A Data-driven, Piecewise Linear Approach to Modeling Human Motions

Guodong Liu

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2007

Approved by:
Leonard McMillan
Carol Giuliani
Ming C. Lin
James Stephen Marron
Wei Wang
Bing Yu

ABSTRACT

GUODONG LIU: A Data-driven, Piecewise Linear Approach to Modeling Human Motions. (Under the direction of Leonard McMillan.)

Motion capture, or *mocap*, is a prevalent technique for capturing and analyzing human articulations. Nowadays, mocap data are becoming one of the primary sources of realistic human motions for computer animation as well as education, training, sports medicine, video games, and special effects in movies. As more and more applications rely on high-quality mocap data and huge amounts of mocap data become available, there are imperative needs for more effective and robust motion capture techniques, better ways of organizing motion databases, as well as more efficient methods to compress motion sequences.

I propose a data-driven, segment-based, piecewise linear modeling approach to exploit the redundancy and local linearity exhibited by human motions and describe human motions with a small number of parameters. This approach models human motions with a collection of low-dimensional local linear models. I first segment motion sequences into subsequences, i.e. *motion segments*, of simple behaviors. Motion segments of similar behaviors are then grouped together and modeled with a unique local linear model. I demonstrate this approach's utility in four challenging driving problems: estimating human motions from a reduced marker set; missing marker estimation; motion retrieval; and motion compression.

To
My Wife, Lan
My Daughter, Yanwei
and
My Son, Yanhui

ACKNOWLEDGMENTS

I would like to sincerely thank my advisor, Leonard McMillan for his guidance, help and support during the whole course of my dissertation research, especially the long hours he has spent before each of my paper submissions and before my dissertation defense. I would also like to thank my committee members, Carol Giuliani, Ming C. Lin, Stephen Marron, Wei Wang and Bing Yu for their constructive comments and suggestions.

I am grateful to Jingdan Zhang, who worked closely with me during the early period of my research and provided conceptual, geometric, algorithmic, and code contributions. My appreciation also goes to many people in the Department of Computer Science. In particular, I want to thank Whitney Vaughan for proofreading my dissertation; Tynia Yang for proofreading my I3D paper submission; Mary Whitton and Eric Bennett for their voices over on the videos; Qi Zhang for her work in my Sigmod demo.

I dedicate this dissertation to my dear wife, Lan Kong and to our wonderful twin babies, our daughter, Yanwei Carolina Liu and our son, Yanhui Shawn Liu. Lan's unconditional love, support, patience and belief in me have been an inspiration to me and kept me through this long journey. I owe her deeply for her tremendous contributions to my research work, let alone numerous sleepless nights she has endured in editing and proofreading my drafts. Yanwei and Yanhui, whom we have been so blessed to have, have brought me constant joy, happiness and reminded me *"What a wonderful world"*.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Motion capture, or *mocap*, is a technique for digitally recording movements for use in entertainment, sports and medical applications. A common form of motion capture uses optical sensing of strategically placed markers, and uses triangulation from multiple cameras to estimate the $3D$ position of each marker. Most often the marker positions are converted to joint angles for an assumed skeletal model.

Motion capture started as an analysis tool in biomechanics research (Winter, 2004) but has grown increasingly important as a source of motion data for computer animation as well as education, training, sports and recently for both cinema and video games. Motion capture saves time in these applications and creates more natural movements than manual animation. Mocap data are the primary sources of realistic human motions. As mocap techniques mature and more and more mocap data are generated, there is an imperative need for compressing, classifying, and searching ever-growing motion databases and reusing motion data.

In this dissertation I propose a piecewise linear modeling approach (PLM) to modeling human motions. PLM is a data-driven modeling approach. It exploits the redundancy and local linearity that human motion capture data often exhibit and models mocap data with a collection of local linear models. By doing so human motions can be sufficiently represented with fewer parameters while still retaining sufficient subtleties. This finding would open a door for more effective mocap systems, better ways of organizing large mocap databases, as well as more efficient methods for compressing mocap data.

In this chapter I will first discuss some existing human motion modeling approaches. Next I will describe the motivations and briefly review a few driving problems in motion estimation from a reduced marker set, missing marker recovery, motion data compression and retrieval. I then will propose my approach and briefly describe how it can be

applied to these driving problems.

1.1 Human Motion Modeling Approaches

Conventional human motion modeling approaches have been based on various human kinematic skeleton models whose complexity depends on specific motions and applications. Generally speaking, human kinematic skeleton models represent the human body as a hierarchy of the bones and joints that together form multiple kinematic chains from the root all the way to the end-effectors. Neighboring joints along a kinematic chain are connected by a fix-length, *i.e.* rigid bone. A pose can then be specified by a translation and rotation of the root bone and rotations of the rest joints. An articulated figure consists of a set of rigid segments connected with joints. Varying angles of the joints yield an enormous number of configurations. The forward kinematics problem takes these angles as inputs and results in a pose configuration of the figure. A more difficult problem is the inverse kinematics problem, which attempts to find the joint angles of a kinematic model given only the positions of the end-effectors and/or constraints. In a general case, there is no closed-form analytic solution for the inverse kinematics problem. However, inverse kinematics may be solved *via* optimization techniques. Certain special kinematic chains with spherical wrists permit kinematic decoupling. Kinematic decoupling treats the end-effector's orientation and position independently and permits an efficient closed-form solution. Solving for a inverse kinematics solution is generally computationally expensive, and the results are often unsatisfactory due to simplifications necessary to make the problem tractable.

An alternative approach for modeling human motions makes use of acquired data to construct sample-based models instead of parametric models in traditional approaches. Data-driven modeling approaches have seen their early successes in static scene modeling using image-based Rendering, *i.e.* IBR, as well as digital human shape modeling. There have also been methods that apply the principles of data-driven modeling in modeling human motions to synthesize realistic motions. These methods take advantage of the fact that mocap data are readily available and treat motion modeling as a database search and interpolation problem. Human motion sequences are organized into a motion database. New motions can be synthesized by querying the motion database with the given information and constraints as keys. These approaches drastically reduce the pressure to build highly accurate, robust and often inevitably complicated human motion models by making a direct use of the available motion database

equipped with efficient motion indexing and retrieving mechanisms. One drawback of these approaches, however, is the necessity of actively maintaining a motion database, a fact that may create scalability problems when a motion database grows too large to be held entirely in the memory. Another approach first assumes a human model from domain knowledge or prior information and then trains the model using the acquired data. However, these *strong* models could be a burden in some applications where such domain knowledge or prior information may not be easily available. These limitations make modeling approaches that assume much weaker priors more attractive. Researchers also realized that certain motions may lie in a manifold with fewer degrees of freedom. They attempted to find a low-dimensional (Safonova et al., 2004; Grochow et al., 2004) space using dimensionality reduction techniques such as principal component analysis, and they searched for a inverse kinematic solution in that space. However, most of these methods represent human motions with a global model and may not be successful in modeling large, heterogenous motion data. Alternatively, there is considerable evidence that human motions exhibit local linearity (Chai and Hodgins, 2005). It is conceivable that we may get a much more compact model if we model human motions with a collection of local linear models instead of modeling globally.

1.2 Data-driven Modeling Approaches

Traditional computer graphics mainly focuses on constructing analytical models, often parameterized. However, an inherent limitation of analytical models as well as physical simulations is that these models are often simplified to simplify calculations and often miss valuable subtleties of the real objects or natural phenomenon. On the other hand, as various data acquisition technologies have matured and data acquisition becomes faster and cheaper, we have begun to see interest in a new data-modeling approach which interpolates sampled measurements.

As advances are made for general data modeling methods, dimensionality reduction techniques, as well as the availability of huge amounts of human mocap data, it is now possible to consider data-driven modeling approaches that leverage the inherent realism of acquired data. Such a human motion modeling approach might be capable of deriving a compact but robust human motion model from mocap data with neither prior domain knowledge nor assumed skeleton models. As I will demonstrate in this dissertation, such a model may have advantages over existing human motion modeling approaches in some applications and may provide an attractive alternative to them

under certain situations.

1.3 Motivations

Although mocap data have been widely used in many applications, current motion capture systems do have limitations, which may prevent them from being used in some applications. The setup of most mocap systems is cumbersome, time-consuming and expensive. Frequently, some markers positions are missing due to occlusions and ambiguities. On the other hand, huge collections of mocap data are rapidly becoming available, and there is an immediate need for tools that analyze, compress, annotate and organize these data sets more efficiently.

There is considerable evidence that mocap data significantly over-specify the actual range of realistic human motions, and thus consistently exhibit redundancy and local linearity. The data-driven, piecewise linear modeling approach, which I propose in this dissertation, exploits the local linearity of human motions and model human motions as a collection of local linear models. For the scope of this dissertation, I concentrate on the following four driving problems:

1. Estimating human motions from a reduced marker set
2. Estimating missing markers from the available marker set
3. Human motion data retrieval
4. Compressing human motion data sequences

In the following subsections, I will give a brief overview on these four driving problems.

Estimating human motions from a reduced marker set

A common form of motion capture uses optical sensing of strategically placed markers. Most often the marker positions are converted to joint angles for an assumed skeletal model. Usually 40 to 50 markers are used to capture a motion sequence. As many as 300 markers may be needed to recover more accurate motions. Another example is capturing facial expression where more densely placed markers (as many as 100 small markers) are necessary to capture the subtleties of emotions.

Adding more markers in a mocap setup typically involves more mounting time and experience of the technicians. Moreover, adding more markers makes the mocap process more uncomfortable for the actors. More markers can introduce occlusion and ambiguity problems, which in turn demand many more cameras. Furthermore, there is a limit on how many markers can be placed, especially on a limited surface like the human face.

A cheaper and faster motion capture system would benefit many applications, such as computer games and virtual reality environments, where it is desired to have interactive, intuitive and accurate control over the characters/avatars. In these applications measurements from only a few markers can be effectively used as control signals. Instead of wearing a tight Leotard with many markers, a user may only need to wear normal clothing with only a few markers mounted on non-intrusive positions. Less mounting time also makes mocap feasible for more applications, since less overhead time is spent between users. Fewer marker measurements also reduce ambiguities during post-processing of mocap data, and thus require less human intervention. Mapping a small marker set to a dense mocap database also can be used to infer subtler motions.

Estimating missing markers from the available marker set

A common problem encountered in motion capture is that some marker positions are often missing due to occlusions or ambiguities. A marker is considered missing if it is not visible to at least two cameras. A major cause of marker missing is that a marker is occluded by props, limbs, bodies or other markers. Frequently, the positions of markers can be missing for a long period of time. Although many methods have been developed to handle the missing marker problem and are already in use in some commercial motion capture systems, most procedures require manual intervention and do not produce satisfactory results for abrupt motions, or for when a high percentage of markers are missing.

In order to avoid the missing marker problem, many mocap systems use many more cameras with hopes that each marker can be captured by at least two cameras, even though it is not guaranteed. When capturing motions that involve interactions of more than one person, a mocap system has to let the actors perform their respective motions individually and then merge these individual motions together. As a result an actor has to pretend interactions with imaginary partners. This motion capture strategy requires actors to perform naturally and precisely, and it also may introduce great difficulty in

stitching these individual motions.

Can we infer the missing marker positions from the other marker positions even under adverse situations when a considerable portion of the markers may miss for extended periods of time? Such a missing marker inference method would greatly improve a mocap system's robustness as well as making it possible to allow capturing multi-person interactions in a natural way, that is, capturing simultaneously at the same site.

Human motion data retrieval

Good realistic motions are costly to create. Such an expense often makes reusing of motion capture data through transformation and retargetting a more attractive option than creating new motions from scratch. As more and more human motion capture data become available, motion databases grow larger and larger. There is a need for the tools to organize large motion databases and support fast retrieval of similar motions.

In designing a scheme to organize a motion database that supports fast motion data retrieval, we have to define similarity among motion sequences. Motion sequences can then be compared and grouped by how similar or dissimilar they are to each other. For time series in general, similarity is often defined by the overall distance between the spatio-temporal curves under certain distance metrics. Various similarity metrics are used by many existing methods. These metrics can be loosely grouped into two categories: *numerical* similarity and *logical* similarity. Numerical similarity is often defined as a spatio-temporal distance between the time series curves under various distance metrics, such as Euclidean distance, Mahalanobis distance, etc. However, human perceptions do not always conform to numerical similarity. Motions may be perceived as similar even if they may not be quite close under certain distance metrics (Müller et al., 2005). For example, two walking sequences may be perceived as similar even they differ considerably in their speeds and styles. Logical similarity is used to reflect human-perceived similarity with a class of Boolean features to encode the geometric relationships between body parts. While logical similarity does incorporate the effects of human perception and are able to differentiate distinct motions, it largely discards all the numerical information. Consequently, it may fail to identify subtler differences among motions whose geometric relationships are similar. A similarity metric combining the strength of numerical similarity and logical similarity would greatly empower current motion retrieval methods with capabilities of differentiating various motions.

Compressing human motion data sequences

As more and more human motion data become available, the growing need for compact storage and fast transmission makes it imperative to compress motion data. For example, online video games often use motion data to interactively control the game characters from a remote site across the internet. It is desirable to quickly and reliably transfer these motion data over the internet with a limited bandwidth. So it is very important to develop an effective compression method to facilitate efficient storage and fast transmission of motion data.

There have been extensive studies on compressing time series data in general, and in particular images, videos and computer animation. However, there is still a lack of studies on compression methods that are tailored to compressing motion data. Although some general compression methods can also be used to compress motion data, these methods do not exploit the unique characteristics of motion data. As a result compressions with those methods are unable to achieve greater compression ratio while attaining sufficient details of the original motion data.

1.4 Overview of My Approach

There is sufficient evidence that human motion data exhibit local linearity. A short and simple motion typically accounts for a single behavior, and its poses often lie near a low-dimensional linear space. Motion data arising from similar motions can often be described by the same local linear model, which is valid for a limited range of articulation. A long and complex motion sequence can be considered as a concatenation of much simpler motion segments. In this dissertation I propose a data-driven, segment-based, piecewise linear approach to modeling human motions. This approach models human motion data as a collection of low-dimensional local linear models.

In this approach I first segment a motion sequence into simple motion subsequences of distinct behaviors and local linearities. These subsequences are referred to as *motion segments*. I apply the probabilistic principal component analysis (PPCA) for the segmentation. PPCA treats motion data as an ordered sequence of poses, and it segments the motion sequence where there is a local change in the distribution of the poses. In practice a multivariate Gaussian distribution is assumed for poses of a distinct behavior, and PPCA is used to estimate their distribution. Motion segments from the same behaviors should have similar Gaussian distributions and share the same low-dimensional

space. This segmentation technique can divide complex sequences into simple distinct and perhaps overlapping behaviors. So I apply a divisive clustering method to identify and group motion segments that can be represented in the same low-dimensional space, or in other words, by the same local linear model. Covariance matrix and mean vector can uniquely determine a Gaussian distribution. I therefore form a feature vector derived from the covariance matrix and mean to characterize each motion segment. I then use the feature vectors, one for each motion segment, as modeling primitives to construct a hierarchy of local linear models. Similar motion segments are partitioned into the same cluster that corresponds to a particular local linear model. A local linear mapping function is computed from the poses that belong to the model. Finally, different classifiers are trained to classify either poses or motion segments to the most appropriate local linear model. This piecewise linear modeling approach may benefit many applications. In the following I will briefly explain how I apply this approach to four driving problems.

To estimate human motions from a reduced marker set, I first select a small set of the most informative markers, i.e. *principal markers*. Then all the motion segments are modeled by PLM, where simple motion segments are grouped together and represented with a unique local linear model. For each local linear model, a linear mapping function is computed from the principal marker set to the rest markers. A classifier is also trained with only the principal markers as inputs to classify the poses to the most appropriate local linear model. Given a new frame with measurements of only the principal markers, I first classify it to a local linear model and then estimate the rest marker positions from the principal markers *via* the associated linear mapping function.

In missing marker estimation problem, any marker is likely to be missing for extended periods of time. My strategy is that, for each frame with an arbitrary set of markers missing, I use all the available marker positions to estimate the missing markers by exploiting the correlations among markers. I also segment sequences and then apply PLM to all the motion segments. Since I can not know what markers are missing until running time, I take a two-stage modeling approach in which I first use a global linear model to estimate coarsely the missing marker positions, which are refined later with a more accurate estimation from a local linear model.

I follow a similar PLM modeling pipeline for motion retrieval, except that it is unnecessary to model each cluster of segments because the purpose here is to search for similar motions. Furthermore, I construct a motion segment indexing scheme instead of a frame-based classifier and use the indexing scheme in the later motion querying

operation. In constructing the indexing structure, I generalize the OBB-Tree, an efficient data structure that has been used in computer graphics for ray-tracing, collision detection, etc., for the purpose of pruning. Given a new motion sequence, I segment it into motion segments whose feature vectors are then derived and used to retrieve similar motions from the database using the indexing scheme.

