system.

Consider a computer system in which a human conductor directs an ensemble of virtual performers. Sensors convey the position and attitude of the conductor's hands to detection and recognition algorithms. The conductor's motions are translated into performance parameters for the musical piece being played, which is represented by a score. A sound generation program running in real time provides audio feedback to the conductor, completing the performance loop.

Such a system could eventually become an instrument for individual musical expression, since performances of the same score differ only in the expressive content of the individual's conducting motions. For those who do not consider themselves musically trained on an instrument, such a system could be an empowering tool for musical expression.

One measure of technical success of an implemented conducting system would be its usefulness to beginning conducting students. Far in the future, a student could benefit from the opportunity to build his or her skills without coordinating a practice session with many musicians.

Finally, a real time conducting system could also serve as a testbed for research topics in musical interaction and expressiveness.

This research is concerned with the gesture recognition and detection algorithms, and the associated musical expression models. The remainder of the implemented system -- audio rendering, real-time inter-process communication, audio representation of musical qualities such as staccato -- are present to the extent necessary to support the recognition and detection module.

Gesture recognition is currently a hotly discussed topic within the MIT Media Lab, and within the vision and AI community in general. The concept is confounded by the vast number of senses of the word "gesture." Indeed, there is currently no satisfactory definition of a human gesture. Useful definitions so far have included those based on a single configuration or position of the hands, a sequence of view-based images [6,23], the trajectories and configurations of the hands of speakers during a dialogue [4], or particular trajectories through three-dimensional space [19]. This research deals with a limited and highly constrained gesture system -- musical conducting -- which has developed over hundreds of years as an international form of communication.

Chapter 2

Motivation and Background

Musical conducting has evolved over two hundred years, and has been formalized to some extent in textbooks [7,17]. In order to introduce the participation of generalized microprocessors in live performance, it is necessary to formalize conducting in a way such that algorithms can be written for detection and recognition of natural conducting motions. This is a relatively new endeavor.

2.1 Why "natural" gestures?

When a person uses this musical conducting system, we do not wish to require him or her to learn a new repertoire of unnatural movements to control the expressive parameters. It is our desire that the conducting system presented here react to gestures that are naturally developed to conduct music.

Clearly, there are as many conducting styles as there are professional conductors. There exists a textbook style of learning conducting, which is accepted in the spirit of the adage, "learn the rules so that you can then break them." Departure from the textbook style is the norm.

It is my belief that a computer conducting system can perform reasonably well when used by a variety of people who use different conducting styles. For some of the most basic concepts, such as staccato, legato, and dynamics, there are basic indications which are interpreted fairly consistently [7]. A general conducting recognition system can be achieved by identifying those features of motion which consistently relate to a particular musical idea.

The system developed here assumes a common denominator across conducting styles. There is strong agreement in the local community that many salient features do exist which describe a variety of conducting styles, and remain fairly consistent from individual to individual. Two examples are beat indication and note articulation. Beats are generally indicated at the lowest point of a trajectory. Note articulation is indicated by severity or smoothness of motion -- harsh motion indicates a staccato attack, while fluid motion signifies a smooth legato note articulation.

Therefore, the recognition model does not require the learning of specific trajectory paths, and does not make use of any extraneous cues or switches other than that of the hand motion.

2.2 Conducting systems

Research on conducting perception has focused primarily on following the beat of the conductor. Within the past few years, attempts have been made to capture more of the expressive nature of musical conducting. Several teams of researchers have implemented systems similar to the one presented here [1,10,12,13,14,20].

A single feature -- the vertical position of the baton -- is typically used for successful detection. In fact, this feature alone is sufficient to study beat detection and tempo tracking [10,20]. The bottom trough of the baton path is taken to be the beat, and the top peak is interpreted as the back-beat.

Max Mathews pioneered expressive control over computer performances. His "Groove" program circa 1960 was one of the first very successful attempts at allowing for expressive control of a computer music piece. Later, he developed a "Radio Baton" three-dimensional input device for his "Conductor" system which allowed for balance and volume controls of individual voices [12].

Because conducting scenarios (including human orchestras) are fundamentally real-time systems, some method of prediction is necessary to sense the beat (and other gestures) in time for the performer to react. This problem of prediction is an interesting enough research problem in itself, and has been implemented by all previous efforts out of necessity.

A team of researchers headed by H. Morita have implemented a virtual performance system which follows the beat of a conductor, deriving volume from the length of the path of the baton through space during a beat. A white marker attached to a baton is used to recover position via a CCD camera [13,14].

A group of researchers working at CNMAT address beat prediction using a neural network learning technique. Only the vertical position of the baton is used for their work, and results have been inconclusive, other than an acknowledgment of a need to represent the beat at much higher than quarter-note resolution [10].

In existing systems, only the vertical position of the baton is considered when detecting the beat. The bottom-most extreme of motion is taken to be the beat; the upper-most extreme is often taken to be the back beat. Some researchers have noted that the top-most position of the baton does not always occur halfway through a beat, and varies from conductor to conductor [10].

Within the standard beat, a conductor may wish to direct at a finer level of detail (say, to indicate the three notes of a triplet on the fourth beat). This addition to the expressive model has not been attempted here. The hierarchical nature of the beating motion has not been adequately addressed in conducting systems to date.

State of the art

The current generation of microprocessors is fast enough to generate CD-quality audio in real-time following arbitrary sound programs. This allows for real-time continuous shaping of audio, and for more programmer control over sound parameters in real-time. Barry Vercoe's Csound program and language is an example of one such system, and is used to generate the audio in the present system.

Work in digital music is yielding a greater understanding and interest in models for expressive musical performance. Recently at the media lab, analysis of expressive timing in percussive music was produced by Jeffrey Bilmes in 1993 [2]. Expressive timing was modeled as a scalar offset from the metric tempo timing. The fledgling field of musical expressive parameters may soon develop many alternate agreed-upon models for expression.

I currently know of no portable, easily-installed conductor recognition system that is robust enough to be destined for popular use. It is conceivable that some of the above-mentioned systems have now been developed to the point where untrained musicians can express themselves through conducting. I personally feel that my own implementation is a mere starting point toward such an expressive experience, but that a system which begins to address these issues may be produced within five years.

2.3 Connections within the Media Lab

Teresa Marrin, for work towards her Masters degree, has constructed a new device for conducting, currently termed “the baton.” She has written a musical piece, and has developed a gestural semaphore language for shaping and composing the piece during live performance. This work is in conjunction with Chris Verplaetse, Joe Paradiso, and several other research assistants, who are instrumental in the design of the hardware interface. The device, which may be made wireless, could be used with the present conducting system. Teresa and I have been in close contact during our research.

Barry Vercoe has implemented a computer performer which follows a human conductor, and was one of the first researchers to examine several of the phenomena involved in performing alongside a machine. In particular, Barry and Miller Puckette created a computational accompanist which learned non-linear tempo accompaniment behavior through rehearsal [20].

Eric Scheirer recently created a system with the goal of extracting expressive timing information from an audio stream, based on the context of a musical score. The output of his system is an estimate of the exact onset time for each note [18].

Robert Rowe's work at the Media Lab culminated in a broad treatment of the properties of real-time musical performance systems as a class [16], which is general enough to be applied to other performance domains such as dramatic theater [15].

Others within the Media Lab have worked toward linking computer vision with the structure recovery problem, which is solved for our purpose by sensors such as the Polhemus. Specifically, Lee Campbell has studied the problem of “seeing” natural movements in real 3D data in his work on recognizing ballet steps using phase-space constraints [5]. Using a phase space representation of the human body allows for vivid analysis of the physical constraints of human body motion, and joint angle pair-predictors can be used to recognize action. Andy Wilson has presented a state-based approach to learning visual behavior from an image sequence, an alternate non-structural approach which lends itself to conducting gesture recognition [23].

2.4 Related research domains

Musical conducting can be cast as an action annotation problem -- the static score is augmented with annotations denoting musical expressive constructs. Annotation is a sufficiently interesting, defined problem, and some progress has been made on annotation from video sources [3,5,9].

Conducting is also an instance of the script-following paradigm. The idea that natural world scenarios may be modeled according to domain-specific scripts is inherited from the artificial intelligence community. Recently, the use of a script has been shown to be of assistance in high-level vision domains such as automated studio camera control [15]. Of course, the "script" for the conductor problem is the musical score.

2.5 Sensing

A common problem in computer vision today is the recovery of 2-D or 3-D shape or structure from video. Even when complete recovery is possible, for many perceptual tasks involving action there often remains a fundamental question, "what do we do with this data?" [5] Musical conducting is one such perceptual task. The position information over time must be analyzed and musical gestures must be recognized.

For this research, a six degree-of-freedom position sensing apparatus tracks the hands; there is no video input to the system. Forgoing the vision problem for now allows us to concentrate on the rest of the problem, which is the recognition and detection of the conductor's motions. When the 3D recovery problem is solved in the future, the results will function nicely as a computer vision front-end to the present conducting system.

Previous conducting systems have utilized technologies which recover the position of the tip of the baton in space. Typically the position of an LED or other light source is recovered from a digital video stream, by locating the point with the highest luminance [1,11,20]. Other approaches use proximity sensors which yield a scalar value: sonar waves, electromagnetic field disturbance, and ambient room lighting have all been used to achieve this effect.

2.6 Context

Classical recognition problems are often posed as categorization problems: given an input, name which class the input belongs to. The perception of conducting motions involves not only recognizing what class a given gesture belongs to, but detecting when it occurs, and also understanding the relevant information it conveys for musical expression.

Indeed, many of the gestures seen by a performer during a performance are expected, and the performer endeavors to understand the style in which they are executed. That is to say, performers often know which class of gesture will occur at a given moment (e.g. an entrance cue). Important information is conveyed in the manner in which the conductor presents each cue, as well as when it happens.

Increasingly, the context of a task is being recognized by the vision community as an aid for recognition. Perception of musical conducting gestures is aided by the context of the musical score. It is this context which enables a performer to expect certain gestures within a given time interval. The musical score is a context which constrains the perception tasks. In particular, the present system takes advantage of the context-expected nature of several conducting gestures to build relatively simple rule-based gesture detection algorithms.