I have also adopted PLM in developing an efficient method to compress human motion sequences. Here the rationale is that, having observed the local linearity exhibited by most human motions, it is conceivable that we may get better overall compression results if we find those local linear pieces and compress each piece of motions individually. I first segment a motion sequence into subsequences, such that poses within a subsequence lie near a low-dimensional linear space. I then compress each segment by projecting the poses data onto that linear space using principal component analysis. I can achieve further compression by storing only the key frames projections to the principal component space and interpolating the other frames in-between *via* spline functions during the decompression.

1.5 Thesis Statements and Contributions

Thesis Statement

Human motions can be represented by a data-driven, piecewise linear model with a small number of parameters. I will show the utility of this model in human motion estimation, motion database retrieval and motion compression.

In this dissertation I present an alternative human motion modeling approach where motion data are described as a collection of low-dimensional, local linear models. Motions of similar behaviors are grouped together on the basis of segments and are modeled with a unique local linear model. I then apply the framework of this approach on four challenging driving problems to demonstrate its usefulness. Specifically, the contributions of this dissertation include:

- A framework of a data-driven, segment-based, piecewise linear modeling approach to human motions. The key components of this approach include: segmentation of motion sequences into *motion segments* of distinct behaviors with low dimensionality; characterization of various-length motion segments by their statistical distributions; hierarchical clustering of motion segments; local linear modeling methods to model similar motions; and Random Forest classifiers for classifying motion data.

- An efficient method to estimate human motions from a reduced marker set.
- An efficient method to infer missing markers from the available markers.
- An efficient indexing scheme for the retrieval of similar motions from a large heterogenous motion database.
- An efficient and high-quality compression algorithm for compressing human motion sequences.

1.6 Thesis Outline

In Chapter 2 I review previous work relevant to this dissertation. I then present a framework of my approach to modeling human motions in Chapter 3. From Chapter 4 to Chapter 7, I discuss how to apply this modeling approach to address the four driving problems. Particularly, in Chapter 4 I present a method to estimate human motions from a reduced marker set. I then address the missing marker problem from the available markers in Chapter 5. In Chapter 6 I discuss a method on efficient motion retrieval based on behavioral similarity. In Chapter 7 I describe an efficient compression method as well as a fast decompression scheme on compression/decompression of mocap data. In Chapter 8 I discuss the strengths and weaknesses of my approach when applied to the problems mentioned above as well as other potential applications. I conclude with a discussion of interesting directions that may be worth further exploration as future works.

Chapter 2

Previous Work

In this chapter I review previous work related to the research done in this dissertation. First, I review methods that apply the data-driven modeling principles in general computer graphics in section 2.1. In section 2.2 I generally review some existing human motion modeling approaches. I discuss motion segmentation methods in section 2.3. In section 2.4 I review some previous approaches to compression, particularly the methods on compressing motion data and animation meshes. In section 2.5 existing database retrieval and indexing methods are reviewed, especially some methods related to motion database retrieval. In sections 2.6 and 2.7 I review machine learning techniques. In particular, I discuss the dimensionality reduction techniques in section 2.6, including feature extraction and feature selection. In section 2.7 I discuss some methods on classification, regression and clustering.

2.1 Data-driven Modeling Approaches in Computer Graphics

The advances of data acquisition technologies as well as the explosive developments of computing power have facilitated the application of data-driven modeling approaches on various fields in computer graphics. For example, a convergence of computer graphics and computer vision has produced a set of techniques collectively known as image-based modeling and rendering (IBMR). Image-based modeling refers to the use of images to drive the reconstruction of $3D$ models (Kutulakos and Seitz, 2000; Debevec et al., 1996; Liebowitz et al., 1999; Cipolla et al., 1999). In image-based rendering, the goal is to achieve more realistic and faster renderings and to simplify the modeling task by using samples, as opposed to analytical models, as rendering primitives (McMillan

and Bishop, 1995; Levoy and Hanrahan, 1996; Gortler et al., 1996). Image-based approaches can potentially eliminate the labor-intensive task usually required for modeling detailed geometric structures. They can also handle subtle real-world effects captured by images but are difficult to reproduce with conventional graphics techniques. In this thesis I apply a similar philosophy to the problem of modeling human motion to make more realistic motions by incorporating actual measurements into the synthesis process. While IBMR uses images to generate novel views or to reconstruct $3D$ models, I use incomplete mocap information to infer the full motion configurations by exploiting redundancy and coherency among poses and mocap markers. In human motion modeling, I focus more on interpolation rather than extrapolation by using a collection of lower-dimensional, local linear models.

There have also been studies that adopted the data-driven approach in modeling human shapes. Among them, Magnenat-Thalmann and Seo (2004) proposed to automatically estimate intrinsic articulated structures of the human body using the range scan data from user-tagged landmarks. Researchers also attempted to apply the principal of data-driven approaches on modeling human motions, which I will discuss as a part of the next subsection.

2.2 Existing Human Motion Modeling Approaches

Motion capture data have been used extensively in animations, movies and interactive games. Arikan and Forsyth (2002) and Kovar et al. (2002a) constructed a motion graph from a corpus of motion capture data that encapsulated connections among the motion clips in the database. Motion can be generated simply by performing walks on the motion graph. Bregler (1995) used linear models to derive motion from photos. Li et al. (2002) modeled motion sequences as stochastic processes and modeled with *motion texture*, defined by a two-level statistical model: a set of motion textons at the lower level, and the distributions of textons at the higher level. Motion textons are defined as the repetitive motion primitives such as hopping, spinning and dancing. A motion texton is modeled by a linear dynamic system, while the texton distribution is represented by a transition matrix indicating how likely each texton is switched to another. Their algorithm learns motion textons and their relationships from the captured motions. The learned motion textures can then be used to generate new animations automatically and/or edit animation sequences interactively. In contrast, Pullen and Bregler (2002) proposed another type of motion texture by analyzing motion

capture data in the frequency domain. Their goal is to assist key frame animation where the basic contents of motions are provided by the key frames, while the subtle style and nuance can be added with a similar texturing in images. Their approach allows an animator to control an initial rough animation with key frames and then fill in missing degrees of freedom and details using the information in motion capture data. While both of these motion texturing methods can synthesize motions statistically similar to the real human motions, they lack the ability to infer human motion with measurable accuracy from incomplete information, such as positions from only certain markers. Our method, on the other hand, is capable of recovering realistic human motions pose-by-pose with high fidelity.

There are also methods that synthesize motions by interpolating between existing motions. Among them, Rose et al. (1998) described motions in a parameterized space, where the parameterized motions are called *verbs* and the parameters that control them are called *adverbs*. This verb/adverb scheme allows interpolating between example motions with substantial repertoire of expressive behaviors and great degrees of subtleties. Brand and Hertzmann (2000) constructed a Hidden Markov Model (HMM)-based statistical model, called style machine, that can generate new motion sequences in a broad range of styles just by adjusting a small number of stylistic knobs (parameters). Mukai and Kuriyama (2005) applied a practical technique of geostatistics, called universal kriging, for statistically estimating the correlations between the dissimilarity of motions and the distance in the parametric space. They interpolated motions based on spatial statistics by regarding all motion clips as spatial samples distributed in a multidimensional abstract space. Lee et al. (2002) proposed to preprocess a motion capture database for flexibility in behaviors and efficient search. Flexibility is created by identifying plausible transitions between motion segments, and efficient search through the resulting graph structure is obtained through clustering with inputs from simple user interfaces.

Researchers have also investigated the problem of inferring plausible human motions from a reduced measurement set. A common approach uses a small set of electromagnetic sensors to drive a virtual human model. Badler et al. (1993) tracked, in real-time, the position and posture of a human body using a minimal number of six DOF sensors to capture full body standing postures. They used four sensors to create a good approximation of a human operator's position and posture and mapped it onto an articulated computer graphics human model. The joints with no sensors attached are positioned by a fast inverse kinematics algorithm. Semwal et al. (1998) used eight

sensors augmented with information about natural body postures to infer full body human motions in real-time. These methods typically rely on inverse kinematics to estimate skeleton joint-angles from the constraints implied by the actual measurements (O'Brien et al., 2000). They depend on nonlinear solution methods and are prone to ambiguous solutions (Craig, 1989). As a result various optimization and smoothing criteria were introduced to resolve a unique solution. Moreover, the placement of the reduced set of markers in these methods is determined by trial and error. Our method offers a simple way to automatically select a set of the most informative markers and use them to infer the underlying human motions on a pose-by-pose basis without assuming human skeleton and other prior knowledge.

There is a significant redundancy in motion data due to spatial and temporal coherences in human behaviors. Safonova et al. (2004) and Grochow et al. (2004) demonstrated that specific simple motions can be accurately described *via* a low-dimensional parameterization based on dimensionality reduction techniques. Safonova et al. (2004) observed that most dynamic simple motions can be represented in spaces of five to ten dimensions. They presented a method to synthesize physically realistic human motions in low-dimensional, behavior-specific spaces. They showed that when the optimization problem was solved within this low-dimensional subspace, a user can specify the desired motion as an initial guess in a sparse, intuitive way. Grochow et al. (2004) proposed an inverse kinematics approach that applied a global nonlinear dimensionality reduction on human motion data. They used a Gaussian Process Latent Variable Model (Lawrence, 2004) to find a low-dimensional nonlinear manifold that can sufficiently describe motion data. Their approaches worked well with small homogenous data sets. However, for a large motion data set with various types of motions, a global modeling approach like theirs may be very slow and might not suit the data set well. I propose a piecewise linear approach to modeling human motions as a hierarchy of local linear models. My approach is better suited to describe a large heterogeneous motion data set. The local linear modeling approach for dimensionality reduction was considered by Bregler and Omohundro (1994), Hinton et al. (1994) and was further developed and applied on motion capture database by Chai and Hodgins (2005). Chai and Hodgins (2005) utilized temporal coherence of the control signals to accelerate the nearest neighbor search for similar poses and dynamically constructed a local linear model for the pose to be estimated. However, their method needs to actively maintain a motion database and might be ineffective when a database is very large. In contrast, I construct a compact model out of a motion database. My method does not need to

maintain a database after an offline modeling process, and it scales well with the size and heterogeneity of human motion databases.

The missing marker problem is commonly encountered in marker-based motion capture systems. Interpolation methods (Guo and Robergé, 1996; Rose et al., 1998; Wiley and Hahn, 1997) can effectively estimate the missing marker if a marker is missing for short periods of time, typically less than 0.5 second (10-30 frames). Some commercial mocap systems such as Peak and Vicon also provide missing marker recovery solutions based on various spline interpolation techniques, coupled with the use of kinematic information and skeleton models. Kalman filters (Dorfmller-Ulhaas, 2003) have also been used to predict the missing markers in current frame with the available temporal information. These methods, however, can become ineffective quickly and often require manual intervention when markers are missing for a long period of time, or are missing from the very beginning.

Herda et al. (2000) used human skeleton and marker positions from the immediately previous frames to predict the missing markers. If only a few isolated markers are missing over an extended period of time, their positions can still be inferred from the neighboring markers that share the kinematic relations with the missing markers. However, an accurate skeleton information must be assumed *a priori* in order to apply this method. Hornung and Sar-Dessai (2005) proposed to utilize more markers in a mocap set up and assemble neighboring markers into a rigid clique. Markers in a clique have fixed inter-marker pairwise distances. When a marker is missing, its position can be recovered through the distance constraints imposed by the markers within the same clique. This approach may become infeasible if a significant portion of the neighboring markers are missing so that the clique is unable to be formed from the available markers.

2.3 Motion Segmentation and Characterization

Many methods have been developed on segmentation of various types of data. Computer vision-based approaches to recognition, tracking and segmentation of high-level behaviors are widely used (Fleet et al., 2000; Oliver et al., 2002). There also have been extensive studies on segmentation of video sequences. Among them, Brand and Kettnaker (2000) used entropy minimization to construct Hidden Markov Models (HMMs), with high-level behaviors being mapped to states of the HMMs. Zelnik-Manor and Irani (2001) applied clustering methods based on distance metrics developed over a variety of temporal scales. Peyrard and Bouthemy (2002) proposed an online seg-

mentation method based on considerations of information loss. Motion data is general simpler than video data. These methods are either too complicated for modeling motion data (Brand and Kettnaker, 2000; Peyrard and Bouthemy, 2002) or may rely on the presence of the hand-annotated data, while I am interested in segmenting motions based only on the information available from the current motion sequence.

Methods have also been developed particularly for segmenting motion data. In particular, there have been some successes in motion segmentation at low levels. Rose et al. (1998) segmented a motion sequence into simple motion strokes characterized by the abrupt changes of velocities during transitions in order to retrieve example motions. Fod et al. (2002) segmented motion data by detecting zero crossings of angular velocities. Li et al. (2002) explored a more sophisticated technique where low-level segments (textons) were represented as the output of linear dynamic systems. A segmentation of motion is also implicit in a state machine or motion graph representation (Brand and Hertzmann, 2000; Kovar et al., 2002a; Arikan and Forsyth, 2002). I on the other hand are interested in segmenting motion sequence into different behaviors. Barbič et al. (2004) presented three methods to automatically segment a motion sequence into distinct behaviors. I implement their Probabilistic PCA method which creates segments using a probabilistic model of motions and has shown better performance in segmenting human motion sequences.

In characterizing human motions, Forbes and Fiume (2005) used a representative pose to characterize a motion clip. But it is often inadequate to use a single pose to summarize a whole motion sequence. It is also challenging to choose a representative pose for each motion sequence. Keogh et al. (2004) used a low-bound box that bounds the high-dimensional spatio-temporal curve of a motion segment to summarize this motion segment. I characterize a motion segment by summarizing the distributions of the poses that belong to the motion segment, and I think it would be more effective to characterize a motion sequence by their statistical properties than the low-bounding boxes. In particular, I derive an equal-length feature vector from the mean vector and the covariance matrix of each motion segment.

2.4 Motion Compression

Many techniques have been developed to compress various types of data (Salomon, 2000). Recently there have been more studies on compression of animation mesh data due to an increasing popularity of animation in movies and video games (Lengyel,

1999; Alexa and Müller, 2000; Gupta et al., 2002; Ibarria and Rossignac, 2003; Guskov and Khodakovsky, 2004; Karni and Gotsman, 2004; Sattler et al., 2005).

A typical animation mesh has at least thousands of vertices with fixed connectivity among vertices. An animation mesh data sequence consists of the vertex positions for each frame/mesh. To exploit the spatial correlations among animation vertices, Alexa and Müller (2000) constructed a compact representation of an animation sequence by applying PCA on a whole animation sequence. Karni and Gotsman (2004) further compressed the PCA projections of frames with linear predictor coding (LPC) to exploit the temporal coherence. Both of these methods attempted to construct a global linear model, while human motion data are in general nonlinear and especially exhibit local linearity. Simple motions in fact lie near a linear subspace with much lower dimensionality. Global PCA in compression of motion capture data tends to require more principal components to be retained and thus may be unable to achieve a higher compression rate. In contrast, my method starts with motion segmentation to identify the local linearity of a motion sequence and then compresses each motion segment individually. Ibarria and Rossignac (2003) introduced a Dynapack algorithm that exploits inter-frame coherence and used two predictors to encode the mesh motion. I exploit the temporal coherence using the spline interpolations to further compress the PCA projections of the frames.

Lengyel (1999) proposed the decomposition of a mesh into subparts and described these parts as rigid-body motions, while only a heuristic solution to segmentation was provided. Sattler et al. (2005) proposed a method for compressing animation sequences based on clustered principal component analysis (CPCA). They considered the trajectory of a vertex as a data point and clustered the trajectories into parts that moved almost independently. These mesh parts could then be compressed separately by PCA. In compressing motion data, Arikan (2006) proposed a method that clusters the features, i.e. *virtual markers*, into groups and compressed the features in each group separately. All of these three methods attempted to exploit the spatial correlations among features through a whole sequence. They broke down a sequence into simpler parts of correlated features and fitted them with low-dimensional linear models for compression. However, a motion sequence also exhibits local low-dimensional linearity at different segments, i.e. subsequences. So I take a different approach from those methods and divide a long and complex sequence into segments along the temporal axis. Since motion capture data sequences are usually much longer than the animation mesh data sequences, and there are typically fewer markers in a motion sequence than

the vertices in the animation mesh, my approach is more suitable for compression of motion data.