Examples of such context-expected events are those associated with the fermata (hold), entrance cues, and the beat. Other performance gestures, such as those for controlling dynamics, are not as strongly expected, but rather are extracted from the data in a context-sensitive manner. The context for this conducting problem is discussed in more detail under the section on "Approach."

2.7 Common denominator features across individual conducting styles

Here, the term "style" refers to a single person's method of conducting, to the extent that any individual signature is present. "Style" is used in an ephemeral sense, analogous to the acknowledging "personality" of a conductor.

If we expect to construct a natural conductor gesture recognition system which is usable by many individual conductors, we have assumed that there are key classes of meaningful features which remain consistently indicative across many people's styles. In other words, it must be that individual styles have enough in common that a usable system can be built upon those common features.

For example, many conductors indicate the beat at the time instant when the hand reaches its lowest vertical point.

To the extent than any new user has to adapt his or her gestures in order for the beat to be recognizable, this system has failed in its goal of recognizing natural motion. If and when user adjustment does occur, an endeavor should be made to represent the user's original style within this system's model of gestures. These cases should guide the development of the conducting system's natural gesture recognition models; it is unacceptable to force a conductor to learn the system's representation.

Verification of this concept is encouraged in the "extensions" section below.

Chapter 3

Approach

3.1 Goal of the work

This recognition subsystem becomes meaningful as part of a greater system: The results of the perception system are converted into musical representations, and are used to generate the audio output of a "virtual performer." The system can be thought of as a filter which reads a static score and augments the information with timing, dynamics, and other expressive parameters based on the motion of a conductor.

The goal of this work is to build recognition and detection algorithms for perceiving a conductor's motions in real time. This is accomplished by (1) constructing a musical expressive model which contains all the types of musical variations the system is capable of generating, and (2) constructing a model for recognition, based on the musical expressive model, in the form of a system of rules for recognition. Throughout this document, the relationship between the expressive models and the recognition models will be explored; construction of the recognition models is driven by the expressive model in each case.

The terms "detection" and "recognition" have appeared several times already. By "recognition," I am referring to a categorization problem in which a segment of data is classified by attaching a label which refers to one of several classes of examples. "Detection" is the acknowledged presence of some feature in the data. Often, the context of following a musical score reduces the set of possible recognized gesture classes; when this situation is recognized, and a single gesture is awaited, the term "detection" is used to describe the problem of recovering the instant at which the expected event happens.

A "recognition model" for this system is a set of feature extractions and rules upon which detection and recognition are based.

The set of gesture-classes which are present in the current musical expression and recognition include the beat for tempo, note articulation (staccato, legato), fermata, entrance cues, and one class of dynamic direction. These parameters are among the most fundamental to be communicated in conducting, and were chosen for this reason [7,17]. In addition, these gesture classes are also among the most temporally-constrained cues available to a performer, effectively simplifying the recognition problem. Modeling specifics for each of these five classes are discussed below.

In the complete system, an attempt has been made to provide satisfactory audio feedback to the human conductor. Although the audio is an important influence on the conductor's motion, well-rendered performance audio is not addressed in this document.

3.2 Definition of the scenario

The performance model used here contains several key assumptions. The musical piece is performed from a known score, and all notes in the score are played; no notes are played that are not in the score. The music is assumed to be in a style that is normally scored in this way-- typically Western classical music.

The computer system (herein referred to as the "virtual performer") determines when, and in what manner, to play the notes in the score. Variables in the performance include tempo, note articulation, dynamics, and other timing information such as entrance cues, exits, and fermata. For the purposes of this system, all of the virtual performer's expressive information comes from the conductor.

In the current scenario, all notes are played by a single virtual performer -- there is only one performer to direct. The conductor cannot direct individual performers or sections.

3.3 Perceptual tasks

This research addresses the question of what a machine must perceive in order to attend to a conductor. That is to say, what does it need to sense, and what does it need to know?

The problem of conductor-following as addressed here involves expressive extraction. Sensing involves recognizing motion by categorization and detecting events which are expected, as well as extracting performance parameters from detected or recognized events. The performance features extracted from a gesture are not necessarily the same as those used to identify or detect the gesture. This concept is strongly tied to the context-sensitivity of gestures within this domain.

3.3.1 Context

A musical performance of the kind addressed here is heavily context-driven; information in the score dictates which notes will be played and in what order, and identifies times at which key events are to occur For example, the conductor can be expected to indicate entrances, exits, and fermatas at appropriate places in the performance. The context for this problem is a set of expected events, as well as a history of events which have already happened.

Within this context, certain gestures are "context-expected," meaning that the context strongly indicates that a specified gestural event will occur within a given time interval. For highly context-expected gestures, detection is sometimes prevalent, instead of the general recognition problem: the set of possible explanations of the data is drastically narrowed under the context-expected condition. Once detected, the timing information and expressive nature of the gesture are then inferred from the data.

As it turns out, many of the gestures seen by a performer during a performance are expected from the musical score. For example, an entrance cue can be expected when a performer enters after a long period of rest. A beat motion will be detected if it occurs a time when a beat is expected to occur.

A second context is the context sensitivity in the interpretation of a motion. A given motion may take on different meanings at different instances, and may carry several embedded meanings, depending upon the current state of the performance. This research models a musical performance as a state machine in which the meaning ascribed to a given motion or gesture is sensitive to the current state. State transitions are governed by the detection or recognition of conducting gestures, and the passage of musical events during the performance.

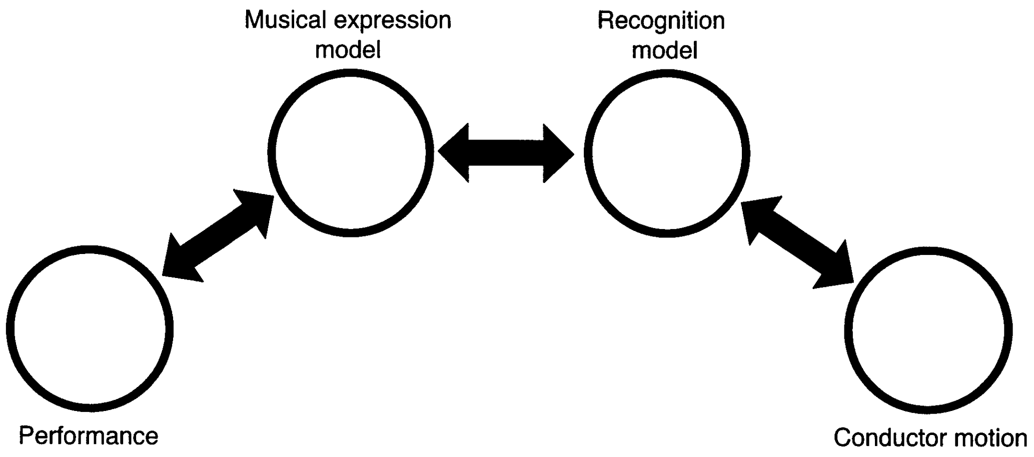

Figure 3-1. The approach for the present conducting system.

3.3.2 Modeling approach

The approach for this conducting system is outlined in figure 3-1. Performance is the music played by the virtual performer; it is based upon some notion of expression of the music in the score. The parameters for expression are extracted from the conductor's motion, using recognition algorithms based on the recognition model. All capabilities for communication between the conductor and the virtual performer are modeled in two phases: an expressive model and a recognition model.

The four realms of the approach -- Performance, Musical expression model, Recognition model, and Conductor motion -- should not be considered completely distinct. For instance, the conductor's motion has much in common with the performance, and it is not clear where the distinction between the two lies.

The virtual performer's performance arises from two sources: the musical information in the score, and interpretation of expressive information from the conductor. The family of expressive techniques that the system "knows about" is referred to as the "musical expression model."

The musical expression model forms the requirements for a recognition model: given a certain expressive characteristic, how can it be inferred or registered from the conductor's motion? The expression model does not necessarily provide insight as to how to build a recognition model, and no procedural guidelines have been developed to assist the development of a recognition model based on an expressive model. Herein lies the fundamental issue of communication during a performance. Later in this document, it is proposed that mappings between expressive classes and data-set features might be learned through experimentation. Clues for constructing the recognition models used in this work have come primarily from textbooks, and from the successes of previous researchers.

The recognition model is instantiated as an extraction of features from the conductor motion data, along with rules for recognizing and detecting the presence of a given expressive motion.

Next, we discuss how each of the classes of expression are modeled in the present system. The five categories are ordered in a progression from their presence as context-expected gestures to context-sensitive gestures. For each class, the underlying expressive model is outlined. The following section, "Algorithms," will present the recognition models based upon each expressive model.

3.3.3 The beat

The beat is part of the tempo aspect of the expressive model. The beat prediction is important for determining a tempo for the music to be played.

Beat prediction for tempo

The problem of tempo mapping -- warping the score's timeline to perform in real time with expressive changes in tempo -- arises in any conductor-following system. Many approaches to beat-following are be based upon a prediction-and-compensation method [13]. A prediction of the next beat is made, based upon the period of previously detected beats. A delay or advance in time is gradually recovered during the following period of a half beat. It has been noticed that a computer which responds immediately to tempo changes sounds unnatural to human collaborators [22]. Please see the discussion on tempo mapping below.

3.3.4 Fermata

The expressive model of the fermata is that, when a fermata is present, all notes are to be held until the fermata and its terminating gesture are detected. The termination is interpreted to extract an expressive characteristic of the cutoff.

The fermata, or hold, is an example of a strictly modal gesture. During the course of the note(s) being held, the conductor motion tends to be either quite insignificant, which signifies the continuation of the hold, or decidedly abrupt, to mark the end [7,17]. For the conductor to make an in-between gesture would be potentially confusing to the performers. This is one example of how the modal nature of gestural communication can be exploited by the perception task [3].

Since the fermata is present in a musical score, it is also an expected gesture. As such, it is an example of a gesture which can be recognized (detected) and acted upon by exploiting context information.

3.3.5 Entrance cues

I propose that the two answers a performer seeks to infer when following an entrance cue are, "when should I enter?" and "how should I enter?" The first question must be answered accurately for the cue to be meaningful. Unlike beat detection, in which timing errors can be made up during the following beat, an entrance cue demands a precise reaction.

The expressive model of the entrance cue is simply that music begins at the instant of the "cue." The "how should I enter?" question in answered by the severity of the approach, and a tempo extracted from the period of the preparatory motion.

Like the fermata, entrance cues are expected from the information in the score.

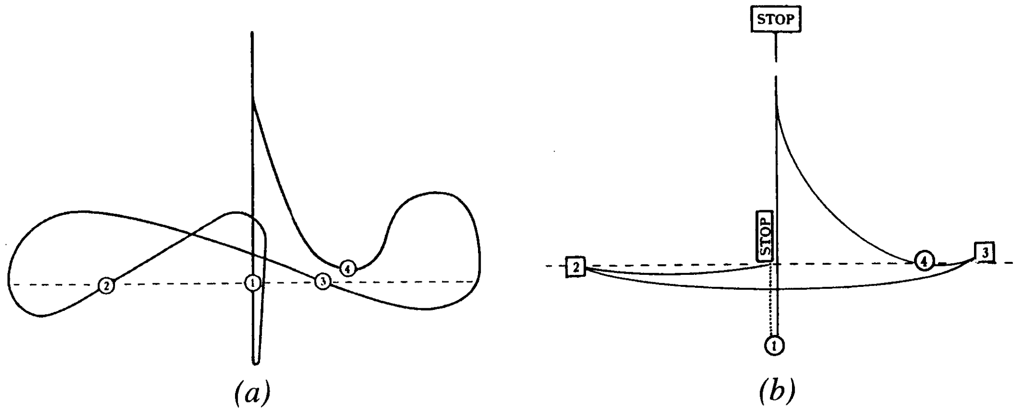

Figure 3-2. (a) espressivo legato, (b) espressivo staccato techniques, from Max Rudolf, "The Grammar of Conducting."

3.3.6 Note articulation

Each note to be played has its own articulation. Here, the articulation of notes is limited to *staccato* and *legato*, which are generally accepted to be complementary categories [7]. A *staccato* note is short in duration, with a sharp attack. A note played *legato* is performed languidly, with a mild attack, and perhaps with a delayed onset [7]. These two modes are indicated by distinctly different conducting motions, as illustrated in figure 3-2. No timing information is present in the diagram above, but the reader can infer timing characteristics from Max Rudolf's description of staccato:

> The full-staccato beat is a quick, slightly curved motion with a stop on each count. It is snappy and energetic, with a characteristic "bouncing" on the downbeat. The size may vary from small to large. [17]

3.3.7 Dynamics

Dynamics will ideally be modeled to accommodate gradual changes in volume (*crescendo* and *decrescendo*) as well as sudden changes. In the "algorithms" section, the present simple approach to modeling dynamics is presented. In the future, the expressive model for dynamics should include localized as well as global volume.

3.3.8 Tempo mapping

In order for a musical performer to follow another performer's beat, he or she must continually vary the tempo to account for changes in the other performer's tempo. In the present system, the computer performer must vary tempo to follow the beat of the conductor. This is the problem of tempo mapping.

A popular way to approach the problem is to use the representation of a mapping, or "warping," from one time-coordinate system to another. Virtual time, used to represent the score, must be mapped into real-time for playback. (This is not to be confused with the *dynamic programming* technique of Dynamic Time Warping.)

Since time intervals are represented in the score with respect to a constant reference time interval, the score can be thought of as taking place in constant virtual time: one quarter-note per beat (whatever constant value the beat might have for a particular score), with beats happening at regular known virtual times. Real-time tempo constantly changes; real-time beats do not happen at a pre-determined instances. The virtual time of the score must then be warped into the real-time of the performance so that the performers beats relate to the beats indicated by the conductor.

Because setting and changing the tempo makes an assumption about when the next beat is likely to occur, prediction of the beat is an integral part of the tempo mapping problem. In addition, the tempo mapping problem involves assumptions which can be learned through rehearsal [22].

Chapter 4

Algorithms

This section presents the algorithms used to address the specific perception tasks defined above. For each musical expression model, a recognition model is developed. The models for expression and recognition are related in that an expressive model provides the requirements for building a recognition model.

Figure 4-1 outlines the features used in the recognition model for each expressive model. Each recognition model is discussed in detail below.

Expressive model	Recognition model feature
Beat detection	peak in second derivative of vertical position
Entrance cues	rule-based over time: period of rest, preparatory motion upwards, downbeat
Note articulation	standard deviation of the normalized distance between samples during the last beat
Dynamics	mean distance between samples during the last beat
Fermata	rule-based over time: relatively constant curvature (thresholded), abrupt change in curvature

Figure 4-1: mapping of expressive models to recognition models for this system.

4.1 Hard-coded vs. adaptive feature selection

Beat prediction, or determining in advance at what instant in time a beat is likely to occur, can be cast as a supervised learning problem. It is not obvious which features of the beating motion are most useful in predicting the beat. The answer of when the beat occurs must be predicted in order for the performer to respond. Once the beat actually occurs, the system receives feedback as to the accuracy of the prediction. It may be possible, using a supervised learning approach, to attend to those features which best predict the beat in a generalized case. In the section 4.2.2, a simple form of model-switching is discussed, which has been implemented as a means for better predicting the beat.

A second level of learning for performance is feasible within a specific performance rehearsal. A specific individual who conducts a certain piece multiple times may convey the same interpretation consistently. Tempo changes will then occur at similar instances, with similar acceleration, and so on. In this case, a "rehearsal paradigm" using a supervised learning technique will conceivably improve in predicting when the next beat will occur throughout the piece, and in anticipating tempo changes. In this manner, the tempo mapping will better respond to a variety of fluctuations in tempo [20].

4.2 The beat

4.2.1 Beat detection

Beat recognition here involves a two-step process. Candidate beats are detected from troughs in the vertical position of the hand. Candidates are then accepted or rejected on the basis of the current tempo -- beats which would result in a drastic tempo acceleration are discarded as error detections. A beat which would result in a dramatically slower tempo may be an indication of a skipped detection -- a beat which was not recovered from the data.

Simply, all troughs in the data are accepted as candidate beats. A trough is defined as a change in vertical velocity from positive to negative (assuming a coordinate system with increasing values for lower altitudes).

Beat candidates are accepted or rejected based upon the ratio of the candidate's indicating tempo (the new value of the tempo in the case that the beat is accepted) to the current tempo.

Skipped beat detections are recovered using a simple linear constant-velocity assumption. Given the last known velocity, the score beat which is closest in real time to the present time is aligned with the new beat candidate. In this manner, the score alignment remains consistent with the audio that has been played.

4.2.2 Beat prediction

Deciding on an effective tempo involves making a prediction about when the next beat is likely to occur. The prediction is based on assumptions about the way the tempo is changing locally.

Two basic classes of assumptions have been made. The assumption of constant tempo yields a predictor which expects the next beat at the same interval as previous known beats. A predictor which uses an assumption of constant acceleration expects the next beat to occur as a geometric progression of previous beat periods.

> Given known beat real times B[0..n], where n is the most recent detected beat, and time intervals I[1..n], where I[n] = B[n] - B[n-1]:
>
> Given that the next beat will occur at some interval after the most recent beat, we define: B[n+1] = B[n] + I[n+1].
>
> Constant tempo assumption: I[n+1] = I[n], yields B[n+1] = B[n] + I[n].
>
> Constant acceleration assumption: I[n+1] - I[n] = I[n] - I[n-1], yields B[n+1] = B[n] + 2I[n] - I[n-1].

We expect the constant tempo assumption to perform better when the real tempo is constant, and we expect the constant acceleration assumption to perform better when the real tempo is gradually increasing or decreasing. An array of different detectors can be constructed from variations on these two basic assumptions.

A switching policy determines which prediction model to use for the next beat. The switching policy used here is simple: choose the predictor which yielded the smallest error in predicting the most recent recovered beat. An analysis of this policy is discussed in Chapter 6.

4.3 Fermata

When a fermata exists in the score, the fermata recognition rules cause the system to hold all notes until the "end fermata" gesture is recognized. This gesture is defined as follows:

- once the time instant of the fermata begins, the conductor's hand motions are expected to be smooth (relatively constant curvature),

- followed by a termination gesture to signal the release of the held note. The termination gesture consists of an "abrupt" motion, to be detected by a sudden change in motion speed and direction.

In detecting the fermata, as well as the other timing cues for entrance and exit, the modal nature of the gesture is extremely useful in recognition and detection. That is to say that the gesture progresses between phases which are greatly different from each other. For example, during a fermata (hold) a conductor makes two kinds of motions: extremely still and then extremely abrupt (to indicate the instant for releasing the hold).

4.4 Entrance cues

An entrance cue is defined here as follows:

- A period of rest,
- followed by an upward motion, which peaks
- followed a downward motion, culminating in a trough.

The upwards and downwards time intervals must be sufficiently close in length, and the resulting tempo must fall within a pre-defined interval. These two rules ensure that random preparatory motions are unlikely to be recognized falsely as the entrance.

If these conditions are satisfied, the piece begins. The tempo is set to the tempo of this initial entrance cue, and the first musical event is scheduled.

This approach follows the spirit of natural gesture recognition, in that there is no "start" key, command, or defined trajectory.

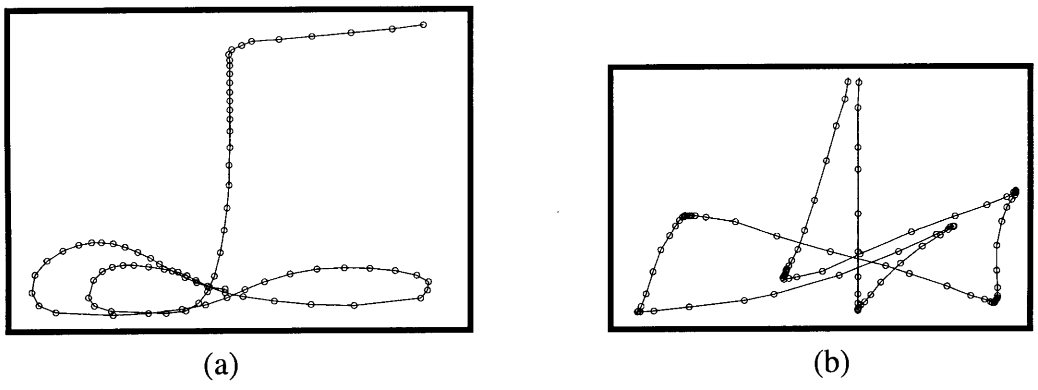

Figure 4-2: real two-dimensional data, exhibiting (a) high value for legato, (b) high value for staccato, along the one-dimensional note articulation scale.