2.5 Motion Database Retrieval and Indexing Methods

Human motion data are a special instance of high-dimensional time series. It can be handled the same way as the other multi-dimensional time series data (Das et al., 1997; Faloutsos et al., 1994; Lee et al., 2000). In general, time series data are treated as multi-dimensional curves in the spatio-temporal space. Similarity is defined as the closeness between the time series curves under specified distance metrics such as Euclidean distance.

An indexing structure is desired for effective motion retrieval. There have been extensive studies in the database community for efficient indexing schemes on a variety of databases. R-tree has been one of the most popular indexing methods, and it has spawned many decedent methods. The original R-trees proposed by Guttman (1984) are hierarchical data structures based on B^+-trees. They are used for the dynamic organization of a set of d-dimensional geometric objects, represented by the d-dimensional minimum bounding rectangles, i.e. MBRs. Many variants have been proposed to make the method more effective or tailored to different databases. R^*-trees and R^+-trees are two of the popular ones. Multi-dimensional data can be indexed by R-trees and many of its variants. However, most multi-dimensional indexing structures have poor performances when dimensionality is greater than 8-12. A dimensionality reduction method must be applied to transform high-dimensional data into lower-dimensional spaces. Faloutsos et al. (1994) introduced the framework of GEneric Multimedia INdexing method (GEMINI) which can exploit any dimensionality reduction method to enable the indexing. The technique was originally introduced for time series but has been successfully extended to other types of data. They provided the *lower bounding lema*, sometimes also called *contractive property*, that guaranteed no false dismissal.

A common drawback of distance-based metrics is that they are sensitive to distortions at temporal axis as well as noise and outliers. This prevents direct application of existing methods to human motion data. Dynamic Time Warping (DTW) technique, which partially addresses the temporal distortion problem by local scaling, can be used to align time series before the calculation of distances (Berndt and Clifford, 1994; Yi

et al., 1998; Kim et al., 2001; Keogh, 2002). Keogh (2002) proposed a lower bounding technique to make DTW fast enough to retrieve time-warped time series in large databases. Longest Common Subsequence method (LCSS) (Das et al., 1997; Vlachos et al., 2002), as an alternative to DTW, further matches two sequences by allowing not only the sequences to stretch at the time axis, but also allowing some elements to be unmatched. It has been used to perform motion retrieval in Keogh et al. (2004). Some other methods (Chan and Fu, 1999; Faloutsos et al., 1994) have also been proposed to transform high-dimensional time series onto another parameterized space such as frequency domain using discrete Fourier transform (DFT) or wavelet transform, and to model the time series in that space. However, these methods may be ill-suited to non-cyclic patterns.

Kovar and Gleicher (2004) developed a DTW-based indexing structure, termed *match web*, to automatically search for similar motions in a motion database and then used them as intermediaries to find more distant motions. I construct a hieratical tree of motion segments, with similar motion segments at the same or the neighboring leaf nodes. My indexing scheme is a generalization of OBB-Trees (Gottschalk et al., 1996) to the high-dimensional feature space. OBB-Trees have been used in computer graphics community for ray-tracing, collision detection, etc. An OBB, i.e. Oriented Bounding Box, is a rectangular bounding box in a $3D$ space oriented in such a way that it would generate the tightest bound to enclose a spatial object. A large and complicated object can be decomposed into a hierarchy of pieces. At each level of decomposition, each piece of data is bounded by an OBB. The resulting hierarchy of OBBs is called a OBB-Tree. OBB Trees achieve tighter bounding box than the regular Minimum Bounding Rectangles.

Forbes and Fiume (2005) presented a method for quickly searching long, unsegmented motion clips for subregions that most closely match a short query clip. Their search algorithm is based on a weighted PCA-based pose representation in the joint angle space. Conventional PCA-based approaches don't take into account the hierarchical nature of the pose data represented by joint angles. Weighted PCA addressed this issue by putting more weights on more important joints. However, it is very much an art to assign the weights to all the joints in the joint hierarchy. In querying for similar motion clips, their query strategy is to select characteristic points from a query motion clip and use that point to search in the motion database. This strategy attempts to summarize a motion clip with a single pose. However, it is inadequate and perhaps unreliable to use only one pose to represent an entire motion clip. It also put great

pressure to choose a right characteristic point since it is vital to the query process.

The aforementioned methods only measure the numerical similarity which may not reflect the way how human perceives similarity among motions. Müller et al. (2005) discussed a notion of logical similarity. They introduced a class of Boolean features expressing geometric relationships between certain body points of a pose, with examples such as whether the right foot lies in front of or behind the plane spanned by the left foot, the left hip joint and the center of the hip. They divided a motion sequence into an ordered list of simple strokes, during which a set of geometric relationships remained invariant. Transitions through a particular path of geometric features were used as the signature to effectively encapsulate qualitative and logical information of the motion sequences. Their approach marked a new attempt to evaluate human-perceived motion similarities. However, useful quantitative information was lost during the encoding process. At the same time, their model may still contain redundant, low-level information. For example, a cyclic motion such as walking has to be segmented into many strokes, one for each foot step, despite the fact that those foot steps may look alike. In addition, defining a set of features manually is difficult and requires human intervention.

2.6 Dimensionality Reduction Techniques

Dimensionality reduction is commonly used in machine learning to deal with a high-dimensional space of features. The original feature space is mapped onto a new, reduced-dimension space and the examples are represented in that new space. The mapping is usually performed either by selecting a subset of the original features, i.e. *feature selection*, or by constructing some new features, i.e. *feature extraction*. Feature extracting methods can be divided into two categories: linear methods and nonlinear methods. Linear methods find a linear mapping from the high-dimensional space to the low-dimensional embedded linear space. Nonlinear methods assume that high-dimensional data of interest lies on an embedded non-linear manifold within a higher dimensional space. High-dimensional data can then be mapped onto these low-dimensional manifolds *via* various nonlinear mapping functions.

Common linear methods include independent component analysis (ICA) (Hyvarinen et al., 2001), principal component analysis (PCA) (also called Karhunen-Love transform KLT) (Jolliffe, 1986) and singular value decomposition (SVD) (Press et al., 1986). Principal components analysis is arguably the most widely used linear

feature extraction method. It is a linear transformation that transforms the data to a new coordinate system, such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. PCA has been used in some algorithms developed in this dissertation.

Nonlinear methods include kernel PCA (Schölkopf et al., 1998), multidimensional scaling (Cox et al., 2000), Gaussian process latent variable models (GPLVM) (Lawrence, 2004), Isomap (Tenenbaum et al., 2000), locally linear embedding (LLE) (Roweis and Saul, 2000), etc. Kernel PCA provides a non-linear PCA through the use of kernel functions. Gaussian process latent variable model is a probabilistic non-linear PCA. Like kernel PCA, GPLVM uses a kernel function to form the mapping in the form of a Gaussian process. However, in GPLVM the mapping is from the embedded space to the data space, whereas in kernel PCA it is in the opposite direction.

Multidimensional scaling (MDS), Isomap and locally linear embedding are proximity matrix-based methods where data are presented in the form of a dissimilarity matrix or a distance matrix. Isomap builds on classical MDS but seeks to preserve the intrinsic geometry of the data as captured in the geodesic manifold distances between all pairs of data points. The crux is estimating the geodesic distance between faraway points given only input-space distances. For neighboring points input-space distance provides a good approximation to geodesic distance. For faraway points geodesic distance can be approximated by adding up a sequence of *short hops* between neighboring points. These approximations are computed efficiently by finding shortest paths in a graph with edges connecting neighboring data points. LLE computes a different local quantity. It calculates the best coefficients to approximate each point by a weighted linear combination of its neighbors, and then tries to find a set of low-dimensional points. These low-dimensional points can be linearly approximated by their neighbors with the same coefficients that were determined from the high-dimensional points. Both Isomap and LLE are simple to implement, have a small number of free parameters and tend not to trap in local minima like many other popular learning algorithms. Also both yield impressive results on artificial and real data sets.

Previous approaches to feature selection fall into one of three basic categories: the wrapper method (Kohavi and John, 1997; Ng, 1998), the filter method (Liu and Motoda, 1998; Hall, 2000; Dash et al., 2000; Yu et al., 2003; Yu and Liu, 2003) and hybrid methods (Tsamardinos and Aliferis, 2003). A wrapper method typically requires a predetermined data mining or modeling task prior to feature selection. A filter method,

on the other hand, relies solely on the characteristics of the data without assuming any predefined task. Algorithms in this category often have lower computational complexity than the wrapper method. Filter methods can be further categorized into two groups: feature ranking (Hall, 2000) and subset searching (Dash et al., 2000). Feature ranking algorithms order the importance of individual features by assigning a weight to each feature.

Cohen et al. (2002) proposed a hybrid method that they called principal feature analysis (PFA). They first applied PCA on the data set and identified a set of principal components that effectively decorrelate the feature set. Feature coordinates can be determined by projecting each feature onto the eigenvectors of this decomposition. Finally, a set of the most informative features are selected in the eigenspace. PFA has comparable performance to PCA but selects a set of original features with a more intuitive interpretation than the principal components derived in PCA. Building upon PFA, I design an algorithm capable of selecting a set of principal markers in addressing one of the driving problems.

2.7 Classification, Regression and Clustering

Supervised learning and unsupervised learning are two important machine learning techniques. Supervised learning is used to create a function based on training data. The training data consist of pairs of input objects (typically vectors) and desired outputs. The output of the function can be a continuous value (called *regression*) or can predict a class label of the input object (called *classification*). Unsupervised learning is distinguished from supervised learning by the fact that there is no *a priori* output. In unsupervised learning a data set of input objects is gathered.

Linear and nonlinear regression are two types of regression techniques. If the relationship between the variables being analyzed is not linear in parameters, a number of nonlinear regression techniques may be used to obtain a more accurate regression. While linear regression models can be built in linear time, constructing nonlinear regression models in general remains as a challenging problem, especially for high-dimensional data. Piecewise linear regression (Ferrari-Trecate and Muselli, 2002; Iwai et al., 2002; Torgo and Costa, 2003; Vijayakumar and Schaal, 2000) has been used as an alternative to approximate the nonlinear surface within certain error tolerance using a collection of local linear models. The underlying principle is that any smooth function can be well-approximated by a low degree polynomial in the neighborhood of any point. The

core kernel of this method is to learn the local linearity.

Commonly used classification approaches include k-nearest neighbor, decision trees, support vector machine, neural networks, Bayesian networks, etc. Random Forest (RF) (Breiman, 2001) is a powerful classification tool that displays outstanding performance in regard to classification error. RF grows and combines many decision trees into predictive models. The overall prediction is determined by voting over all the trees in the forest and choosing the class with the most votes. Since the trees are generated randomly and independently, there is no risk of overfitting for large numbers of trees. I choose the RF classifier over other popular classification methods as a key component of the PLM pipeline, because RF has shown excellent classification results and seems to be a good fit for classifying human motion data.

As a form of unsupervised learning, clustering (Jain et al., 1999) is the partitioning of a data set into subsets (clusters), so that the data in each subset share some common traits - often proximity according to some defined distance measure. Data clustering algorithms can be hierarchical or partitioning. Hierarchical algorithms find successive clusters using previously established clusters, whereas partitioning algorithms determine all clusters at once. Hierarchical algorithms can be agglomerative (bottom-up) or divisive (top-down). Agglomerative algorithms begin with each element as a separate cluster and merge them into successively larger clusters. Divisive algorithms begin with the whole set and proceed to divide it into successively smaller clusters. In constructing a model hierarchy, I adopt the divisive clustering approach for constructing a model hierarchy.

K-means clustering method (MacQueen, 1967) clusters objects into k partitions based on attributes. It is a variant of the expectation-maximization algorithm in which the goal is to determine the k means of data generated from Gaussian distributions. It assumes that the object attributes form a vector space. The objective is to minimize total intra-cluster variance. The algorithm starts by partitioning the input points into k initial sets, either at random or using some heuristic data. It then calculates the mean point, or centroid, of each set. It constructs a new partition by associating each point with the closest centroid. Then the centroids are recalculated for the new clusters, and the algorithm repeats by alternate application of these two steps until the points no longer switch clusters. The algorithm has remained extremely popular because it converges extremely quickly in practice. I use k-means as my splitting algorithm to partition the data points at each branch into two subsets.

Chapter 3

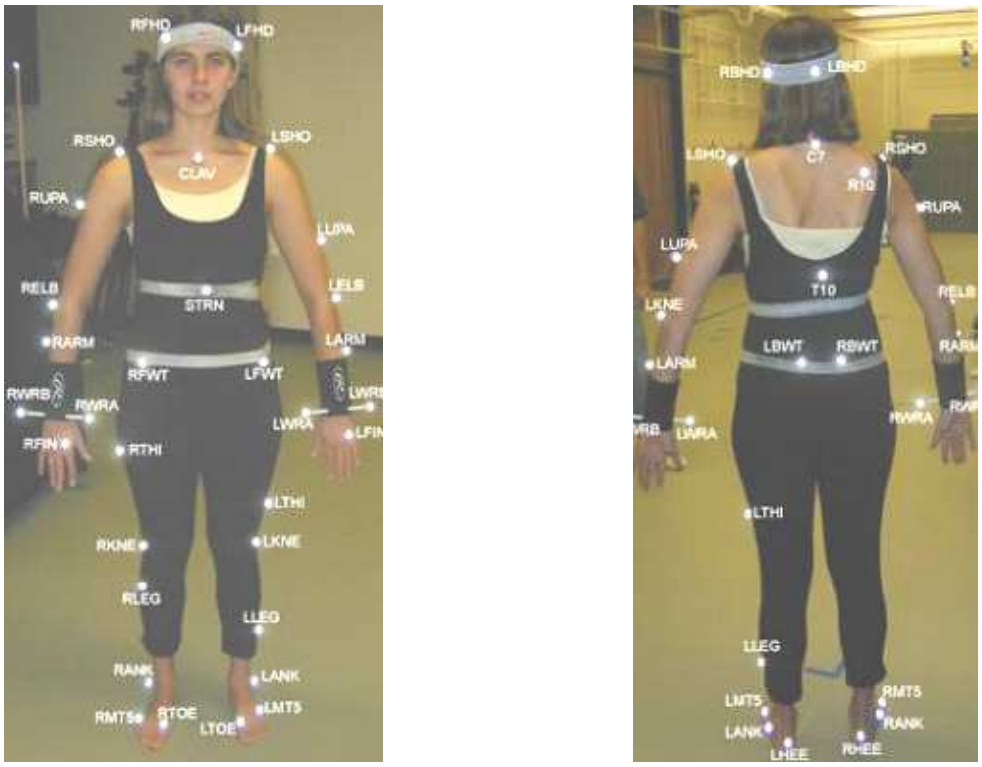
Data-driven Piecewise Linear Modeling Preliminaries

Human motion data exhibit local linearity. A short and simple motion typically accounts for a single behavior, and its poses often lie near a linear space with a much lower dimensionality than the original space. Motion data arising from similar motions can often be described by the same local linear model, which is valid for a limited range of articulation. A long and complex motion sequence can be considered as a concatenation from much simpler motion segments. I propose a data-driven, segment-based, piecewise linear modeling (PLM) approach to finding and grouping motions of similar behaviors in their inherent low-dimensional linear spaces.

In section 3.1 I review commonly used mocap techniques and popular mocap data representations. In section 3.2 I present an overview of my approach. From section 3.3 to 3.8, I discuss the key PLM components. In particular, in section 3.3 I describe the normalization of motion data as a preprocessing step. In section 3.4 I discuss segmenting motion sequences into distinct behaviors. In section 3.5 I explain the characterization of motion segments by their means and covariance matrices. I then present a hierarchical clustering method to group simple motion segments in section 3.6. In section 3.7 I describe how to construct a local linear model out of the motions grouped together. Finally, in section 3.8 I discuss training the classifiers used in classifying poses into the most likely local linear models.

3.1 Motion Capture and Representation

Motion capture, or *mocap*, is a technique of digitally recording movements for entertainment, sports and medical applications. A performer wears a set of markers at



(a). Front (b). Back

Figure 3.1: Motion capture marker placement guide. Courtesy of Carnegie Mellon University's Graphics Lab.

each joint: acoustic, inertial, LED, magnetic or reflective markers, or combinations of any of these, to identify the movements of the joints. Sensors track the positions or angles of the markers. The motion capture computer program records the positions, angles, velocities, accelerations and impulses, providing an accurate digital representation of the motion. Motion capture can reduce the costs of animation, which otherwise requires the animator to draw each frame or, with more sophisticated software, key frames which are interpolated by the software. Mocap also saves time and creates more natural movements than manual animation.

There are two types of optical capture systems available commercially today - active optical and passive optical systems. They both use the same underlying principals. Figure 3.1 shows a standard optical mocap marker configuration. A series of cameras placed around the capture space track the positions of markers attached to the body of a performer. Triangulation is used to compute the 3D position of a marker at any given sample from an array of $2D$ information recorded by every camera. Passive optical systems, by which motion data used in this dissertation were acquired, use retro-reflective

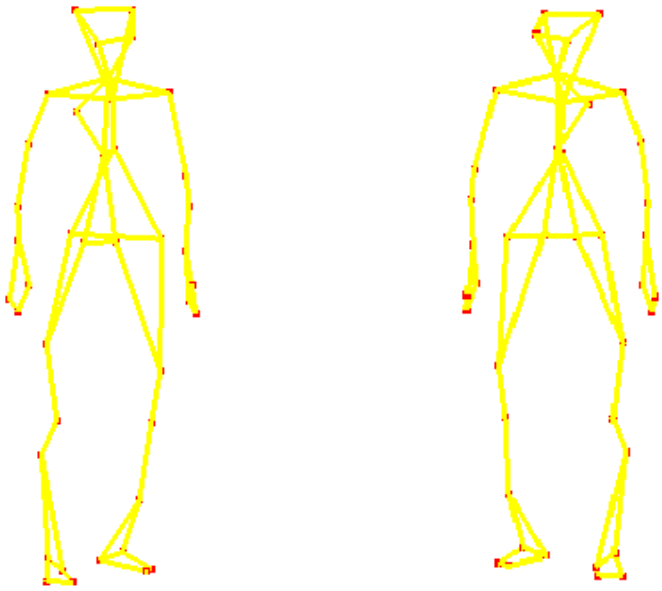


Figure 3.2: Human skeleton model. The red dots indicate the marker positions. The yellow line segments indicate the body segments (i.e. bones).

markers, while active optical systems use illuminating elements as markers. Active markers require wires to the markers, while passive markers do not. Passive optical systems currently are the dominant favorite among entertainment and biomechanics groups due to their ability to capture large numbers of markers at high frame rates and accuracy.

Magnetic systems calculate position and orientation by the relative magnetic flux of three orthogonal coils on the transmitter and each receiver. The relative intensity of the voltage or current of the three coils allows these systems to calculate both range and orientation by meticulously mapping the tracking volume. The markers are not occluded by nonmetallic objects but are susceptible to magnetic and electrical interference from metal objects in the environment or wiring, which affect the magnetic field, and electrical sources such as monitors, lights, cables and computers.

RF (radio frequency) positioning systems are becoming more viable as higher frequency RF devices allow greater precision than older RF technologies. However, to achieve the resolution of optical systems, frequencies of 50 gigahertz or higher are needed.

In recent years there have been studies on markerless motion capture in computer vision community. However, this technique is still in its infancy, and so far the proposed methods are generally slower as well as suffering from resolution issues that make them

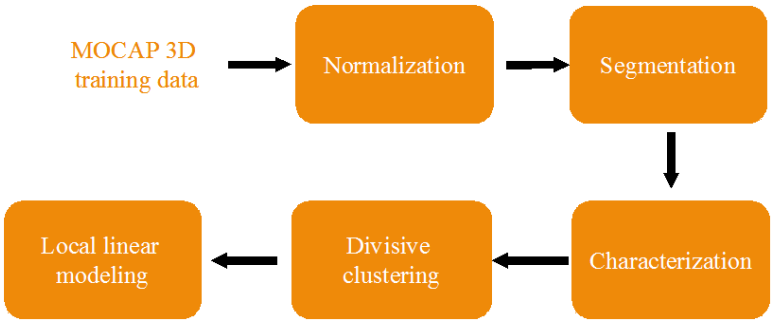


Figure 3.3: A diagram of segment-based, piecewise linear modeling process.

less attractive than marker-based approaches.

Motion capture data are usually represented in two popular formats: $3D$ marker positions and joint angles. A marker position has x, y, and z coordinate values in the world coordinate system. C3D is one of the mostly used data formats to store the $3D$ marker positions. Mocap data used in this dissertation are in the C3D format. Marker positions are often converted into joint angles under a pre-specified skeleton model (Figure 3.2). A joint angle is simply the angle between the two body segments on either side of the joint, measured in degrees. To represent motions in a joint angle format, one has to store a skeleton model which is a rooted hierarchy of connected, fixed-length bones and frame-by-frame changes in joint angles. Commonly used joint-angle based mocap data formats are ASF/AMC, BVA, BVH and HTR.

3.2 Overview

The segment-based, piecewise linear modeling (PLM) approach models human motions as a collection of low-dimensional, local linear models. Figure 3.3 shows a diagram of the PLM modeling framework with key components including segmentation, characterization, clustering, local linear modeling and classification. *Segmentation* divides a motion sequence into segments of distinct behaviors. *Characterization* describes each motion segment with an *equal-length* feature vector summarizing the pose distribution. *Clustering* these feature vectors to group together the motion segments of similar behaviors. *Local linear modeling* constructs a low-dimensional, local linear model for each group of similar motion segments. *Classifier Training* trains classifiers to identify the most appropriate local linear models for the frames in a motion sequence. Not all the components are necessary for every application of the PLM approach. As I will show in the later chapters, some applications may apply only to a subset of this general PLM

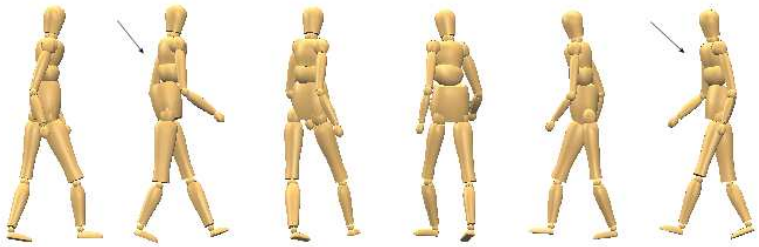


Figure 3.4: Snapshots of a walking sequence.

framework, while adding some other modules if necessary.

Throughout the paper I treat each pose of motion data as a data point represented by a $3m$-dimensional column vector, $y \in R^{3m}$, containing $3D$ marker positions of m markers. Thus a motion data set with N pose instances can be represented by a $3m \times N$ data matrix $Y = [y_1, y_2, \cdots, y_N]$, where y_i is a column vector of marker positions ($i = 1, \cdots, N$). For convenience, each entry of the $3m$-dimensional marker position vector is referred to as a feature and each pose is then considered as a high-dimensional data point with $3m$ features.

3.3 Normalization

Normalization serves as a preprocessing step in the modeling pipeline. Typical motion data are captured in an absolute world coordinate frame. Data of poses that are essentially similar to each other may appear to be quite different due to offsets in translations and orientations. For example, in Figure 3.4 two poses pointed to by the arrows appear to be very similar to each other except that the second pose is just a few steps away from the first pose and faces to a different direction. Furthermore, motion sequences captured in different absolute world coordinates may have quite different data matrices, although they appear to be very similar to each other. In order to remove the influence of translation and orientation among poses and be able to reveal their inherent similarity/dissimilarity, I describe relative motions in a model-rooted frame, where all the translational and orientational effects are removed from the pose vectors through appropriate transformations. Such a transformation procedure is called *normalization*.

The normalization process is quite straightforward. Figure 3.5 illustrates this normalization process. I choose the marker located at the STRN as the origin, the same z-axis as in the original world coordinate system. I compute a vector from the left shoulder marker to the right shoulder marker, and I then project it to the horizontal

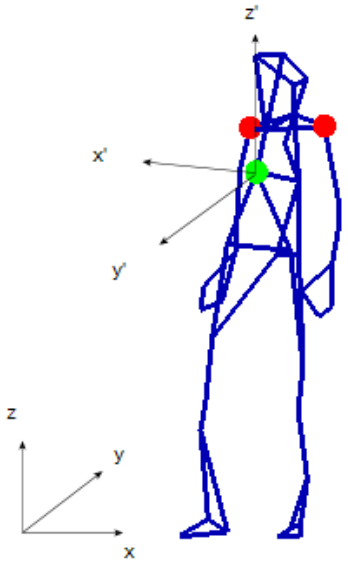


Figure 3.5: An illustration of a normalization process. The green disk indicates the marker position at the STRN, which is the origin in the normalized coordinate system. The two red disks indicate the marker positions at left and right shoulders. The x, y, and z axes of the original world coordinate system is drawn at the left bottom; the x', y', z' axes of the normalized coordinate system are drawn around the green disk of the STRN marker.

plane and use the projected vector as the new x-axis. The cross product between newly defined z and x axes produces the y-axis. For a given pose, I first center the pose position vector by its STRN marker position and then apply a series of rotations to convert the x, y and z coordinates of each marker position to the corresponding coordinates in the normalized coordinate system. For some applications of the PLM approach, a denormalization procedure is necessary to transform the results back to the original feature space.

3.4 Motion Segmentation

There is ample evidence that pose sequences from simple single-behavior motions lie on or near a very low-dimensional linear subspace (Safonova et al., 2004). For example, in Figure 3.6 I show PCA of several simple walking motion clips. Although the original feature space has over 100 dimensions, PCA computation reveals that the inherent dimensions are much lower. While a short and simple motion sequence is more likely to contain a single behavior, a long and complex motion may consist of a series of different

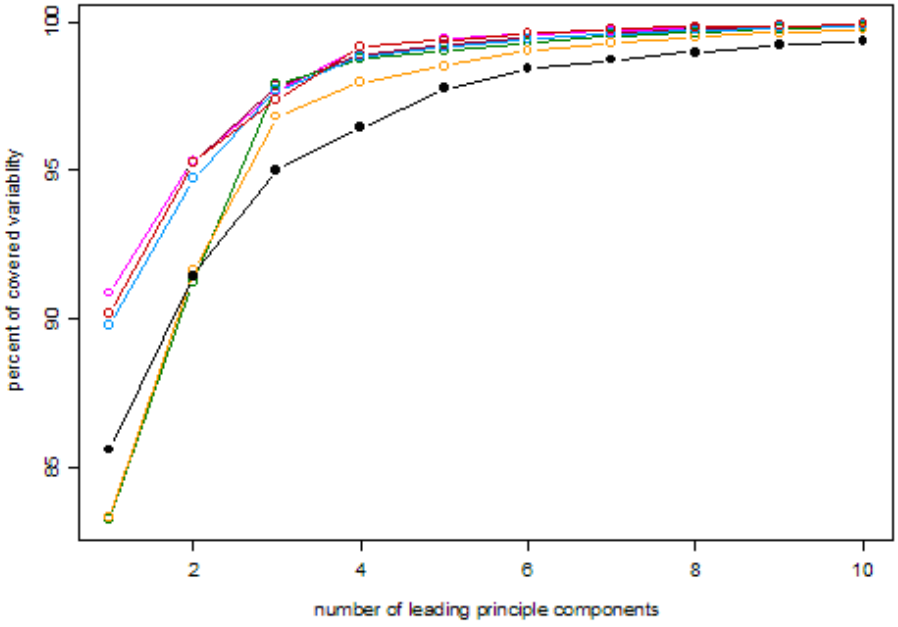


Figure 3.6: Percentages of variances explained by the principal components for different walking sequences. The curve in black with dots is for all the walking sequences combined; the rest of the curves with circles are for individual walking sequences of different styles.

behaviors, with transitional frames from one behavior to another. In Figure 3.7 I show a long motion sequence, including several simple motions of distinct behaviors. In order to further model motion data and effectively organize a motion database, it is important to correctly identify different behaviors in the motion sequences and divide them into motion segments accordingly.

There have been studies on motion segmentation (Pavlovic et al., 2000; Li et al., 2002; Barbič et al., 2004). I choose the probabilistic PCA (PPCA) approach (Barbič et al., 2004) to segment a motion sequence into subsequences of distinct behaviors. As an extension of traditional PCA, PPCA is based on a probabilistic model (Tipping and Bishop, 1999) and models the residual variance discarded by PCA. The pose distribution of a motion sequence is an important statistical property that can be exploited for segmenting motion data into distinct behaviors. Motion data are considered as an ordered sequence of poses and are modeled with multivariate Gaussian distributions in PPCA. A motion sequence is segmented where there is a local change in the distribution

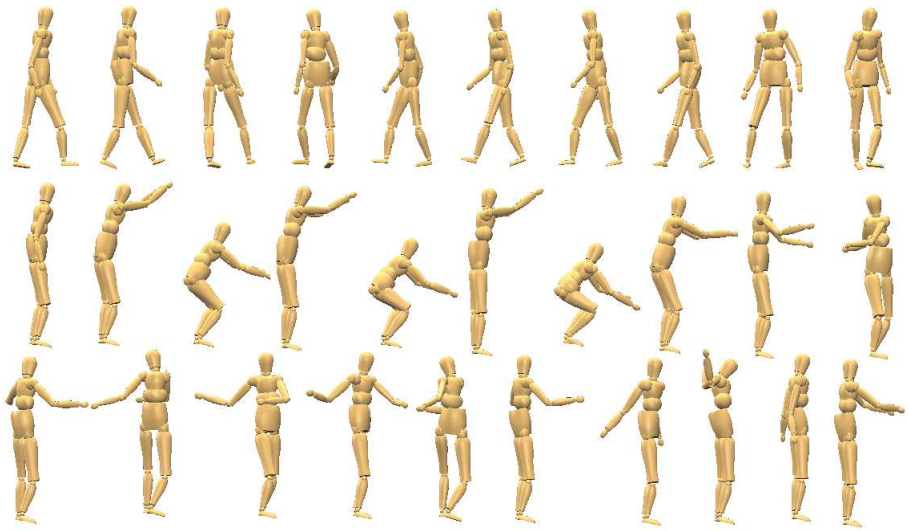


Figure 3.7: Snapshots of the sample frames from a long motion sequence. The sequence can be seen as a concatenation of three single-behavior motions: walking (frames 1:10 in the top row), squatting (rames 11-20 in the middle row) and stretching (frames 21:30 in the bottom row).

of the poses.

To derive a multivariate Gaussian model from a motion sequence of n poses, $X = [x_1, x_2, ..., x_n]$, where x_i is a $3m$-dimensional column vector of marker positions ($i = 1, ..., n$), and I first retrieve the mean pose of the motion sequence as

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i, \tag{3.1}$$

I subtract the mean pose from each pose vector and form a data matrix D with each row corresponding to a $3m$-dimensional mean-centered pose vector. I then apply Singular Value Decomposition to D such that

$$D = U \Sigma V^T, \tag{3.2}$$

where columns of U and V are orthogonal unit vectors, and Σ is a $3m \times 3m$ diagonal matrix with nonnegative decreasing singular values σ_j on its diagonal. I compute a ratio

$$E_r = \frac{\sum_{j=1}^{r} {\sigma_j}^2}{\sum_{j=1}^{3m} {\sigma_j}^2}, \tag{3.3}$$

where E_r indicates the portion of the total variance covered by the leading r principal

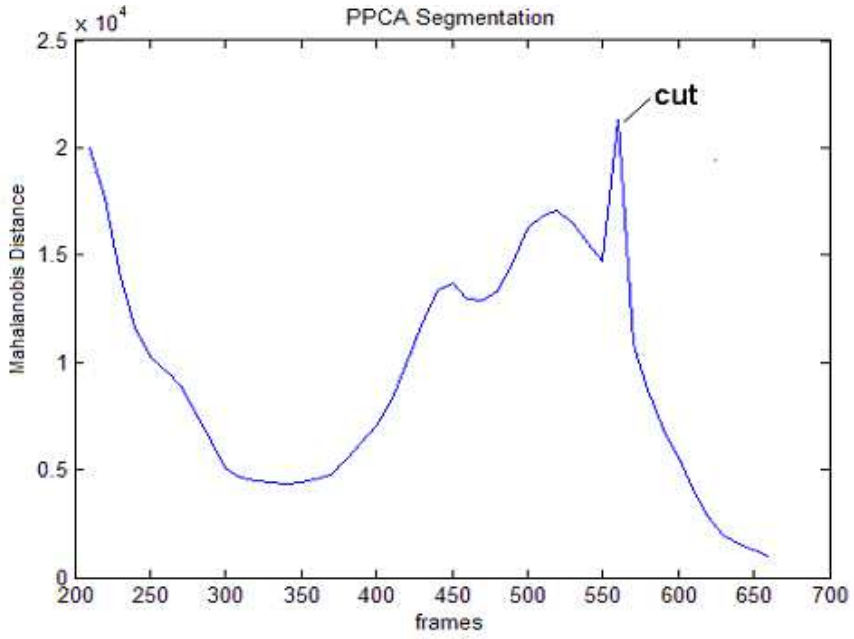


Figure 3.8: Plot of *Mahalanobis* distance H as K is repeatedly increased by Δ in PPCA segmentation.

components. I keep the first r principal component such that E_r is larger than a preset tolerance τ. I then define an average square of discarded singular values σ_j $(j = r+1, ..., 3m)$ as

$$\sigma^2 = \frac{1}{3m-r} \sum_{i=r+1}^{3m} {\sigma_i}^2, \tag{3.4}$$

and subsequently

$$W = V_r(\Sigma_r^2 - \sigma^2 I)^{1/2}, \tag{3.5}$$

where Σ_r^2 denotes the upper right $r \times r$ block of Σ, and V_r is the first r columns of V. Now I can compute the covariance matrix C as

$$C = \frac{1}{n-1}(WW^T + \sigma^2 I) = \frac{1}{n-1} V \tilde{\Sigma}^2 V^T \tag{3.6}$$

where $\tilde{\Sigma}$ is acquired by replacing all discarded singular values of Σ with σ^2.