4.5 Note articulation

Note articulation is modeled here as a scalar value, ranging from *staccato* to *legato*. Note articulation is computed for each note just before it is played.

Staccato and legato are common modes of articulation. Although there is probably not a one-dimensional scale between "staccato-ness" and "legato-ness," articulation can be modeled in that manner as a beginning. It is accepted that the articulation of a given note is foretold by the back-beat of the conducting gesture [7]. This is necessarily true since, once the beat occurs, the performers have begun playing the note and the stylistic form (of the note's attack, at least) cannot be altered.

As a first-order description, staccato is indicated by rapid, straight motions of the baton, with motion-pauses after each beat. Legato is indicated by fluid motions of near-uniform speed. Two primary visual features associated with both gestures are the degree of curvature of the path of the baton, and the acceleration as the baton moves between beats [7,17]. Figure 4-2 illustrates these concepts with real two-dimensional data.

4.6 Dynamics

In basic conducting texts, a starting point can be found for the gestural models for controlling dynamics. For a start, there are these three basic ways a conductor can control dynamics: [17]

(1) Through grandness of motion. A beating gesture which covers more distance typically indicates a louder volume; shorter trajectories indicate softer.

(2) Through changing the baseline of beating. Vertical changes in the baseline -- the imaginary horizontal line where the beat is indicated -- may be used to indicate louder or softer dynamics (for higher and lower baselines, respectively).

(3) Through use of the non-dominant hand. The left hand (right hand for left-handed conductors) may control dynamics. Upward motions, or facing the palm toward the body, indicate louder volume. Downward motions, and facing of the palm away from the body indicate a softer passage.

The current implementation uses only guide (1) above.

4.7 Tempo mapping

A tempo mapping model is a strategy for mapping virtual time (the score) to real time (a specific performance). A satisfactory tempo mapping model must answer the question, "given the current time, where exactly in the score is the performance?" The model also must answer the question "given a score event in the future, at what time will it occur under the current state?" When the tempo state changes, all times for pending events must of course be recalculated to reflect the new mapping. In the present system, only one musical event is pending at a given time.

Two key representation issues shape the tempo mapping model in this system. First, the tempo mapping consists of continuous linear segments. The points at which the linear segments join are referred to as "anchor points." Secondly, an anchor point is set each time a beat is detected. These two features determine the shape and performance of the tempo mapping.

This representation also enforces certain assumptions about the tempo of the piece. For instance, this representation can only encode instantaneous changes in tempo.

The implementation for tempo mapping contains two fundamental pieces: a beat-prediction method, along with a tempo-mapping method. Both methods are intimately related, and use a common model along with common assumptions. Given that the real times of all previous beats are known, it is necessary to predict the real time at which the next beat will occur. In turn, given the current tempo, and current virtual time, the new tempo must vary in order to align the two time-frames with respect position and time.

The method of tempo-varying is also based on a simple model. Recall that tempo mapping can be thought of as a mapping between a virtual timeline (the score), and a real timeline (the real time performance). Given the current position in the score (position in the virtual timeline), the tempo is set to a constant value which will bring the performance to the desired place in the score (i.e. the next beat) at the correct real-time (i.e. the predicted time of the next beat), given the prediction of when the next real-time beat will occur. This model represents the last known beat, called the *anchor*, as well as the *prediction* beat. Figure 4-3 illustrates an instance of a tempo mapping based upon predicted and detected beats.

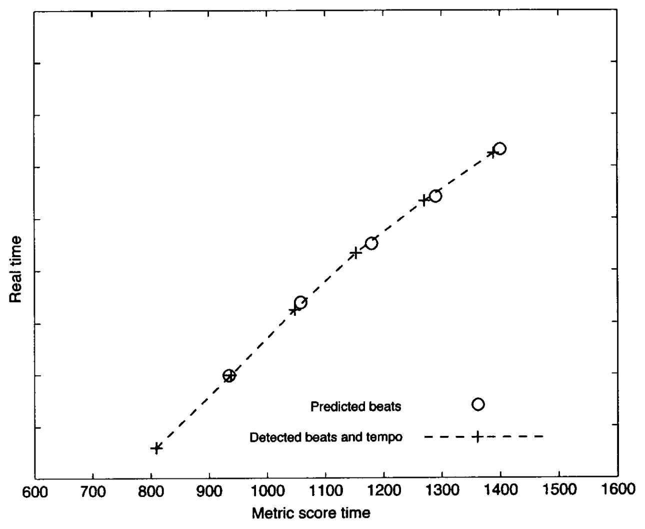

Figure 4-3: Predicted vs. detected beats.
Tempo acceleration is evidenced by the curvature of the graph, and the overestimation of the interval under the constant tempo assumption.

In the current scheme, this determination of tempo state changes only when new information is gathered about real-time beats, which is effectively when a new beat is detected. As a result, velocity changes instantaneously when a new beat is detected, and remains constant until the next detected beat. Clearly, the ability to represent smooth tempo transitions is desirable. The implementation for tempo mapping is discussed in the next section.

Chapter 5

The Implemented System

5.1 System components

The major conceptual modules of the current system are the subsystems for sensing, recognition, and audio rendering. The associated hardware configuration is illustrated in figure 5-1.

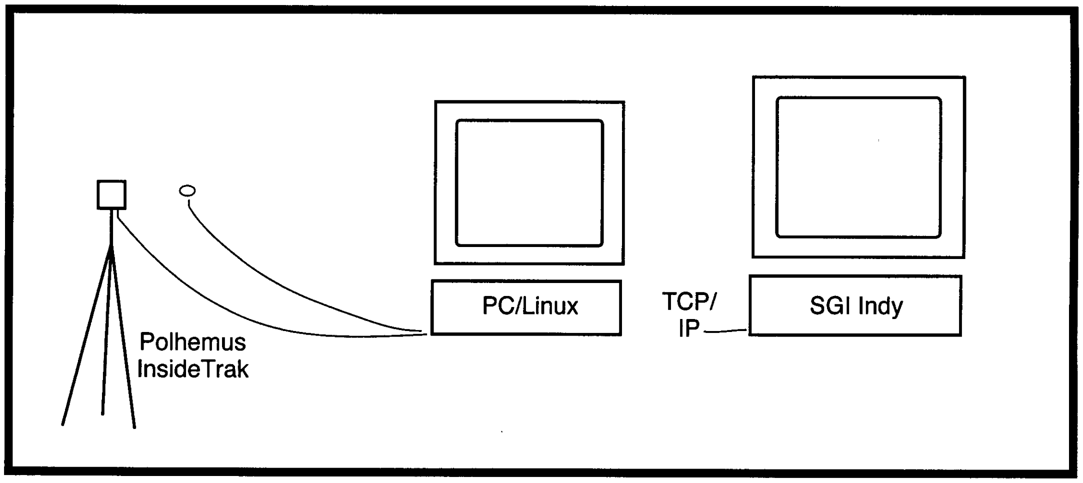

Figure 5-1: system hardware configuration. Polhemus data read by a PC is sent to an SGI Indy for analysis.

Sensing apparatus

The sensor used for recovering position data is a Polhemus InsideTrack(r), which attaches to a PC via the bus. Software provided by Polhemus to interface with the sensor has been modified for use under the Linux operating system. The Polhemus data is sent to a host machine via a TCP/IP socket for analysis.

The Polhemus installation consists of a base module, which represents the origin of the coordinate system, along with two sensors attached to long cables. The Polhemus uses magnetic field technology for position sensing, thus interference and singularities can be expected when using the sensor near large metallic objects or display monitors. Extremely satisfactory operation has been achieved in placing the base module on an aluminum tripod, with the sensors at least one foot away from metallic objects or electric fields. The default filtering algorithms shipped with the Polhemus

Recognition software

The heart of the system, the recognition software, runs in real time on the host machine. The software has been developed for an SGI Indy with R4400 200Mhz processor, and should be easily portable to other architectures, with changes allowing for machine-specific real-time execution.

Appendices A and B explain the operation of the software system, provide detailed structure and flow information , and provide functional descriptions of the primary source code modules.

Audio rendering

Csound for IRIX, written by Barry Vercoe, renders the audio output. Csound runs concurrently on the same host cpu as the recognition algorithms; real time music events are transmitted to Csound via an unbuffered pipeline.

5.2 Score representation (MIDI standard)

The MIDI standard as the score representation has been a satisfactory choice. Its widespread use assures that already-prepared scores are available to the researcher, and that the score file format is compatible with existing tools. MIDI metric-time score files are accepted as input; the internal score representation is based on MIDI as well.

There are two MIDI standards. One is for real-time use, and does not represent time explicitly. Instead, signals are transmitted at the real time at which they are intended to take effect. The second standard, the MIDI file format standard, explicitly represents time as intervals between events. This second format is used here to represent a musical score. Please see Appendix C for a quick reference to the MIDI standards.

MIDI serves many purposes well -- it is useful as a standard mode of communication between musical software and hardware. It has many shortcomings; for instance, the musical parameters it defines are strongly tied to the keyboard instrument model.

However, MIDI provides for extra control parameters -- "escape sequences" -- and is fairly extendible for any purpose. Inasmuch as data must be sent from one point to another, MIDI serves as a suitable wrapper for carrying any user-defined information.

The use of the MIDI standard allows access to the vast library of classical MIDI score files available on the Internet. However, several key shortcomings exist when considering these available files. Primarily, some of the desired score constructs are never present in standard MIDI files. The MIDI score file standard lacks representations for such indications as staccato, fermata, or crescendo. Instead, these constructs are typically present as variations in the low-level note information contained in the score. For example, a *crescendo* may simply be encoded by increasing note velocities over time. While automatic score analysis might yield extraction of higher-level notations including *crescendo* and *staccato*, such score analysis is not of primary interest in this work.

Since several key constructs are not explicitly present in standard MIDI score files, some human pre-processing is necessary to add project-specific score directions. In the present system, fermatas must be added to any scores requiring their use; entrance cues and exit cues are expected only at the beginning and end of the performance.