In practice I start by modeling the first K frames of a motion sequence as a multivariate Gaussian distribution using the method described above. I then estimate how likely motion frames $K+1$ through $K+T$ belong to the Gaussian distribution of the first K frames, defined by their mean $\bar{x}$ and covariance matrix C, computed by PPCA.

I do this by calculating an average *Mahalanobis* distance H (Duda et al., 2000) as

$$H = \frac{1}{T} \sum_{K+1}^{K+T} (x_i - \bar{x})^T C^{-1} (x_i - \bar{x}) \tag{3.7}$$

Next I increase K by a small number of frames Δ and repeat the estimation of distribution for the first K frames ($K := K + \Delta$), and I compute the distance H with respect to the new distribution. Figure 3.8 shows a plot of H as K repeatedly increases by Δ. In the plot a cut is declared at the peak following a valley and when the height difference between the valley and the peak is greater than a threshold R. In general, increasing R results in fewer segments that correspond to more distinct behaviors, and decreasing R results in a finer segmentation. When a cut is made, the first K frames constitute one segment and are removed from the sequence. I continue to segment the rest of the sequence until the length of the remaining subsequence is less then the sum of the initial value of K and T. In my experiments I set the initial values $K = 200, \tau = 0.95, T = 150, \Delta = 10, R = 500$. The segmentation results are not very sensitive to the minor adjustments of K, T or Δ, although bigger adjustments would certainly affect the sizes of the segments.

3.5 Characterization of Motion Segments

Segmentation divides complex motion sequences into simple and distinct behaviors but provides no information on which segments are more similar to each other. In addition, motion segments may have a different number of frames; this fact prevents us from directly comparing or clustering these motion segments for the purpose of grouping, since most clustering techniques require the participated objects to have the same dimensionality. I am aiming to derive from each motion segment a set of features that can sufficiently characterize a motion segment while having the same number of features across the motion segments with different number of frames.

It has been observed that motion data representing similar behaviors nearly lie in the same low-dimensional space, with similar shapes, orientations and pose distributions (see an example shown in figure 3.9). This property is insensitive to spatial variation as well as temporal distortion. Based on this observation and also considering the fact that pose distribution is a good summary of a motion segment, I decide to derive features that characterize the pose distribution of each motion segment. First, I assume a

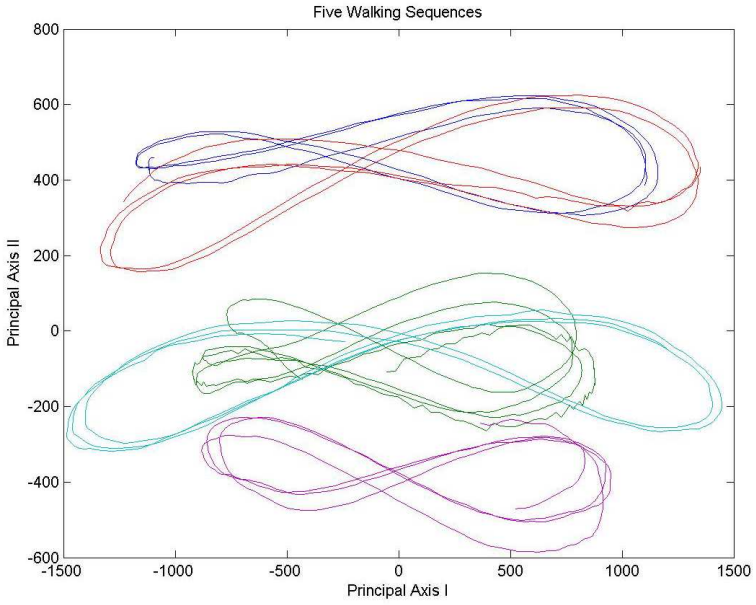


Figure 3.9: Projections of five walking sequences onto their two common leading principal axes. The curves of these walking sequences clearly have similar shapes and orientations. They differ mostly by mean positions and scales.

multivariate Gaussian distribution on the poses in each motion segment. I then choose to derive features from the covariance matrix and the mean vector of each motion segment, since they can uniquely determine a Gaussian distribution. Even for segments of different number of poses, the derived means and covariance matrices would be of the same size as along as each pose has the same number of features. This is particularly appealing to us since it satisfies my goal to derive an equal number of features from every motion segment of arbitrary number of frames. Let $\mu = (\mu_1, \mu_2, ..., \mu_{3m})^T$ be the mean pose and Σ be the covariance matrix of a motion segment defined as

$$\Sigma = \begin{pmatrix} \sigma_{1,1} & \sigma_{1,2} & \cdots & \sigma_{1,3m} \\ \sigma_{2,1} & \sigma_{2,2} & \cdots & \sigma_{2,3m} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{3m,1} & \sigma_{3m,2} & \cdots & \sigma_{3m,3m} \end{pmatrix}$$

I form a feature vector

$$f = [w\nu^T, (1-w)\mu^T]^T, \tag{3.8}$$

where ν is a column vector consisting of the variances and covariances. These values are

retrieved by concatenating the elements in the upper triangle and the main diagonal of the convariance matrix Σ, and w is a weighting factor to balance the importance between the covariance matrix Σ and the mean vector μ. If I consider poses as data points in a $3m$-dimensional space, Σ is a $3m \times 3m$ matrix, and μ is a $3m \times 1$ vector. I then have a $3m(3m+3)/2$-dimensional feature vector associated with the segment.

However, this feature vector space has very high dimensionality with very sparse data points, each representing a motion segment. So I use PCA to reduce the dimensionality of the feature vectors. My empirical results show that typically fewer than 40 principal components are needed to cover the 95% of the variance.

3.6 Clustering Motion Segments

I present here a divisive clustering method to identify and group motion segments that can be represented in the same low-dimensional space, or in other words, by the same local linear model. The goal of clustering is to group a collection of data points into subsets, i.e. "clusters", such that those within each cluster are more closely related to one another than those assigned to different clusters. Before any clustering I have to give a definition on similarity. I define the similarity as the Euclidean distance between the feature points. Given a collection of n feature points of d dimensions, the similarity is defined as follows:

$$similarity = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2}, \tag{3.9}$$

where p and q are two arbitrary d-dimensional feature points. That is to say, given two feature points, the closer Euclidean distance is between the two feature points, the more similar they are and vice versa.

I construct a hierarchy of local linear models by divisive clustering on feature vectors of motion segments, with the Euclidean norm as a distance metric. I start by putting all the feature points into a single cluster. Before I start the procedure, I need to decide on a threshold distance. Once this is done, the procedure proceeds as follows:

1. The mean of the cluster is computed.

2. The pairwise distance between each feature point in the cluster and the cluster mean is computed.

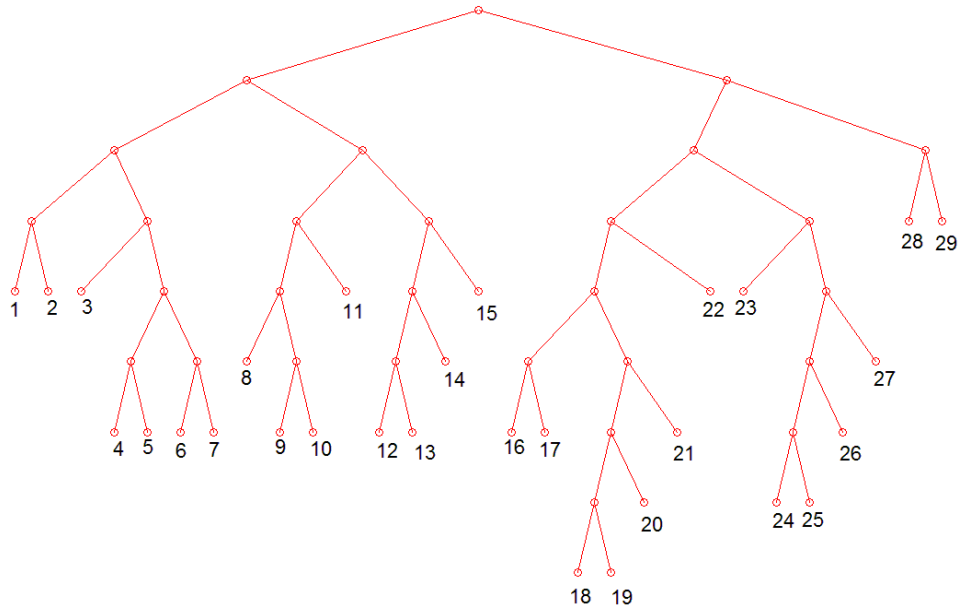


Figure 3.10: An illustration of cluster tree. Each leaf node is associated with a local linear model by a model ID.

- If the pairwise distance between any feature point and cluster mean is greater than the threshold, this cluster is divided into two sub-clusters. The splitting is done by a K-Means splitting algorithm with K set to 2. Then for each of the sub-clusters, the procedure goes back to step 1.
- If the pairwise distances between all the feature points and cluster mean are equal to or less than the threshold, then the clustering procedure stops at this branch, and this cluster is considered as a leaf in the modeling hierarchy.

The whole divisive clustering process continues until the feature points in all the clusters satisfy the distance tolerance. Figure 3.10 illustrates an example cluster tree.

3.7 Local Linear Modeling

For the motion segments grouped into the same cluster, their poses often lie near a linear space of much lower dimensionality. Poses can be interpolated on this low-dimensional space. For example, Figure 3.11 shows one of these clusters and interpolates novel poses along each of the four principal axes. I aim to find a local linear model that can sufficiently describe the lower-dimensional linear space as well as a series of

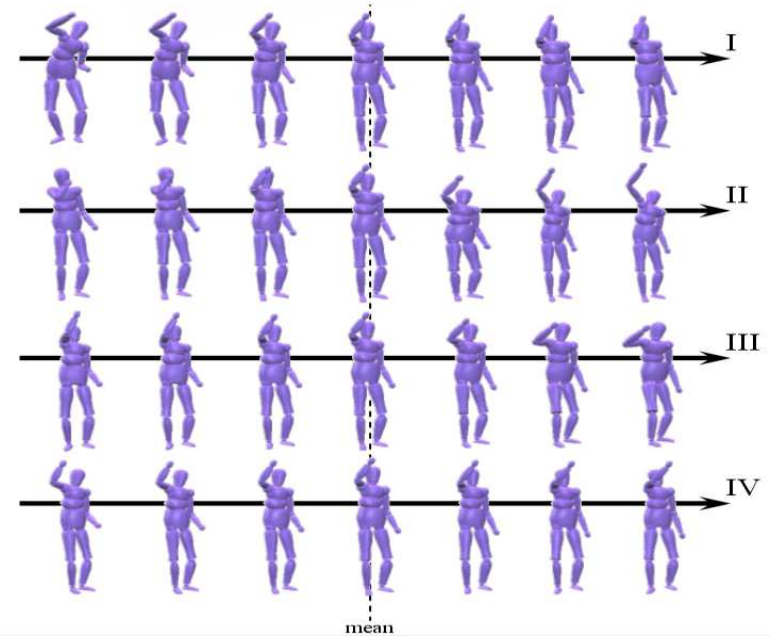


Figure 3.11: Motion interpolation along the four principal axes of local linear model. I, II, III and IV represent the leading four principal axes. Four principal axes are drawn horizontally with the cluster mean in the middle. Poses are interpolated in both directions along the four axes from the mean.

linear operations that can transform poses onto this newly formed coordinate system. Since principal component analysis has been one of the commonly used techniques and popular choices to find the embedded linear space out of high-dimensional data sets, I apply PCA on the mocap data to find the local linear models. I first compute the mean pose out of those pose vectors and subtract the mean from the poses. I then apply PCA on these mean-centered pose vectors. I keep the first k eigenvectors, such that the residual variance covered by the discarded eigenvectors is less than a preset threshold. For each group of similar motions, I save the mean vector and the leading k principal components as a compact model of the poses in the group. As I will demonstrate in the later chapters, these low-dimensional linear models are good approximations of the original human poses. By keeping an adequate number of principal components that sufficiently cover the variance of the associated poses, I can preserve the important subtleties exhibited by most human motions.

Least-squares methods are often used in a local linear modeling process. As I will later show in the driving problems, I apply the linear least-squares methods to approximate the unknown motions from incomplete information. For example, in missing marker estimation, a least-squares method is used to estimate a pose's projection onto

the principal component space from the available marker measurements. In order to facilitate the understanding on my local linear modeling scheme, I will briefly discuss the principal of standard linear least square method.

Linear least-squares method is a mathematical optimization technique to find an approximate solution to a system of linear equations that has no exact solution. This usually happens if the number of equations (m) is bigger than the number of the variables (n).

In mathematical terms, we want to find a solution to the equation

$$Ax = b, \tag{3.10}$$

where A is a m-by-n (with $m > n$) matrix; while x and b are respectively n- and m-dimensional column vectors. This is equivalent to minimizing the Euclidean norm squared of the residual $Ax - b$, that is, the quantity

$$\|Ax - b\|^2 = \sum_{i}^{m} ([Ax]_i - b_i)^2, \tag{3.11}$$

where $[Ax]_i$ is the i^{th} component of the vector Ax. Using the fact that the squared norm of v is $v^T v$, where v^T is the transpose of v, we may rewrite the expression as

$$(Ax - b)^T (Ax - b) = (Ax)^T (Ax) + b^T b. \tag{3.12}$$

The minimum is found at the zero of the derivative with respect to x, i.e.,

$$2A^T Ax - 2A^T b = 0, \tag{3.13}$$

which leads to the normal equation

$$A^T Ax = A^T b. \tag{3.14}$$

Thus, the unique solution is given by

$$x = (A^T A)^{-1} A^T b. \tag{3.15}$$

3.8 Training Classifiers

My segment-based, piecewise linear modeling approach partitions motion segments into a collection of clusters that are then modeled by low-dimensional linear models. A key component in this framework is to correctly identify a local linear model for a given pose in a motion sequence. A classifier is trained for this purpose.

Among various types of classifiers, The Random Forest (RF) classifier has shown excellent performance in handling human motion data. A Random Forest classifier is a classifier that consists of many decision trees, i.e. CART trees. Each tree is constructed with the following algorithm:

1. Let the number of training cases be N, and the number of variables in the classifier be M.

2. Let m of input variables be used to determine the decision at a node of the tree; m should be much smaller than M.

3. Choose a training set for this tree by choosing N times with replacement from all N available training cases. Use the rest of the cases to estimate the error of the tree, by predicting their classes.

4. For each node of the tree randomly choose m variables on which to base the decision at that node. Calculate the best split based on these m variables in the training set.

This process is repeated until a user-defined number of trees have been created. The collection of the trees is a Random Forest. In Random Forest the individual CART trees are not influenced by each other when being constructed. Once a Random Forest is constructed, the prediction for each tree is used in a *voting* process. The overall prediction is determined by voting over all the trees in the forest and choosing the class with the most votes. Random Forest has some advantages over other classification techniques that are quite appealing to us. For example, Random Forest does not need to do any variable selection or data reduction and will automatically identify the best predictors. It also has an ability to handle data without preprocessing. Data do not need to be rescaled, transformed or modified. It is also resistant to outliers. Random Forest is also resistant to over-training. Since it generates numerous trees based on two forms of randomization, and each tree is an independent, random experiment, growing a large number trees in Random Forest does not create a risk of overfitting. Finally,

Random Forest has a mechanism of self-testing using *out-of-bag* data and is based on an extension of cross validation. These good properties make Random Forest a very attractive candidate for classifying human motion data in this dissertation research.

In my classifier training process, the inputs are the marker positions and model IDs of the poses in the training set. The output is the resulting RF classifier. As my experiments later show, Random Forest performs well with a high degree of accuracy and is robust to the size and heterogeneity of motion data. This in turn indicates that my piecewise linear modeling approach to label identification is sufficient and effective to characterize human motion data.

Chapter 4

Human Motion Estimation from a Reduced Marker Set

Motion capture is a prevalent technique for capturing and analyzing human articulations. However, most motion capture systems are cumbersome, expensive, intrusive and time consuming. These drawbacks may not only prevent mocap data from being easy to use, but they also might make it impractical for potential applications.

I use a small set of markers to quickly generate plausible human motions on a frame-by-frame basis. My model is very compact and completely eliminates the need for a motion database after offline training. It is also very fast in estimating motions from a reduced marker set. As the experiments show in later sections, I can reconstruct human motion frame by frame at a rate of over 600 frames per second. Thus, my method shows great promise for use in most interactive motion applications.

I eventually hope to employ this method for generating self-avatars for virtual environments (VEs), where the combined encumbrances of both a head-mounted display and a full mocap setup are impractical. However, it is conceivable that a participant might undergo a short mocap session prior to entering a virtual environment. This training data would then be used to estimate a plausible avatar from a significantly reduced marker set.

This chapter is organized as follows: in section 4.1 I give an overview on the algorithms and briefly discuss the key components, including principal marker selection, piecewise linear modeling, classifier training as well as the motion reconstruction. Then from section 4.2 to 4.5, I will describe these key components in detail, respectively. I will present the experimental results with discussions in section 4.6. Finally I will summarize this chapter in section 4.7.

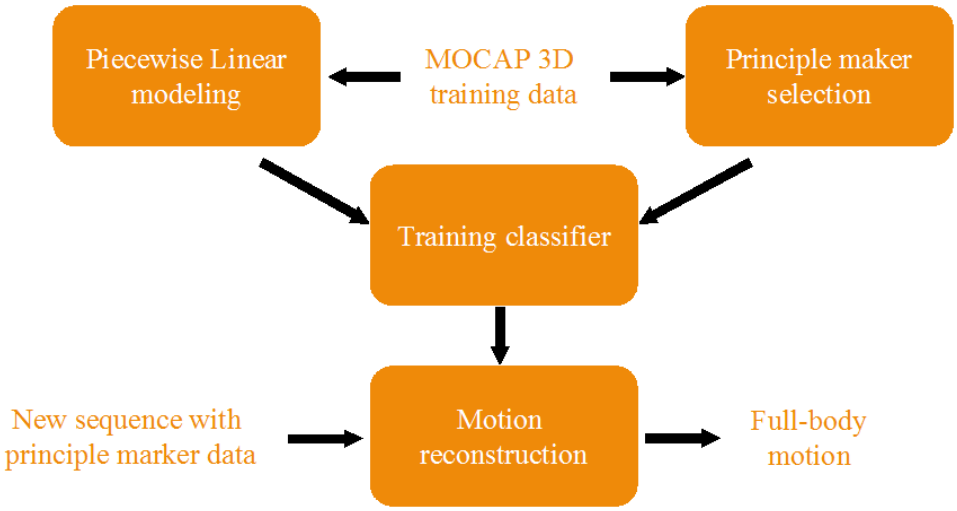


Figure 4.1: Key component diagram of the motion estimation process from a reduced marker set.

4.1 Overview

My goal is to estimate human motions from a small set of the most informative markers. Figure 4.1 is a diagram of the key components in a process of motion estimation from a reduced marker set. I will give a brief overview here, with more details explained later.

Principle marker selection. Principal component analysis (PCA) is one of the most popular methods for dimensionality reduction of a feature set. However, the principle components are latent variables and are hard for interpretation. I, on the other hand, want to choose a small subset of the original markers which I call *principle markers* and which contain most of the essential information of the whole marker set. Figure 4.2 shows a couple of example illustrations of human poses with the principal markers highlighted. I adapt a PCA-based principle feature analysis method (Cohen et al., 2002) to selecting a set of principle markers.

Piecewise linear modeling. I first apply the probabilistic PCA (PPCA) (Tipping and Bishop, 1999; Barbič et al., 2004) to divide a motion sequence into simple motion subsequences of distinct behaviors and local linearities. These subsequences are referred to as *motion segments*. I then characterize each motion segment by an *equal-length* feature vector and use these feature vectors as modeling primitives to construct a hierarchy of local linear models *via* a divisive clustering method. Similar motion segments are partitioned into the same leaf cluster. Poses in a leaf cluster are associated with a unique local linear model and are used to compute a linear mapping function

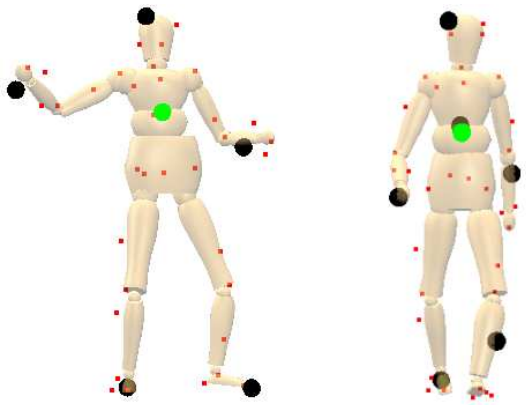


Figure 4.2: Shown above are the principal markers selected from 2 motion data sets. The green disk indicates the marker used as the origin in the normalized coordinate systems. The principal markers are shown in black and the estimated markers are shown in red.

from a set of principal markers to the rest markers.

Training classifier. In order to be able to use the local linear models to estimate full-body human poses from a principle marker set, I need to identify the most appropriate model given a pose with only the positions of principle markers available. A Random Forest classifier (Breiman, 2001) is trained for this task from the training set data, of which each pose is labeled with a local linear model ID.

Motion reconstruction. Given a new motion sequence with only position measurements from the principal markers, I classify each frame to the most appropriate local linear model using the Random Forest classifier. I then use the associated linear mapping functions to reconstruct the full marker positions of the poses from the principal marker set. I smooth out possible discontinuities using a mixture of local linear models for the poses at the transitions between models.

4.2 Principal Marker Selection

Human motion data has significant redundancy which can be revealed with PCA. Figure 4.3 shows the accumulative variance explained by the principal components for a data set comprised of a variety of human behaviors, including walking, running, bending and washing. The first 10 principal components cover 99% of the variability, implying that a data set like this, with 40 markers (120 features), has only slightly more than 10 degrees of freedom. In selecting the principle markers, an approach like principal

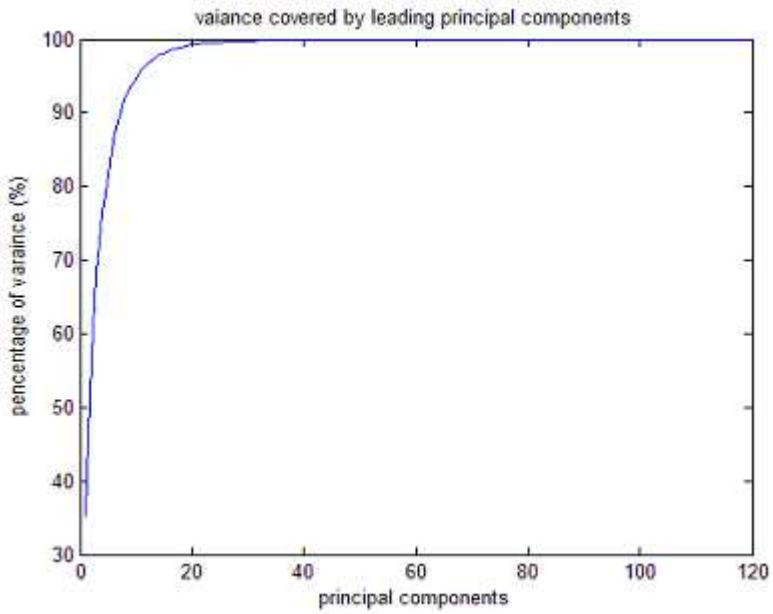


Figure 4.3: Percentage of variance explained by the principal components of a motion data set composed of 12,670 frames with 40 markers (120 features). Fewer than 20 principal components are needed to reconstruct the original feature set to with 99% accuracy.

feature analysis, i.e. PFA (Cohen et al., 2002), is very appealing. PFA has comparable performance to PCA but selects a set of original features which are measurable and have a more intuitive interpretation than the principle components derived by PCA. On the other hand, PFA treats each feature independently, even though the three features of each marker are always measured together. So I design an algorithm based on PFA, which selects a minimal set of principal markers instead of individual principal features in PFA.

The basic idea of PFA is to exploit the structure of the principal components from PCA and choose the principal features, which retain the essential information in the sense of both maximizing variability of the features in the lower-dimensional space and minimizing the reconstruction errors. I first use PFA to partition all the features into clusters. Then I impose some criteria to weight the importance of each marker and select a minimal set of the most important markers satisfying a cover of all clusters of features. The steps of principal marker selection are summarized as follows:

1. Run PCA on the covariance matrix of data matrix Y.

2. Construct a $3m \times q$ matrix A_q by selecting the q dominant eigenvectors that are

sufficient to satisfy a desired reconstruction error tolerance. The rows of A_q form the feature weight vectors, i.e. V_1, ..., V_{3m}, $V_i \in R^q$ $(i = 1 : 3m)$, which are the projections of the feature variables on the q leading principle components.

3. Take element-wise absolute value of V_i to obtain absolute feature weight vectors $|V_i| \in R^q$ and use K-means clustering algorithm to partition the $3m$ absolute feature weight vectors to K clusters with K slightly greater than q.

4. Weight markers according to their importance. Remove the least important markers, as long as every cluster is still covered by at least one marker after the removing.

A key rationale behind this feature clustering method is the realization that the rows of the matrix A_q can be used to effectively characterize the relationships between the features. In other words, if two features are highly correlated, they will have similar absolute value weight vectors.

I use the number of unique clusters containing a marker feature to define the importance of a marker. That is to say, a marker that appears in more distinct clusters is considered to be more important. To break ties between markers, I prefer those whose sum of the square distances from the marker's features to their cluster mean is minimal. Markers are sorted in the order of least importance. I continue removing the least important markers as long as every cluster is still covered by at least one feature. This process is repeated until no more markers can be eliminated.

K-means clustering is an iterative algorithm whose result depends on the choice of initial cluster means. However, it is my experience that the resulting clusters and the set of principal markers are surprisingly consistent and insensitive to the initial settings. In Figure 4.4 I illustrate a frequency histogram of the selected principal markers from 1000 runs of my principal marker selection algorithm on a motion data composed of 40 markers and 12,670 frames. All seven principle markers are consistently selected in more than 94% of all runs.

4.3 Piecewise Linear Modeling

I apply the piecewise linear modeling approach to partition motion data into a collection of local linear models using motion segments as the modeling primitives. The modeling process, as described in Chapter 3, consists of key components such as processes of

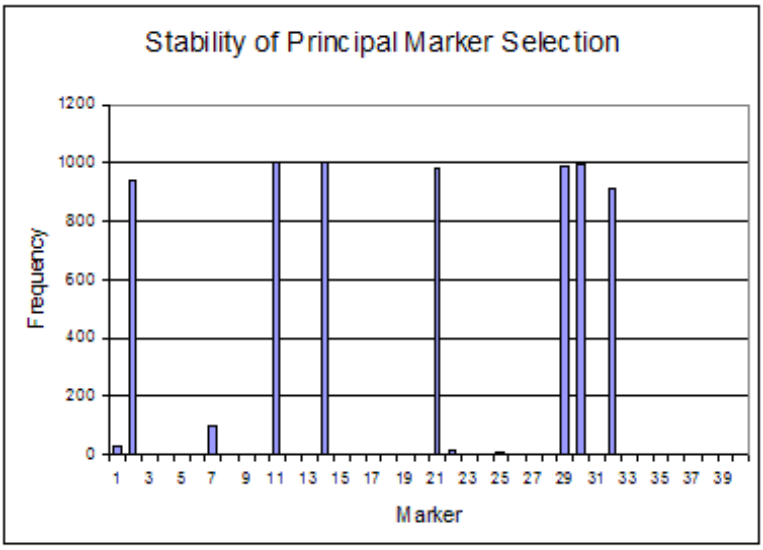


Figure 4.4: Frequency histogram of selected principle markers from 1000 runs.

motion segmentation and characterization, computing local linear models as well as classifier training. Segmentation and characterization are implemented exactly the same way as described in Chapter 3, while local linear modeling and classifier training are modified to accommodate the situation in this driving problem. In this section I only describe how I compute the local linear model and train the classifiers.

For each local linear model, I compute a least-squares mapping function to estimate the $3D$ positions of the non-principal markers from a principal marker set. Assuming k out of m markers constitute a principal marker set, I represent a pose as a $3m \times 1$ vector $y = [x^T, z^T]^T$, where $x \in R^{3k}$ represents the 3D positions of principle markers, and $z \in R^{3(m-k)}$ represents the $3D$ positions of the rest markers. Then the least squares mapping matrix B can be computed for a cluster of n poses as

$$B = ZX^T(XX^T)^{-1}, \tag{4.1}$$

where $X = [x_1, x_2, ..., x_n]$ is a $3k \times n$ matrix, and $Z = [z_1, z_2, ..., z_n]$ is a $3(m-k) \times n$ matrix.

In order to use the local linear models and the associated mapping functions to estimate full-body poses from a principle marker set, a Random Forest (Breiman, 2001) classifier is trained to identify the most appropriate model with only the position values from the principle markers. In my classifier training process, I label each frame from the training set with its model ID, and I use its principal marker positions as input for

training the Random Forest classifier. Random Forest (RF) is a powerful classification tool that has displayed outstanding performance in regard to classification errors. RF grows and combines decision trees into predictive models. The overall prediction is determined by voting over all the trees in the forest and choosing the class with the most votes. Since the trees are generated randomly and independently, there is no risk of overfitting for large numbers of trees. As my experiment shows, RF performs well with a high degree of accuracy, and it is robust to the size and heterogeneity of motion data. This, in turn, indicates that my piecewise linear modeling approach to label identification and selection method of principle markers is sufficient and effective in characterizing motion data.

4.4 Motion Reconstruction

4.4.1 Estimations of Poses

Once I've learned piecewise linear models and trained a Random Forest classifier with a training set, I are ready for estimating human motions from a principle marker set. For each frame, given measurements on its principle markers denoted by vector x, I use an Random Forest classifier to identify the most appropriate local linear model and the associated least-squares mapping function from the principal markers to the rest of markers. I then estimate the $3D$ positions of the remaining markers, z, as

$$z = Bx \tag{4.2}$$

where B is a linear mapping matrix.

4.4.2 Estimating Poses in Transition with Mixture of Local Linear Models

An inherent shortcoming with piecewise linear modeling approach is the temporal discontinuity at the transitions between models, manifested as visible jerkiness in the reconstructed motion. A change of bias in the reconstruction errors is one of the leading causes to temporal discontinuity. For example, if the reconstruction errors of consecutive frames are all biased towards the same direction, the motion may still appear smooth, although its root-mean-square (RMS) error may be a bit higher. On the other hand, if the biases are toward different directions, it may cause more severe jerk-

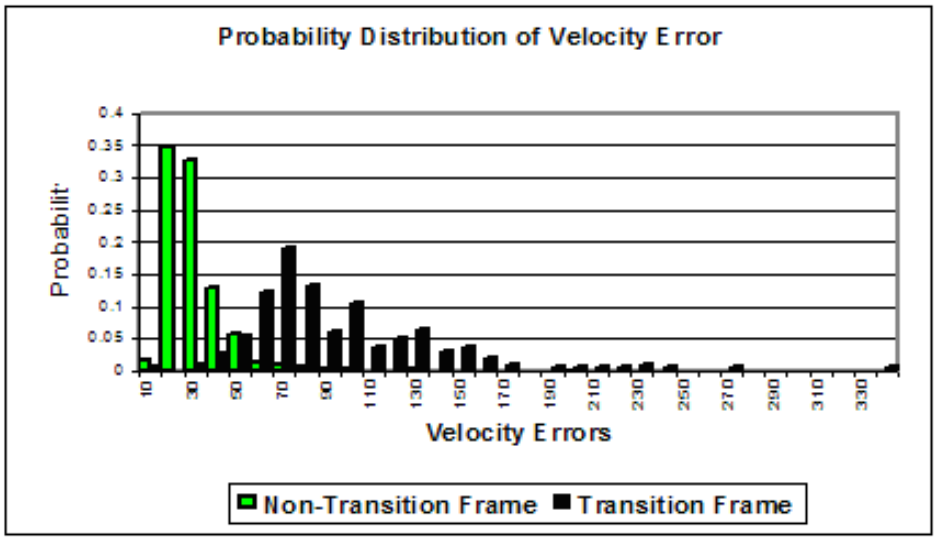


Figure 4.5: The probability distribution histogram of the velocity errors for reconstructed motions from a motion data set.

iness even if the root-mean-squared error is moderate. The bias direction tends to vary more dramatically during transitions between local linear models. Therefore, temporal discontinuities are frequently visible at the transitions between local linear models.