Incidentally, the real-time MIDI format is used to send note information to the CSound program. This is a practical choice as well -- CSound includes hooks for MIDI control of instruments, including time-varying parameters (known as "MIDI controllers").

5.3 Sensory data

As in all similar systems that I know about, the human conductor is modeled as a disembodied hand, represented as a vector in space: using the Polhemus sensor, the only conductor motion data available to the system is 6-degree-of-freedom position and orientation information. All decisions made by the conductor-following system are therefore based on these data alone, although primarily only two values -- the horizontal and vertical position -- are used.

5.4 Implementation of recognition

All of the recognition within this system is accomplished using a system of rules. This choice arose naturally when considering, for each class of recognized motion, how one might recognize that a given meaning is indicated. Again, all recognition is based on hard-coded rules; there is no learning in this system, and no adaptive feature selection.

5.4.1 Tempo mapping

The tempo mapping problem -- beat prediction

The simplest selection policy is to always use the same single detector. A very simple *switching* selection policy is to use for the next beat the predictor which resulted in the smallest error for the most recent beat.

The tempo mapping problem -- time warping

There are few subtleties in implementing the tempo mapping algorithm described above. The current velocity (tempo) is simply modeled as a linear segment between the last detected beat and the predicted beat. The predictions and tempo value are altered by setAnchor(), which is called whenever a new beat is detected. (Please refer to description of conductTiming.c in Appendix B).

5.4.2 Entrance cues

The algorithm for recognizing an entrance cue as described above is

1. A period of rest,
2. followed by an upward motion, which peaks
3. followed a downward motion, culminating in a trough.

which, when translated for implementation, becomes a representation consisting of an atRest() function and two time variables which represent the times of the peak and trough detections.

1. Wait until the function atRest() returns a positive result. The function atRest() returns true if, over the past second, the average distance between consecutive samples is less than .9 cm, and the maximum distance between consecutive samples is 3 cm.

2. When the rest state is left (atRest() returns to false), an upward preparatory motion is assumed. When a vertical peak is reached, if atRest() remains false, the time of the vertical peak is recorded, and the downward preparatory motion is assumed.

3. When a vertical trough is reached, if atRest() remains false, the time of the trough detection is recorded.

4. The relative time intervals of the gesture are then analyzed in order to reject irregular motion which is not intended as an entrance cue. If either the upward motion time interval or the downward motion time interval has duration longer than 60 per cent of the other, the gesture is rejected.

5. The indicated tempo is rejected if it falls outside a predetermined interval. Currently, the entrance cue is rejected if it indicates a tempo slower than .5 bps (beats per second), or faster than 5 bps.

Once an entrance cue is accepted, expressive performance information is extracted from the preparatory motion. Note articulation and dynamics are extracted as in the usual case for each note (discussed below).

5.4.3 Beat detection

Troughs in the vertical position data are considered as beat candidates. That is, a beat is detected by finding a peak in the second derivative of the vertical position. Beat candidates are verified or rejected, based on an assumption of how fast the tempo would then change if the candidate were accepted.

Some filtering of the detected beats over time is necessary to account for sampling error, and to diminish slight conductor errors. A very simple nonrecursive averaging filter is used for now.

Beat acceptance or rejection

As discussed above, an additional layer is necessary to reject "obvious" false positives. Here, any beat detections which would account for more than a 70% increase in tempo are rejected. Beat detections which would register a dramatic *slowing* of the tempo are another case -- in this situation, it is assumed that a beat gesture for a previous interval either went undetected or was not indicated at all. The detected beat is then matched to the nearest real-time beat of the performance.

5.4.4 Fermata

Implementation of the fermata recognition model is based on the simple rule algorithm:

- the conductor's hand motions are expected to be smooth (relatively constant curvature),
- followed by an "abrupt" termination gesture to signal the release of the held note.

Although standard deviation (of normalized distance between samples) is the measure for abruptness for note articulation, a simple threshold is used here to recognize the fermata "cut-off." That is, all current notes are held while the motion remains above a certain threshold, and the notes are released once the motion drops. Resumption of the piece is effected using exactly the same algorithm as for the entrance cue, except that no period of rest is expected before the preparatory motion.

5.4.5 Note articulation (staccato vs. legato)

For each note to be played, a measure of smoothness of motion is taken over the most recent samples. The value is scaled to the scalar articulation representation so that the result can be applied to the audio rendering process. The current measure of smoothness is the standard deviation of the normalized distance between the consecutive sample points in space -- a high standard deviation reliably indicates staccato motion. The sample distances are normalized so that the measure of note articulation is scalable in gesture size. The standard deviation measure is then scaled to the interval [0..127] for use with MIDI protocol.

5.4.6 Dynamics

Overall piece dynamics should be related to grandness of the conducting motion, and perhaps peak velocity of motion. Intra-measure dynamics should vary according to a model of the *pulse*, or phrasing structure which occurs during each measure to emphasize the hierarchical nature of the beat. Currently, dynamics are calculated on a per-note basis, and are derived from the mean value of the distance between samples within the last beat period.

Filters with histories of varied length may be useful in modeling local and global dynamics. Dynamic control through grandness of motion has been applied in the implementation. Additionally, dynamic information should be recovered through changing the baseline of beating, as well as through gestural cues using the non-dominant hand [17].

5.5 System issues

The executable process specifies that it is to receive non-degrading scheduling under IRIX (the default is that IRIX schedules smaller time slices as a process ages). A maximum "nice" priority is also set for the process. Both of these operations require that the process be run under superuser privileges.

Due to a delay of up to 160ms for transmitting Polhemus data from client to host over a TCP/IP socket, a known delay was injected into the position-recovery representation. A known delay of 200ms ensures that no time is spent waiting for sample data via the socket. Although tempo changes are calculated to occur at the real-time instant the beat was indicated, the calculation can not take place until 200ms after the indication.

In a similar vein, it is necessary to specify "no buffering" for the pipeline between the process and the Csound audio rendering process, in order to achieve instantaneous response time. The standard C command

```
setvbuf()
```

accomplishes this task.

5.6 Additional software modules

Scheduler

In order to maintain the real time nature of the system, all events are scheduled to occur at some real time, and then quickly return control to the scheduler. The scheduler is implemented as an associative list of time-function pairs. The earliest item in the list is selected, and if its scheduled time is in the future, the process sleeps for the appropriate delay, at which time the scheduled item's function is called. The scheduler does no scheduling on its own -- each function is responsible for rescheduling itself when necessary.

Scheduling of musical events

Only one musical event (note on, note off, etc.) is scheduled at any time. In scheduling an event, getSchedInfo() is called, which returns the real time at which the score event should occur according to the state of the tempo model. The musical event is scheduled for that time value. It is important to note that the pending musical event must be rescheduled whenever the parameters of the tempo model change. In the system as currently implemented, this results in a rescheduling of the pending music event each time a beat is detected.

Within the MIDI representation of the score, musical events that are to happen simultaneously are encoded with a time delay of zero between them. These zero-delay events are not rescheduled; instead, the event is handled instantly before returning control to the scheduler.

Chapter 6

Procedure And Results

6.1 Beat predictors

Several beat predictors run in parallel on the current system. A "selection policy" decides which predictor to use to predict the next beat. After experimentation with real data, a set of predictors and a selection policy have been chosen which have consistently performed well, based on a simple error function.

Comparison of predictors and selection methods

Two different beat predictors were built: "ConstantT," which assumes constant tempo, "ConstantA," which assumes constant acceleration (the change in tempo).

A performance was recorded by a novice conductor. The conductor's input was analyzed using three different beat prediction schemes, and compared for local and global prediction error. The data was recorded from a performance which used only the constant velocity assumption, and lasted for 75 beats.

Using only the constant velocity assumption resulted in overall prediction error of 490.2 ticks (a cumulative squared error of 7,849 ticks). Since there was no selection between prediction models, the ideal error, defined as the lower bound on the error resulting from choosing the ideal predictor at each beat, was also 490.2 ticks.

A second trial utilized both the constant velocity and constant acceleration models, with a selection policy of choosing the model which yielded the smaller error for the last detected beat. This scheme resulted in an error of 551.2 ticks (a squared error of 11,940 ticks).

The constant acceleration model assumption was chosen for 27.2% of the beats. It was noted that of the ten switches to this model, six were answered by an immediate switch back to the constant velocity model. The beats during which the constant acceleration was used only once (6 samples) resulted in an average penalty of 14.21 ticks when compared with the constant tempo model for the same beat.

Constant-acceleration model beats within strings of more than one constant-acceleration choice resulted in an average *improvement* of 3.2 ticks over the constant-tempo choice.

From these results, a new selection policy was constructed, which only selects the constant acceleration model when it has yielded the smaller error for two consecutive beats. For the trial utilizing this policy, the cumulative error was 492.7 ticks (8273 squared error).

Note that the data recorded for this analysis was performed using a single prediction scheme. Although the input data was consistent through the analysis, the manner in which the system played the score differed, resulting in clearly unreliable data. More reliable data should be gathered through recording many sessions under differing prediction schemes; a statistical analysis will account for variations between performances. Alternately, identical motion data could be used if the conductor did not listen to the audio during recording; however, this approach defeats the purpose of the recognition system.

It is concluded that selection policy plays a demonstrable role in improved beat detection; new analysis should be produced under the situation that the performance operates on the selection scheme being tested.

6.2 Tempo tracking robustness

Three trial performances were recorded from a novice conductor using the system for the first time. In each of the three cases, the beat was tracked until the end of the piece, with tempo ranging from .37 to 1.84 beats per second. In one of the performances, two detected beat candidates were rejected due to noise: these detections occurred at 17% and 3% of the time predicted for the next beat. In essence, the tempo tracking and beat detection algorithms provide a stable base for the rest of the system.

6.3 Entrance cues

Exactly one entrance cue is detected per performance. For three different performances by an novice conductor using the system for the first time, two intended entrance cues were rejected due to the time difference between the length of the preparatory motion and the length of the downbeat. In a separate trial, ten entrance cues were rejected during only two performances. For seven different performances by a novice conductor who was experienced with the system, only one intended entrance cue was rejected.

Practice with indicating entrance cues does appear to improve recognition. According to the motivation of natural gesture understanding, the recognition model for entrance cues should be re-evaluated based on experimental data.

6.4 Note articulation

As stated earlier, note articulation for each note is computed from the standard deviation of the normalized distance between samples, over the samples in the last beat. This value is then scaled into the range of MIDI controller values [0..127]. Resulting values from three performances by an untrained novice conductor follow.

The standard deviation measure ranged from .178 to .307 over the performances. The resulting mapping yielded note articulation values from 0 to 89 (on the MIDI 0-127 value scale).

6.5 Note velocity

Velocity (volume) for each note is calculated from the mean of the distance between samples. Resulting measures ranged from 6.4 to 34.8, mapping into the range 21 to 127 on the MIDI 127-value scale.

6.6 Scheduling error

The real-time scheduler is an important system consideration. As the main loop, most of the execution time is spent in the scheduler, sleeping until the next event. It is of course possible for the scheduler to fall behind, when a function does not return control in time for the next event (typically due to being scheduled out by the operating system). Normal operation under IRIX on the R4400 processor has shown scheduler performance to be reliable.

For scheduled sensor sampling events, a delay in scheduling results in a delay in response to the sample. However, the real time of the sample, which originated on the PC/Polhemus platform, is known (subject to scheduling error on the PC host, which is insignificant). Schedule delays for musical events, of course, result in perturbations in the audio output.

For a sample of 7,424 scheduled events occurring over 196 seconds from four different performances, the scheduler fell behind on 550 events, or 7.4% of all events. The mean delay was 6.5 msec, the maximum 250 msec, and the standard deviation 22 msec. When all scheduled events are considered for this data set, (including those events for which the scheduler did not fall behind), the mean delay falls to 0.5 msec, with a standard deviation of 6.3 msec. It is concluded that delays due to scheduling error are not significant for the performance of the current system, although rare maximum delays of 250 msec in the audio will most likely present a halting performance to the conductor [8]. Typical note-lengths range from approximately 100 msec to 1000 msec and longer.

6.7 Adherence to natural gesture

The criteria for adherence to natural gesture recognition for this system are that

- Only the motion of the conductor should be used as input to the system -- keyboard controls and switches should not be used, and
- Conductors using the system should not be required to learn specific gestures or trajectories for controlling system parameters. For example, requiring a conductor to draw a straight line near the top of the frame to indicate termination is unreasonable.

In these senses, the present system does make use of natural gesture.

Chapter 7

Conclusion

7.1 Summary

A computer system has been implemented which analyzes the motions of a musical conductor in real time. Playing notes from a classical score, the program augments the performance with timing, dynamics, and articulation information extracted from the motion of the conductor.

Specific performance constructs that the system responds to are beat-tracking and tempo, global dynamics, note articulation, the entrance cue, the fermata, and the cue to end the performance.

The problem is approached by first modeling the desired expressive musical characteristics, then developing a rule-based recognition model for the musical expressive model. Due to the nature of the problem, context from the score heavily drives recognition and detection of gestural events, more so for certain gesture classes which are called "context-expected."

As a beginning approach to the problem of beat predicting for tempo mapping, multiple prediction models have been considered, and a model selection scheme has been implemented which yields the smallest error under trial performances.

7.2 Contribution

The present work presents an approach to the problem of recognizing natural conductor motions. Natural motions and gesture have been defined here by the absence of any requirements for a conductor to learn trajectories specific to this particular implementation.

This paper suggests a modeling approach for the problem in which musical expression and gesture recognition are modeled as two separate realms. This model allows for, among other things, the decoupling of perceptual features from expressive effect. Making this explicit may serve as a conceptual aid for similar systems. A third model, that of the virtual performer, may be added based on previous researchers' work.

Some insight is provided into the contextual nature of the problem, by way of context-expected gestures and the context-sensitivity of gestures. The musical score is a context which prescribes a small set of expected gestural cues. Future work should address the extent to which a given gesture can be expected based on information in the score and the performance.

The performance features extracted from a gesture are not necessarily the same as those used to identify or detect the gesture. This concept is integral to the context-sensitive nature of the gestural cues in conducting, and may be applicable to many other recognition tasks.

Materially, the resulting real-time system exists as a framework for studying the conductor natural gesture recognition problem. As such, it could serve a small community as a platform for exploring models of conductor recognition, and models of expressive music for synthesis.

7.3 Suggested extensions

Envisioned extensions are presented here according to the level of the system which the extension would affect the most: the definition of the problem, the algorithms used in solving the problem, and the implementation of the algorithms.

7.3.1 Extensions to the problem definition

Musical expression model

A model which accounts for truly expressive and varied musical style can be a powerful tool for recognition. More responsive and interesting natural conductor gesture recognition systems must incorporate more musical constructs into the expressive model. Several key constructs can be considered for a starting point:

- the pulse, or the variation of note dynamics within a measure, reflecting the relative importance of each note in the rhythm.
- beat hierarchies, which represent timing information at finer resolutions than the quarter note. A conductor might desire, for instance, to direct each of the three notes in a triplet during a slow passage.
- expressive timing. Small deviations from the straight tempo of the piece play an important role in overall expressive style [2].

- multiple virtual performers which accept individual direction.

Virtual performer modeling

A global performance model has been assumed for the current work. It is desirable to formulate models for individual performers, which interact both with the conductor and with other performers. A successful performer model would allow for individualized direction by the conductor, and would also stimulate a higher quality performance experience.

Richer sets of features from conductor

In the future, additional features of the conductor's motion will conceivably be recovered, including position and orientation of the head, position of the end of the baton, orientation of grip on the baton, direction of gaze, motion of the shoulders, extent of the elbows, position of the torso, and so on.

Correlations and predictors within such a rich feature set might result in improved recognition models. Clearly, the associated musical expressive models should also be enriched to allow for communication of yet higher-level musical constructs.

Validation of "common denominator of styles" thesis

To what extent can this assumption, that some salient features are consistently meaningful from conductor to conductor, really respond to many amateur conductors? As the number of people who use the system increases, a generalized (user-independent) solution to the problem may not prove to be fruitful. Consideration should be made for adapting to an individual's style through rehearsal. Within the current framework, it is suggested that the expressive music model be developed independently of conductor styles, while the associated recognition models for a given conductor learn which features of motion to attend to.

7.3.2 Extensions to the approach

Experimental framework for training

Several distinct classes of learning are possible within this framework. The first is learning parameters for recognition of musical models for expression. Given conductor data which indicates a known musical intention, a learning approach could aid in selecting which of many features correspond to the intention. This may be useful for attempting to model some "humanly subjective" constructs such as "hesitation," "caution," or "power."

Likewise, a learning approach may indicate which features best predict an expected event. Work at CNMAT [10] has addressed this problem using neural networks for beat prediction; however, the approach used only the single feature of the vertical baton position.

A third learn method is to learn an individual conductor's interpretation of a specific piece; this can be called a "rehearsal" paradigm because it is similar to what happens when a conductor rehearses with an orchestra. Over several iterations, the conductor and players may both learn to expect cues and changes, resulting in a better communication of fluctuations in tempo and expressive parameters. One of Barry Vercoe's accompaniment systems demonstrated that virtual performers using this learning scenario are able to improve accompaniment tempo after several rehearsals [20].

Additional recognition models

The recognition model for note articulation currently contains a measure of the abruptness of acceleration in space. The model also incorporate a measure of the curvature of gesture trajectory. Staccato indications result in trajectories with relatively low curvature, containing sharp cusps (a high variance in local curvature). Legato trajectories are more fluid, producing trajectories with relatively constant curvature (having a lower variance in local curvature).

Position from video

A conductor-following system based on recognition from video is highly desirable. There are two broad classes of vision systems which may be developed for this problem in the near future.

The first class consists of algorithms which determine the 2-dimensional position of the hand, finger, or baton. A system which follows the centroid of a Gaussian hand-blob is attainable with existing software [24].

The second class of recognition algorithms are structure- or configuration-based. In particular, appearance-based classification of hand configuration appears very promising [23].

For computation vision, this addresses the issue of *seeing* versus polling for position data. Currently, it is often more feasible to get performance for a system or demo through "wiring" the actors, props, and scene of an experimental setup. The question arises, when is it practically feasible to *see* the scene rather than sense it through switches and position sensors? Clearly, there is much more useful information to be had through seeing, and using vision in such real-time interactive situations is now a thrust in parts of the computational vision community.

Currently, computer systems are still basically inept at working with real-time video, so that this too becomes an issue in any real-time interactive video analysis.

Selective feature adaptation

In the current system, all detection is based on hard-coded feature selection: all recognizers have been implemented as a set of hard-coded rules. If a ground truth can be provided for any task, selective feature adaptation might result in a better-tuned system.

For example, the lowest point on a beat trajectory is taken to be the moment of the beat. This approach has been taken by many of the existing conducting implementations to date. A delay in processing this information has been likened to similar delays in the human listening system [21]. Obtaining ground truth from an audio piano performance, for example, a researcher could frame an adaptive feature selection problem in which various trajectory features (based on energy, acceleration, etc.), best predict the performer's interpretation of the beat.

Examination of tempo-mapping problem

New approaches should model smooth velocity change vs. instantaneous velocity change, which both naturally occur in different parts of the same piece. This may be approached by either (a) learning of warping-maps for a particular musical piece, through a rehearsal paradigm [20], or (b) some demonstration of the viability of using different tempo assumptions under different conditions which are present in the context of the data and the score.

Higher timing resolution within the beat

Currently, anchor points (points which determine the tempo mapping) are set only when a beat is detected. It is desirable to develop recognition models which allow for anchor points to be set at smaller time intervals. For instance, a vertical peak in the vertical position of the hand could be taken as a back-beat, occurring midway through the beat. Additionally, beat prediction could be refined if timing information could be recovered from the approach to the beat indication. That is, motion just before the beat indication may be analyzed for telltale signs of an impending beat.

Notions of higher-level percepts

It will be useful to develop a perceptual definition of such concepts as “powerful motion,” or “relaxed beating.” Modeling instantaneous velocity changes should provide some insight into concepts relating to “forces” in the conductor’s motion. Higher-level percepts could then be built out of specific force features, or states of such forces over time.

For example, a powerful sweeping motion with the non-dominant hand (typically the left hand) could indicate an intensive change in volume if coupled with a certain pose of the hand, or it could add information to the beating motion when corresponding with motion of the dominant hand.