I provide a simple metric to evaluate the jerkiness for each marker reconstruction. For a given marker, let p_t and $p_t^{'}$ be the true and predicted positions at time t;, and p_{t-1} and $p_{t-1}^{'}$ be the positions at time $t-1$. Then I compute the true and predicted velocities v and $v^{'}$ as follows:

$$v = p_t - p_{t-1} \tag{4.3}$$

$$v^{'} = p_t^{'} - p_{t-1}^{'} \tag{4.4}$$

I then take the Euclidean norm of their vector difference (e.g. errors) as my jerkiness metric, γ i.e.,

$$\gamma = \| \, v - v^{'} \, \| \tag{4.5}$$

When the errors are biased in similar directions, v and $v^{'}$ tend to be closer to each other, leading to smaller values of γ. On the other hand, differences in the bias directions of v and $v^{'}$ lead to larger values of γ.

I verify the validity of this jerkiness metric experimentally on motion reconstruction using my method from a motion data set. In Figure 4.5 I plot a histogram of γ for all markers, where transition and non-transition frames are shown in different colors. The

histogram shows that non-transition frames tend to be non-jerky, or in other words, smooth. On the other hand nearly all jerky frames occur at transitions between local linear models.

A typical solution is to incorporate a factor that evaluates the continuity of the pose relative to the previous poses into the optimization phase (Chai and Hodgins 2005; Grochow et al. 2004). In contrast, I perform a fuzzy regression for the poses at the transitions of local linear models to smooth out the temporal discontinuity. Instead of using only the local linear model where the pose is classified to reconstruct a full-body pose, I use a mixture of models associated with the current pose and some poses prior to it. This approach shares the spirit of fuzzy/soft classification and addresses the fact that transitional poses tend to be competed by different local linear models. Let x_t be a pose vector containing the $3D$ positions of principle markers at time t, I estimate the positions of the rest of the markers, z_t, as,

$$z_t = \sum_i w_i B_i x_t, \tag{4.6}$$

where $w_i = r_i/(h+1)$ is a weight for the i^{th} model, r_i is the number of poses classified to the i^{th} model among the prior h poses and current pose, and B_i is the mapping matrix for the i^{th} model. Basically, I want to put more weights on the model that is favored by more of the h poses prior to the current pose. In my experiments, $h = 10 - 30$ works very well.

4.5 Experiments

4.5.1 Design

I evaluated my modeling approach using Carnegie Mellon University's Graphics Lab Motion capture database available at http://mocap.cs.cmu.edu. To obtain a reasonable representation of motion data space, I prepared a large and heterogeneous human motion database including various motions from multiple subjects. I divided the motion sequences into a training set and a testing set, with the training set having similar sequences to the sequences in the testing set. I used the training set to learn local linear models and to extract a set of principle markers. Full-body poses were then reconstructed for the testing sequences based on the marker positions of the principle marker set, and they were compared to the actual full marker measurements. I

compared the performance of my method to the nearest neighbor search method.

4.5.2 Results

My training set consisted of 132 sequences with a total of 151,882 frames collected from 21 subjects. The training sequences contained a variety of motions, such as walking, running/jogging, golfing, soccer kicking, Salsa dancing, jumping, cartwheeling, climbing steps, etc. Even for the same category of motions, sequences of different styles from different subjects were included. The testing set contained 28 sequences with 19,553 frames from 18 subjects. Among them, there were 9 walking sequences, 6 running, 5 golfing, 2 cartwheeling, 2 Salsa dancing, 1 walking on uneven terrain, 1 running jumping, 1 soccer kicking, and 1 climbing three steps. Four testing sequences, namely, 1 walking, 1 soccer kicking, 1 running and 1 golfing were from 4 new subjects who never performed any motion that was used in the training set.

In selecting a set of principle markers from the training set, I computed PCA to cover 95% of the total variance of all the poses in the training set. Then I applied the principal marker selection method to automatically select a set of six principal markers, placed at left forehead (LFHD), right elbow (RELB), left arm (LARM), right leg (RLEG), left toe (LTOE) and right toe (RTOE). The training set sequences were segmented into 271 motion segments with lengths varying from 128 to 3,670 frames (mean: 560; standard deviation: 425; median: 440). The dimensionalities of motion segments were as low as 2 for walking motion or as high as 14 for Salsa dancing. Divisive clustering of motion segments according to their feature vectors yielded 65 clusters, i.e., 65 local linear models. Principle marker positions were used to classify the frames into the most appropriate local linear models *via* the Random Forest classifier. The classification error rate was 0.29%.

The reconstructed motions were visually plausible for all the testing sequences. There was no visible jerkiness at the transitional poses. My method performed reasonably well for the motions acted out by new subjects who never appeared in the database. I compared my method to the nearest neighbor search method with respect to root-mean-squared (RMS) reconstruction errors and jerkiness. In general, my method produced much more accurate results with less jerky estimates of motions. The average RMS error and velocity error over all the testing sequences were 45 mm/marker and 20 mm/marker, respectively, with my method. The corresponding errors with the nearest neighbor search method are 56 mm/marker and 76 mm/marker, respectively.

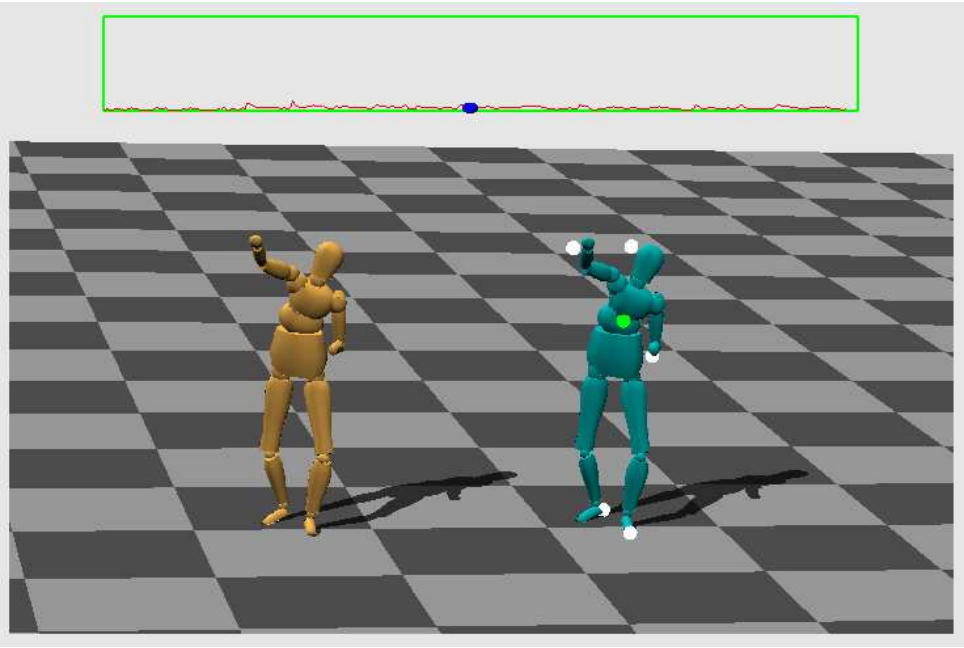


Figure 4.6: Shown above is a snapshot from my motion model viewer. The golden model on the left represents the actual pose data. The cyan model on the right shows an estimate of this pose based on the principal markers, which are depicted as white disks. The green disk indicates the origin marker. An RMS error meter for the entire marker set appears above the models with a full-scale value of 200 mm/marker.

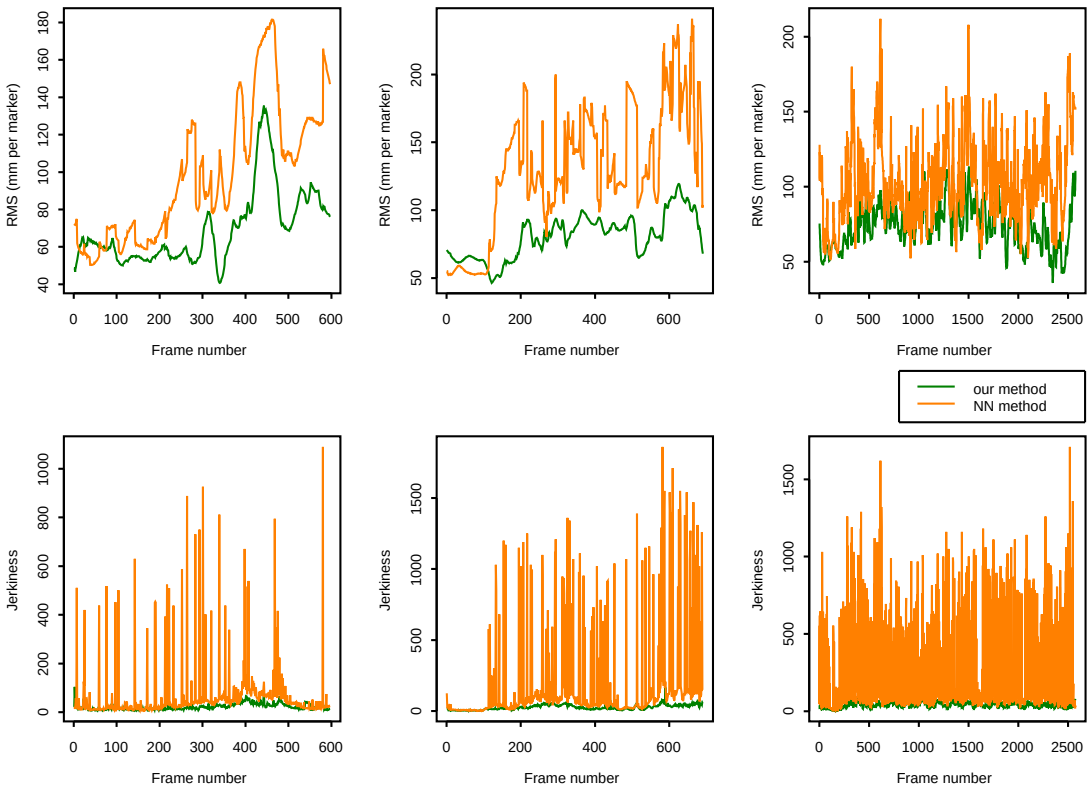


Figure 4.7: Comparison of my method to the nearest neighbor search method in estimating three motion sequences. The top row compares the reconstruction RMS error (mm/marker). The second row compares the jerkiness (i.e., velocity error γ as previously defined).

Figure 4.7 showed frame-by-frame RMS errors and jerkiness for three testing sequences with poses estimated using both my method and the nearest neighbor search method. The RMS error curve was much smoother using my method than the nearest neighbor search method. In particular, the nearest neighbor search method had a lot of spikes in the reconstruction error curve, which could have indicated severely jerky artifacts, confirmed by the frame-by-frame jerkiness curve. In fact, my method reduced the jerkiness by 80% in most of the sequences. Visual inspection of the reconstruction results also confirmed my conclusion.

I also demonstrated in my experiments that the motion reconstruction by my method was very fast. With the Random Forest classifier, the classification time was 0.00012 sec/frame, while the linear pose reconstruction time was 0.0014 sec/frame. This brings the total time needed for estimating a pose from a set of principal markers to 0.0015 sec/frame, or over 600 frames per second. I ran my experiments in Matlab V7 on a Dell Inspiron Laptop, with 1.4GHz CPU and 512M physical memory. A more powerful computer and more efficient code implementation may push the performance higher.

4.6 Conclusions

I presented a piecewise linear modeling approach to human motion data that were parameterized by a set of principal markers. I learned local linear models and principle markers from a training set of human motion data samples. The whole motion reconstruction process was efficient, with no need to search in a motion database. The experimental results demonstrated that this method can quickly generate plausible human motions on a frame-by-frame basis, and scales well with size and heterogeneity of motions. Thus, I believe it is possible to use only a few markers as control signals for interactive computer applications.

I identified a low-dimensional, local linear space at the motion segment level instead of at the frame level. Motion segments offer a more appropriate resolution for motion data modeling to retain temporal relationships to some extents by grouping temporally adjacent yet spatially homogenous frames together into one local linear model. Fewer local linear models are needed when modeling with motion segments than with individual frames, resulting in a more compact model hierarchy. It also improves the reconstruction quality by reducing unnecessary model transitions, a primary source of temporal discontinuity, i.e. jerkiness.

Modeling human motions in the mocap marker space pushes motion data processing a step closer to raw data measurements, eliminating skeleton estimation, skeleton calibration and potential information loss during the conversion of marker measurements to joint angles. On the other hand, there may be a normalization issue with the use of marker data due to size differences among human subjects. Nevertheless, the experiments showed that the performance of the proposed method was not sensitive to normal variations in subjects' sizes. In the experiments equivalent motions from different subjects tend to lie in the same local linear space, so the corresponding mapping function is actually computed based on data from different subjects. Calibration of subjects of different sizes does not appear to be essential with this marker-based approach. However, more experiments are needed in this regard.

I presented an algorithm for selection of a principle marker set. People may also want to follow their intuitions or experiences to select the principle markers, for example, on the extremities. It is of interest to compare the results obtained from automatically selected markers with those from the manually selected markers. Missing markers are often encountered in mocap data. It is desirable to use a training set with complete and precise marker measurements to learn a reliable model. However, in reconstruction of a new motion sequence, my method potentially allows for missing principle markers because Random Forest has efficient imputing methods to replace missing values. It is worth conducting experiments to see to what extent the missing principal markers are allowed to retain an acceptable motion reconstruction.

Chapter 5

Estimation of Missing Markers in Human Motion Capture

A common problem encountered in motion capture is that some marker positions are often missing during the course of motion capture. As mentioned in the previous chapters, an optical mocap system utilizes video cameras to track the movements of a set of reflective markers that are strategically attached to the actor's body. The $3D$ marker positions are estimated *via* triangulation from multiple cameras. A marker is considered missing if it is not visible to at least two cameras.

A major cause of missing markers is that a marker is occluded by props, limbs, bodies or other markers. Also, it is not unusual that some markers can be missing for long periods of time. Although many methods have been developed to handle the missing marker problem and are already in use in commercial mocap systems, most procedures require manual intervention and are not satisfactory with problems of diverse motions, a high percentage of missing markers, and/or extended occlusions. In this chapter I propose a method for missing marker estimation that is especially appealing under these situations.

Previous missing marker estimation methods are very effective in recovering missing marker positions if markers are only missing for a very short period of time. However, most methods quickly become ineffective or even inapplicable when a significant portion of markers is missing for a long period of time, or missing at the very beginning or the end of a motion sequence. The method proposed in this chapter complements those methods by allowing arbitrary markers to be missing for a considerable period of time, while still being able to recover their positions using all the available marker positions.

Without assuming any skeleton model, I learn a global model as well as a hierarchy of local linear models from a training set that contains sufficiently representative motion

sequences. I then take a two-stage, *coarse-to-fine* approach to estimating the missing marker positions of a new sequence. I apply the global linear model to obtain a coarse estimation at the first stage, and I then refine the estimation *via* local linear models at the second stage. This method is very simple, fast and robust in recovering missing markers and estimating human motions. Most importantly, it allows different sets of markers to be missing for a *moderate-to-long* period of time. In the experiments I demonstrate that this method can successfully estimate missing markers over a variety of motions from multiple subjects.

This chapter is organized as follows: in section 5.1 I give an overview on the algorithms and briefly discuss the key components, including global linear modeling, piecewise linear modeling, classifier training and the motion reconstruction. Then from section 5.2 to 5.4, I describe these key components in details. I present the experimental results in section 5.5. I conclude this chapter with discussions in section 5.6.

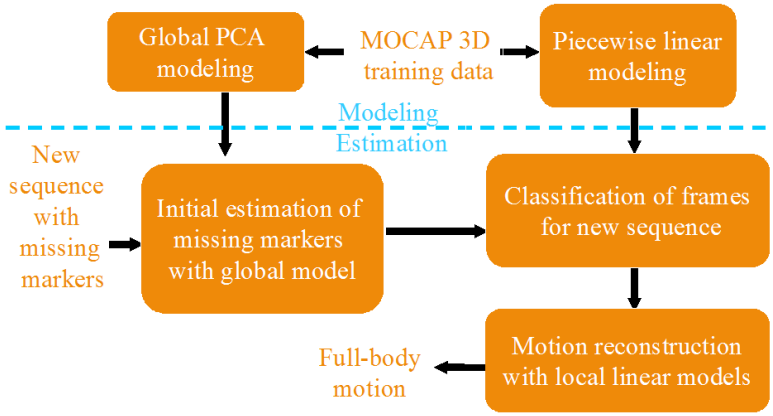


Figure 5.1: Motion data modeling and missing marker estimation process.