7.3.3 Extensions to the implementation

Fuller use of expressive model in recognition model

The expressive model for volume contains three guides for recognition (please see section 4.6). Currently, only one of the three guidelines is used; the other two should be added. Position recovery for the second conductor hand (the non-dominant) hand will be required to make use of all three guidelines.

Audio rendering

The CSound program provides a very powerful and flexible programming environment. Additional work should shape notes along dimensions such as "staccato/legato." Attention paid to this end of the overall system will quickly yield a much more realistic performance.

Chapter 8

Afterword

My impressions

I hope that anyone who reads this document will quickly become familiar with many of the important issues fundamental to the conductor-following paradigm.

I have become very interested in the symbiosis between recognition and modeling. Naturally, in this paradigm, recognition is driven by the complexity of the musical model at hand. Clearly, future advances in conductor-following systems will require more sophisticated models of musical expression, and more sophisticated models of how meaning is indicated through the conductor's motions. An exception to this is the features-from-video problem, which presents challenging problems directly related to recognition, and may eventually lead to a more powerful set of features which are not available through position-sensing technology.

For myself, this project has gravitated toward an interest in a general-purpose framework which can be expanded upon. Alas, early aspirations of tackling complex musical-expression models faded in order to complete the necessary, but satisfying, groundwork. This paradigm is certainly wide open at the current time; there are many possible specific research angles, covering a multitude of interests. I expect to continue my interest in this specific application of perception.

The idea of using such a system as an entertainment device has not left me -- I would very much like to see "armchair conductors" have a chance to express themselves.

The conducting system

The conducting system itself may soon be suitable for use as a "demo" within the MIT Media Lab. More quantitative results for recognition will be forthcoming, given the current framework as a guide.

It is possible that an interested research assistant or undergraduate researcher may use the system as a starting point for their own area of interest, which I encourage wholeheartedly. I particularly look forward to increasingly developed musical models from the community.

References

[1] G. Bertini, Carosi and Paolo, "Light Baton System: A System for Conducting Computer Music Performance," *Interface*, vol 22(3), pp. 243-258, August 1993.

[2] J. Bilmes, "Timing is of the essence: Perceptual and computational techniques for representing, learning, and reproducing expressive timing in percussive rhythm." S.M. Thesis, MIT Media Laboratory, 1993.

[3] A. Bobick, "Natural Object Categorization," Ph.D. Thesis, MIT.

[4] J. Cassell, M. Steedman, N.I. Badler, C. Pelachaud, M. Stone, B. Douville, S. Prevost, B. Achorn, ``Modeling the Interaction between Speech and Gesture" in *Proceedings of the 16th Annual Conference of the Cognitive Science Society*, Georgia Institute of Technology, Atlanta, USA, 1994.

[5] L. Campbell and A. Bobick, "Recognition of Human Body Motion Using Phase Space Constraints," Vismod Technical Report #309, MIT Media Laboratory, 1995.

[6] T. Darrell and A. P. Pentland, "Space Time Gestures," Vision and Modeling Technical Report #196, MIT Media Laboratory, 1992.

[7] Elizabeth A.H. Green, The Modern Conductor, 5th ed., Prentice Hall, 1992.

[8] S. Handel, Listening, p.70, MIT Press, Cambridge, Massachusetts, 1989.

[9] S. Intille and A. Bobick, "Exploiting Contextual Information for Tracking by Using Closed-Worlds," in *Proc. of the IEEE Workshop on Context-Based Vision 1995*, pp. 87-98.

[10] M. Lee, G. Garnett, and D. Wessel, "An Adaptive Conductor Follower," Center for New Music and Audio Technologies (CNMAT), Dept of Music, UC Berkeley, http://www.cnmat.berkeley.edu/conduct.html, August 1995.

[11] T. Marrin, "Toward an Understanding of Musical Gesture: Mapping Expressive Intention with the Digital Baton," S.M. Thesis, MIT 1996.

[12] M. Mathews, "The Conductor Program and Mechanical Baton," in Current Directions in Computer Music Research, eds. M. Mathews and J.R. Pierce, MIT Press, Cambridge, Massachusetts, 1989.

[13] H. Morita, S. Hashimoto, and S. Ohteru, "A Computer Music System that Follows a Human Conductor," *Computer*, vol 24(7), July 1991.

[14] H. Morita et. al. "A Computer Music Performer that can Follow a Human Conductor," manuscript obtained from Barry Vercoe.

[15] C. Pinhanez, "Computer Theatrer," Tech Report 378, Vision and Modeling Group, MIT Media Lab, 1996.

[16] R. Rowe, Interactive Music Systems, MIT Press, 1993.

[17] M. Rudolf, The Grammar of Conducting, 2nd ed, Schirmer Books, 1980.

[18] E. D. Scheirer, "Extracting Expressive Performance Information from Recorded Music," S.M. Thesis, MIT Media Laboratory, 1995.

[19] T. Starner and A. Pentland, "Real-Time American Sign Language Recognition from Video using Hidden Markov Models," in *Proceedings of ISCV,* 1995.

[20] B. Vercoe, Virtual piano player following human conductor; duet with flute, Experimental Music Studio demonstration, 1982.

[21] B. Vercoe, "The synthetic performer in the context of live performance," *Proc. Int. Computer Music Conference, 1984.*

[22] B. Vercoe and M. Puckette, "Synthetic Rehearsal: Training the Synthetic Performer," in *Proceedings of the 1985 International Computer Music Conferencee*, pp. 275-278. Computer Music Association, 1984.

[23] A. Wilson, "Learning Visual Behavior for Gesture Analysis," S.M. Thesis, MIT Media Laboratory, 1995.

[24] C. R. Wren, A. Azarbayejani, T. Darrell, and A. Pentland, "Pfinder: Real-Time Tracking of the Human Body," in *SPIE Photonics East* 1995, Vol. 2615 pp. 89-98.

Appendix A

User's Manual

The executable name for this system is 'conduct,' compiled for IRIX. Following is a brief description of the run-time options for 'conduct,' which are specified in the form of command-line options.

-m [filename]
(required) reads score from specified MIDI file

-d [filename]
debug output filename (default is stderr)

-SN
set samples-per-second from input sensor (default is 30)

-r [filename]
record input, save to specified file

-i [filename]
reads sample input from a file, instead of Polhemus

-M
read live input from the mouse, instead of Polhemus (sets sampling delay to 0 msec)

-x[dta]
disable interpretation of various features: d=dynamics, t=tempo, a=note articulation

-sN
set known sampling delay, in microseconds (default is 190,000, or 190 milliseconds)

several other options are available via the -help command-line option.

As evident from the command-line options, input may be read from one of three sources, live from the Polhemus sensor, live via the mouse, or off-line from a file. Position data may be saved to a file for reference. The current "debug" output, which may be redirected to a file as above, contains timing information for beat detection and prediction.

A trial run

'Conduct' has not been packaged for general consumption, therefore running a performance trial is more complicated than it needs to be.

Three programs must be executed: 'trak' on the PC/Linux, 'conduct' on the SGI, and csound on the SGI. 'Trak' sends its data to 'conduct' via a TCP/IP socket; use the command-line option -ip to specify an Internet host (on which 'conduct' will be run). e.g.

```
trak -ip 18.85.9.151
```

'Conduct' sends MIDI output to csound via a standard pipe. Therefore, 'conduct' and csound must be piped together from the command line. Note that stdin must be specified as an input for csound, and a csound orchestra file (e.g. filename.orc) is required as well. The only requirement for 'conduct' is that a MIDI input file is specified. e.g.:

```
conduct -m midi/airgstring.midi | nice -20 csoundNew -odac -d t.orc -M
      stdin
```

Note the two additional csound arguments: -d suppresses displays, and -odac sends audio output to the SGI digital-to-audio standard output port.

'Trak' must be running in order for 'trak' to make its socket connection. Once a connection is secured, the message "Accepted!!" appears in the debug output of 'conduct.'

Once a Polhemus socket connection is established, an X-window appears for plotting position samples. Depending upon your window manager, you might need to place the window quickly. Two-dimensional samples are plotted in red, and beat candidates (not necessarily detected beats) are plotted in yellow.

The output from Csound appears in the terminal window. This output is interwoven with debug output from 'conduct,' unless the -d option is used to redirect output.

'Conduct' may be terminated by using CTRL-C in the controlling terminal window, or with the right mouse button over the visual sample window. Under the current implementation, the Polhemus data connection is subsequently broken; both 'trak' and 'conduct' must then be re-started for another run.

Appendix B

Software source overview

A high-level code overview (section B.1) is followed by functional descriptions of the most important software models (section B.2).

B.1 High level overview

What follows is a concise description of the most prominent functions in pseudocode. This overview is intended to supply the structure and flow of the software.

```
I. main() -- main body
    (in file conductMain.c)
    readargs()
    init()
        see II below.
    scheduler()
        the "main loop." See III below.
    goodbye()
.

II. init() -- initialize variables and states
    give this process non-degrading scheduling time slices under IRIX
    renice process to run at a higher priority
    open the X window
    initialize the scheduler
        schedinit()
    schedule first sampling event
.

III. scheduler() -- the real main loop. Call events at their scheduled real-times.
    (in file conductSchedule.c)
    loop
        identifies the next scheduled event,
            (if none found, exit)
        sleeps until that time,
        runs that event (by calling the associated function)
.

Functions that are typically scheduled:
    getSample (see IV)
    midiSched (see V)
```

```
IV. getSample() -- obtain data sample, and incorporate into performance
    (in file **conductMain.c**)
    obtain data point from appropriate source
        (either live Polhemus, live mouse, or off-line file)
    add data point to buffer
    if recording data to file, output data point
    call timingNewSample(), for analyzing initial entrance
    call analyze() to see about updating timing information
        (see VI)
    draw 2D data point in X-window, conductXDraw()
    schedule the next sample event
.

V. midiSched() -- play a score event
    (in file conductMain.c)
    plays the next event in the musical score:
        determine articulation, based on std. deviation of recent data
        determine note volume
        call either midiNoteon or midiNoteoff as appropriate
    schedule the next score event
.

VI. analyze() -- look at new data sample, analyze
    (in file conductMain.c)
    consider new data point:
        if peak, call timing peak function
        if trough,
            consider this a beat indicator.
            call setAnchor() to establish now as the next beat time
                (please see VII)
            draw current data point in highlight color in X-window
            reschedule the pending score event, since the timing model has changed
                (this should really be done by setAnchor())
.

VII. setAnchor() -- consider now to be the next beat, change timing model accordingly
    (in file conductTiming.c)
    (see the section of thesis document which discusses the implementation of the timing
        model -- the model is based on the last known beat "AnchorTime," and the
        predicted next beat "PredictTime", with a constant velocity between the two)
    determine the "error" between now and predicted next beat
    setAnchor(now)
    Interval = duration(now,PredictedBeatTime(bestPredictionModel))
    TargetTime = now + duration(AnchorHistory(most recent anchor),now)
    set Velocity according to Anchor and Target
.
```

B.2 Function descriptions for software modules

Software modules

Modules described below:

conductMain.c main body, also containing some data analysis functions
conductMidi.c low-level MIDI output functions
conductMidiFile.c translates MIDI file into score data structure
conductOffline.c handles input from off-line data file, as well as recording to data file
conductSchedule.c scheduler at the heart of the real-time system
conductTiming.c model for score tempo mapping

Other modules:

conductTime.c methods for real-time data structures.
conductScore.c data structure representing the musical score.
conductSocket.c for communication with the Polhemus position sensor client.
conductPoint.c methods for Point data structure, to represent position data points.
conductX.c interface with the X-window position display.

Csound, written by Barry Vercoe. Csound and its accompanying manual are available from Barry Vercoe at the MIT Media Lab; older versions are available via anonymous ftp from sound.media.mit.edu.

trak, Polhemus interface software supplied by Polhemus, Inc. The DOS version of the software has been modified by Matthew Krom to run under Linux.

conductMain.c

```
main() {
        readargs();
        init();
        scheduler();
        goodbye();
}

/* Called only once; initializes states.
 */
initialize() {
        [initializes states]
        openXWindow();
        schedule(getSample());
}

midiSched() {
        for [all notes occuring this timestep] {
                [determine staccatoness for note]
                [play MIDI note]
                }
        scheduleNextMidiEvent();
}

getSample() {
        [use appropriate source to get next sample]
        [record sample to output file if specified]
        analyze();
        conductXDraw();  /* draw cursor in window */
        schedule(getSample(),nextSampleTime);
}
```

conductMidi.c

Low-level MIDI outputs, with wrappers for specific classes of events.

```
private midiFlush() {
        [flush MIDI output stream]
}

/* called if using socket MIDI output */
private writeSock() {
        [write character(s) to output socket stream]
}

private putmidi() {
        [write arguments to specified MIDI output stream]
}

public midiAllNotesOff() {
        [send 'all notes off' controller to MIDI stream]
}

public midiNoteon(), public midiNoteoff() {
        [send a note-on (note-off) to MIDI stream]
}

private midiController() {
        [send a MIDI controller value to specified MIDI controller]
}

public midiStaccato() {
        midiController(MIDI_STACCATO,value);
}
```

conductMidiFile.c

```
int Track; /* current track data is read into */

/* Return a variable-length value.  'variable' here does not refer to a storage variable, but
rather modifies the word 'length.'  A MIDI variable-length value is comprised of the low 7
bits of consecutive bytes, while bit 8 of each byte is high, terminated with a byte with bit 8
low.
*/
private getVariableLength() {
        while(bit_8_high(pointer)) {
                [put low 7 bits into value]
                pointer++;
        }
}

private newTrackScore() {
        Track++;
        [malloc space for Track data]
}

private addTrackScoreItem() {
        [store MIDI event info into current Track]
}

/* called-by readMidiFile
*/
private sortScore() {
        [using conductScore.c methods, create new score by zippering all tracks read]
}

/* called-by readArgs/conductMain.c
*/
public readMidiFile() {
        [read all Tracks sequentially from data file, using methods contained herein]
        sortScore();
}
```

conductOffline.c

For use with offline position data, both for reading offline samples and for writing live samples to a data file. openSampleFile() and recordInput() may not both be active in the same session (this would effectively copy a sample file!).

```
/* open a file for offline input.
*/
public openSampleFile() {
        open(filename);
        [fill sample buffer with file data];
        close(filename);
}

public usingOffLineFile() {
        return(InUse);
}

public getSampleOffLine() {
        return(next sample in data buffer);
}

/* open a data file for sample input.
*/
public recordInput() {
        store(outputFileName);
}

public usingRecordInput() {
        return(recordingInput);
}

public recordSample() {
        store(dataPoint);
}

public doneRecordingInput() {
        [save data buffer to outputFileName]
}
```

conductSchedule.c

Real-time scheduling routines. Position sampling and music playing are both schedule-driven real-time events.

```
/* data structure for scheduled events */
struct schedItem {
        . . .
        } schedList[...];

public schedule(function,time) {
        [schedule function to happen at time]
}

public unschedule(function) {
        [delete scheduled event for function]
}

/* called-by main()/conductMain.c.
this is the true "main loop". Will terminate if no events are scheduled.
*/
public scheduler () {
        while (true) {
                I = [find next scheduled event]
                sleep (until time of I)
                I.function()
                [delete I from event list]
        }
}

public schedClear() {
        [remove all scheduled events]
}

/* called-by init()/conductMain.c
*/
public schedInit() {
        [clear schedule table]
}
```

conductTiming.c

The tempo-mapping module. Maintains the tempo mapping representation, and answers the two questions, “given the current time, where exactly in the score is the performance?” and, “given a score event in the future, at what time will it occur under the current state?”

public initTiming()

/* return the real time at which the specified tick (score time) will occur
 */
public getSchedInfo(tick)

/* set now (the current real time) to be a known score beat point */
public setAnchor()

public timingNewSample()

/* represents the currently-used method for predicting the beat */
private int PredictMethod;

/* data structure for storage of previously-detected beat events */
private struct timeval anchorHistory[];
private addAnchorHistory(time);
private getAnchorHistory(offset);

Appendix C

MIDI Standards Quick Reference

Quick reference guide to event codes and file format

(note: Note articulation is transmitted via MIDI controller MIDI_STACCATO, as defined in conductMidi.c)

MIDI event codes

from "The USENET MIDI Primer", by Bob McQueer

status byte	meaning	data bytes
0x80-0x8f	note off	2 - 1 byte pitch, followed by 1 byte velocity
0x90-0x9f	note on	2 - 1 byte pitch, followed by 1 byte velocity
0xa0-0xaf	key pressure	2 - 1 byte pitch, 1 byte pressure (after-touch)
0xb0-0xbf	parameter (also, "controller")	2 - 1 byte parameter number, 1 byte setting
0xc0-0xcf	program	1 byte program selected
0xd0-0xdf	chan. pressure	1 byte channel pressure (after-touch)
0xe0-0xef	pitch wheel	2 bytes gives a 14 bit value, least significant 7 bits first

For all of these messages, a convention called the "running status byte" may be used. If the transmitter wishes to send another message of the same type on the same channel, thus the same status byte, the status byte need not be resent.

Also, a "note on" message with a velocity of zero is to be synonymous with a "note off". Combined with the previous feature, this is intended to allow long strings of notes to be sent without repeating status bytes.

The pitch bytes of notes are simply number of half-steps, with middle C = 60.

MIDI file format

text from the MIDI File Specification, by Dave Oppenheim
(notes by Matthew Krom)

(note: multi-track MIDI files are sorted into a single-track score representation, in module conductMidiFile.c)

A MIDI file always starts with a header chunk, and is followed by one or more track chunks.

```
MThd  <length of header data>
<header data>
MTrk  <length of track data>
<track data>
MTrk  <length of track data>
<track data>
```

The MTrk chunk type is where actual song data is stored. It is simply a stream of MIDI events (and non-MIDI events), preceded by delta-time values.

Some numbers in MTrk chunks are represented in a form called a variable-length quantity. These numbers are represented 7 bits per byte, most significant bits first. All bytes except the last have bit 7 set, and the last byte has bit 7 clear. If the number is between 0 and 127, it is thus represented exactly as one byte.

Here is the syntax of an MTrk chunk:
<track data> = <MTrk event>+
<MTrk event> = <delta-time> <event>

<delta-time> is stored as a variable-length quantity. It represents the amount of time before the following event. If the first event in a track occurs at the very beginning of a track, or if two events occur simultaneously, a delta-time of zero is used. Delta-times are always present. (Not storing delta-times of 0 requires at least two bytes for any other value, and most delta-times aren't zero.) Delta-time is in some fraction of a beat (or a second, for recording a track with SMPTE times), as specified in the header chunk.

<event> = <MIDI event> | <sysex event> | <meta-event>
(note: “sysex event” and “meta-event” are not covered in this appendix)

The header chunk at the beginning of the file specifies some basic information about the data in the file. The data section contains three 16-bit words, stored high byte first (of course). Here's the syntax of the complete chunk:

<chunk type> <length> <format> <ntrks> <division>

As described above, <chunk type> is the four ASCII characters 'MThd'; <length> is a 32-bit representation of the number 6 (high byte first). The first word, format, specifies the overall organization of the file. Only three values of format are specified:

0	the file contains a single multi-channel track
1	the file contains one or more simultaneous tracks (or MIDI outputs) of a sequence
2	the file contains one or more sequentially independent single-track patterns

The next word, <ntrks>, is the number of track chunks in the file. The third word, <division>, is the division of a quarter-note represented by the delta-times in the file. (If division is negative, it represents the division of a second represented by the delta-times in the file, so that the track can represent events occurring in actual time instead of metrical time. It is represented in the following way: the upper byte is one of the four values -24, -25, -29, or -30, corresponding to the four standard SMPTE and MIDI time code formats, and represents the number of frames per second. The second byte (stored positive) is the resolution within a frame: typical values may be 4 (MIDI time code resolution), 8, 10, 80 (bit resolution), or 100. This system allows exact specification of time-code-based tracks, but also allows millisecond-based tracks by specifying 25 frames/sec and a resolution of 40 units per frame.)

(NB: MIDI files which use the actual-time specification, instead of the metrical-time standard, are not of use to the present conducting system. Of the two, the metrical-time specification is analogous to the information contained in a classical music score.)