5.1 Overview

There are two essential components in the process of missing marker estimation (Figure 5.1): modeling training data and estimating missing markers for new sequences. A training dataset contains sufficiently representative examples of motions. I take a two-stage modeling approach. At stage 1 I model motion data as a single global linear model, represented by the principal components. At stage 2 I model motion data in a refined fashion by a collection of local linear models, which together form a model hierarchy. Given a new sequence with missing data, I first fill in missing marker positions with

the approximations derived from the global linear model. Next, for each frame with initially filled-in values, I identify the most appropriate local linear model *via* a classifier trained at the modeling stage, and I then make a refined estimation for the missing markers by obtaining a least-squares solution from the known markers and the principal components associated with the local linear model.

5.2 Global Linear Modeling

Global PCA modeling is the first and the coarser stage of the modeling process. At this stage a single linear model is constructed by applying PCA to the whole training set. I compute the principal components by applying singular value decomposition (SVD) (Press et al., 1986) on data matrix Y and form a $3m \times d$ eigenvector matrix P, with its d columns being the leading d principal components. The eigenvector matrix P as well as the mean vector μ will be used to calculate the initial estimates of the missing markers.

5.3 Piecewise Linear Modeling

Piecewise linear modeling is the second stage of the two-stage modeling process. It is used to provide a refined estimation of missing markers based on the coarse estimate from the global linear model at stage 1. The piecewise linear model described here is also a modified version of the general PLM approach presented in Chapter 3. First I segment motion sequences into segments of simple motions and characterize each motion segment with a feature vector derived from the mean vector and the covariance matrix of the pose data of each segment. Next, I group similar motion segments *via* divisive clustering of the feature vectors. Finally, I construct a local linear model for the poses in each cluster by their mean vector as well as by their principal component vectors. Among these key components, segmentation and characterization are implemented exactly the same way as described in Chapter 3, while local linear modeling and classifier training are modified to accommodate the situation in this application. In this section, I only describe how I compute the local linear models and train the classifiers.

5.3.1 Classifier training

I train a Random Forest classifier(Breiman, 2001) to classify frames of a new sequence into different local linear models that are extracted from the training set. Random Forest (RF) is a powerful classification tool that grows and combines decision trees into predictive models. The overall prediction is determined by voting over all the trees in the forest and by choosing the class having the most votes. For each frame labeled with a model ID, instead of using only a subset of markers (principal markers) as in *human motion estimation from a reduced marker*(Chapter 4), I retrieve all of its marker position values and use them as input variables for training the RF classifier.

5.3.2 Local linear modeling

For the motion segments partitioned into the same group, i.e. cluster, a single linear model is constructed by applying PCA to all the poses belong to these segments. To compute each local linear model, I retrieve a mean pose of all the poses. I then compute PCA to obtain an eigenvector matrix P_i for the i^{th} local linear model and take the leading d principal components, i.e. the leftmost d columns of the matrix P_i. Finally, I save the mean vector and the principal components.

5.4 Missing Marker Estimation

I take a two-step, coarse-to-fine approach to estimating the missing marker positions of new motion sequences. In the first step, I apply the global PCA model computed from the training set to obtain a coarser estimation of the missing markers. Then a frame, with missing markers replaced by the initial estimates in step 1, is classified into the most appropriate local linear model *via* the RF classifier. I next find a least-squares solution to the projection of the frame onto the principal component space associated with the identified local linear model. Finally, I transform them back to the original marker space to obtain the estimates of the missing marker positions. I will explain in more detail on these two estimation stages.

5.4.1 Estimation with the global linear model

Given a frame with all the known markers correctly labeled and the rest of the markers missing, I compute a least-squares approximation of the missing marker positions from

the available (known) marker positions using the global PCA linear model. Let the mean vector of the global linear model to be μ, with μ^a and μ^m being the parts of μ corresponding to the available and missing unknown markers, respectively. I first retrieve the $3k \times 1$ position vector of the k known markers, f, and obtain a centered vector s by subtracting from f the corresponding part of the mean vector of the global PCA linear model. Then I form a $3k \times d$ matrix U from the eigenvector matrix P by taking the entries corresponding to the known markers, with a $3(m-k) \times d$ matrix V taking the remaining entries. If I let a $d \times 1$ vector, w, be the projection of a frame on the leading d principal component axes, I compute a least-squares solution to w according to

$$Uw = s \tag{5.1}$$

and estimate the $3(m-k) \times 1$ position vector of missing markers, x, as

$$x = Vw + \mu^m. \tag{5.2}$$

The least-squares solution to w is

$$w = U^T(UU^T)^{-1}s, \tag{5.3}$$

and thus

$$x = VU^T(UU^T)^{-1}s + \mu^m. \tag{5.4}$$

However, such an initial estimate of missing markers from one global model may be too coarse, especially when the database is a large, heterogeneous motion data set where various types of motions are included. So it is crucial to use the local linear models at the next stage to refine the estimation result of at this stage.

5.4.2 Estimation using the local linear models

Once I fill in the missing marker positions of a frame with the estimated values at stage 1, I classify this updated frame, consisting of full marker positions, to the most appropriate local linear model by the Random Forest classifier. I then retrieve the mean vector and the eigenvector matrix associated with the local linear model to estimate the missing marker positions through a least-squares solution method as described at stage 1. Let s_i be the centered position vector of the available markers from the i^{th} local linear model. Similar to the definition of μ above, I denote the mean vector of the

i^{th} local linear model as μ_i with μ_i^a and μ_i^m being the parts of μ_i corresponding to the available and and missing markers, respectively. For a particular pose classified into the i^{th} local linear model, I estimate the missing marker positions z as

$$z = V_i U_i^T (U_i U_i^T)^{-1} s_i + \mu_i^m, \tag{5.5}$$

where U_i and V_i are the two matrices formed by taking the corresponding entries from the eigenvector matrix P_i.

When modeling time series data, an inherent drawback of piecewise linear modeling approach is temporal discontinuity at the transitions between two different linear models. As a solution I incorporate a mixture of local linear models associated with the previous poses to smooth out the jerkiness at the transitional poses, and re-estimate z as

$$z = \sum_j w_j V_j U_j^T (U_j U_j^T)^{-1} s_j + \mu_i^m, \tag{5.6}$$

where $w_j = r_j/(h+1)$ is a weight for the j^{th} model, and r_j is the number of poses classified to the j^{th} model among the prior h poses and current pose. The rationale behind this is that I want to put more weights on the model that is favored by more of the current pose and its previous h poses. In my experiments, $h = 10 - 30$ works well.

5.5 Experiments

I divided data into a training set and a testing set. The training set consists of 132 motion sequences with a total of 151,882 frames collected from 21 subjects. I included a variety of motions (i.e., walking, running/jogging, golfing, soccer kicking, Salsa dancing, jumping, cartwheel, climbing steps), as well as different styles of the same motion from different subjects. Segmentation of the training sequences yielded 271 segments, with length varying from 128 to 3,670 frames (mean: 560; standard deviation: 425; median: 440). All the motion data were preprocessed by a normalization procedure, as described in the previous chapter, to remove the effects of translation and rotations of the poses. Hierarchical clustering of segments according to their feature vectors produced 65 clusters, i.e., 65 local linear models. I retained the leading 15 principal components to approximate the poses of each local linear model.

I used a testing set to validate my method. The testing set contained 28 sequences with 19,553 frames from 18 subjects. Among them, there were 9 walking sequences, 6

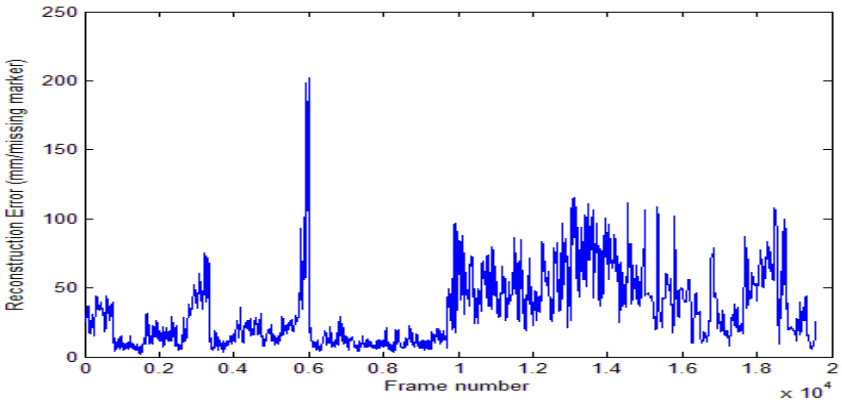
running, 5 golfing, 2 cartwheel, 2 Salsa dancing, 1 walking on uneven terrain, 1 running jump, 1 soccer kicking, and 1 climbing three steps. None of the testing sequences were included in the training set. Four testing sequences, namely, 1 walking, 1 soccer kicking, 1 running and 1 golfing were from 4 new subjects who never appeared in the training set.

I assessed the performance of my method with different number of markers missing (i.e., 5, 8 10, 15 and 20) in each frame of the testing sequences. In each experiment, for every testing motion sequence, I randomly chose a fixed number of markers to be missing for a period of 1 second (120 frames). For example, in the first experiment I randomly chose markers 1, 15, 21, 32, 38 to be missing for the 1st second and 2, 7, 12, 29, 40 to be missing for the 2nd second.

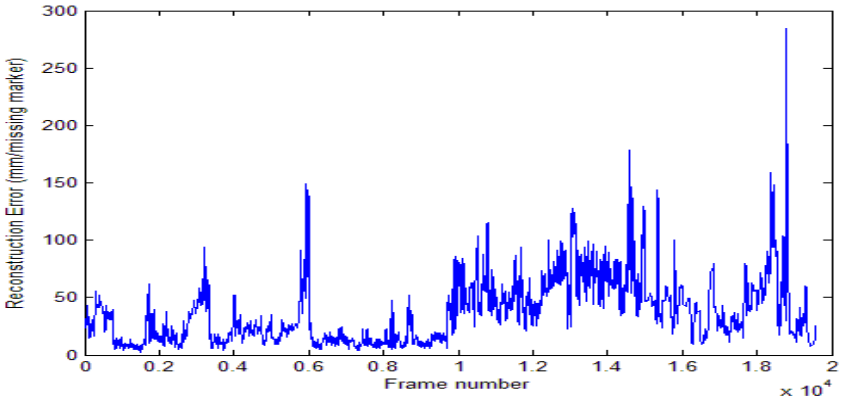
I also compared my method to spline interpolation in two scenarios where there were always 8 markers missing in the middle of a sequence, with the missing marker set being changed for every second. However, in the first scenario, full marker positions were known at the two ends of the motion sequence, while in the other scenario, 8 randomly selected markers were also missing for a period of time either at one end or two ends of a sequence.

Figure 5.2 shows the frame-by-frame root-mean-squared (RMS) errors (mm per missing marker) of each experimental result. It appeared that the reconstruction errors were minimal when there were 8 markers missing. Although the errors increased with increasing number of missing markers, the magnitude remained acceptable. Even when the number of missing markers reached 20, i.e., 50% of the total markers, the reconstructed motions were still plausible. I tested my method against various types of motion sequences that were not included in the training set. The results showed that my method was robust enough to produce reasonably good estimation to these heterogeneous motions. However, I also observed that the estimation results may appear unsatisfactory when many important markers were missing together, e.g., when all the markers on one leg and arm were missing.

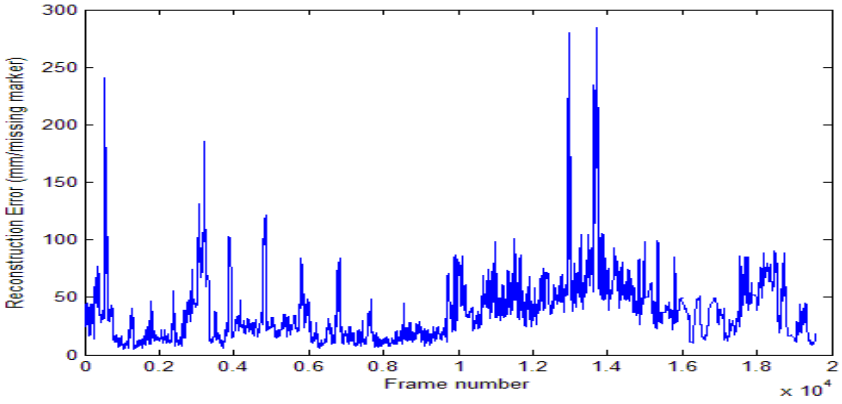
I showed in Figure 5.3 the reconstruction results for each coordinate of two missing marker positions in a short segment of a motion sequence, using both the spline interpolation method and my method. When a marker was missing in the middle of the sequence, shown in the top panel of Figure 5.3, my method recovered sufficient details missed by spline interpolation. In another example where a marker was also missing at the very beginning of the sequence, as shown in the last two panels of Figure 5.3, spline extrapolation completely failed due to the fact that there was only support on



(a). 8 missing markers.



(b). 15 missing markers.



(c). 20 missing markers.

Figure 5.2: Marker reconstruction errors in the scenarios of different numbers of missing markers.

Number of missing markers	Global PCA estimation	Local linear model classification	Local model estimation	Total time
5	0.52	15.36	1.90	17.78
8	0.50	14.98	1.94	17.42
10	0.50	15.32	1.95	17.77
15	0.46	15.25	1.93	17.64
20	0.42	15.61	1.95	17.98

Table 5.1: Running time (ms/frame) of estimation procedure when various numbers of markers are missing.

one side of the missing frames. In contrast, my method was still able to recover the missing marker reasonably.

One advantage of my method is that the estimation procedure can run very fast after off-line motion modeling using the training set. In Table 5.1 I show the distribution of the time spent at each key step. It only takes on average less than 18 milliseconds to estimate the missing marker positions per frame. That is over 50 frames per second, well above the typical interactive frame rate (i.e., 30 frames/second). It also appears that the total estimation time per frame remains about the same despite the increasing number of missing markers. I ran my experiments in Matlab V7 on a Dell Inspiron Laptop, with 1.4GHz CPU and 512M physical memory. A more powerful computer and more efficient code implementation may push the performance much higher.

Due to the fact that there is a lack of real datasets with missing markers, I have to randomly remove the measurements from some markers to create missing markers at each frame. However, this random missing marker setup may not be a perfect reflection of the real missing marker scenario, where it is perhaps more likely that neighboring markers tend to be missing together. For example, most markers on a arm or a leg are more likely to be missing at the same due to occlusion. In the future I would like to have an opportunity to gather real missing marker datasets to demonstrate my method's utility on those datasets.

5.6 Conclusions

I presented a piecewise linear modeling approach to estimating missing markers in human motion capture data and reconstructing plausible human motions. I learned the local linear models from a training set without prior knowledge of the human

skeleton. I exploited the correlations among mocap markers to infer the missing marker positions from the positions of the known markers. The motion reconstruction process was efficient, with no need to search in a database or to estimate/calibrate a skeleton model. The experimental results demonstrated that my method can quickly generate plausible human motions on a frame-by-frame basis without any manual intervention.

This method complements the interpolation-based methods in that it consistently produces reasonable estimation of missing markers even when the missing time gap is so long such that the interpolation methods become ineffective or inapplicable. It also achieves better estimation than the spline interpolation methods when the frames at either ends of a sequence have missing markers. On the other hand, this data-driven, piecewise linear modeling method has limitations similar to other data-driven modeling approaches. It assumes that the training set both adequately samples and is representative of the data space. Moreover, its ability to extrapolate from the training data is more limited than its interpolation capabilities. The notion, however, of an interpolation system that is limited by its underlying model is not unique to data-driven methods. Kinematic models are also limited by the accuracy with which they represent linkages and by their motion ranges.

I limited my model to only marker positions and ignored velocities and accelerations. Using more information could improve my model; however, in my approach, adding more data may also increases the dimensionality of the problem. This implies the need for even more data, and I are already undersampled. This increases the likelihood of overtraining my model, thereby limiting its ability to generalize, as discussed earlier. One of the strengths of my models is that it is very simple and fast. There are few parameters to be tweaked during the modeling phase. Incorporating velocity and even acceleration may make the model too complicated and slow down the estimation. Another reason that prevents us from using the acceleration and velocity is the concern of the accumulation of errors. Computing the acceleration and velocity requires the knowledge of the positions of the previous frames. However, some markers in the previous frames may have been missing and have to be estimated as well. So these frames may not be accurate enough to be used to estimate the current frame. I are concerned that this may in fact affect the estimation of the current frame due to the accumulation of errors. In my opinion only the available marker positions of the current frame have the most accurate information since they are actually measured. They should therefore play more important roles in estimating the other marker positions.

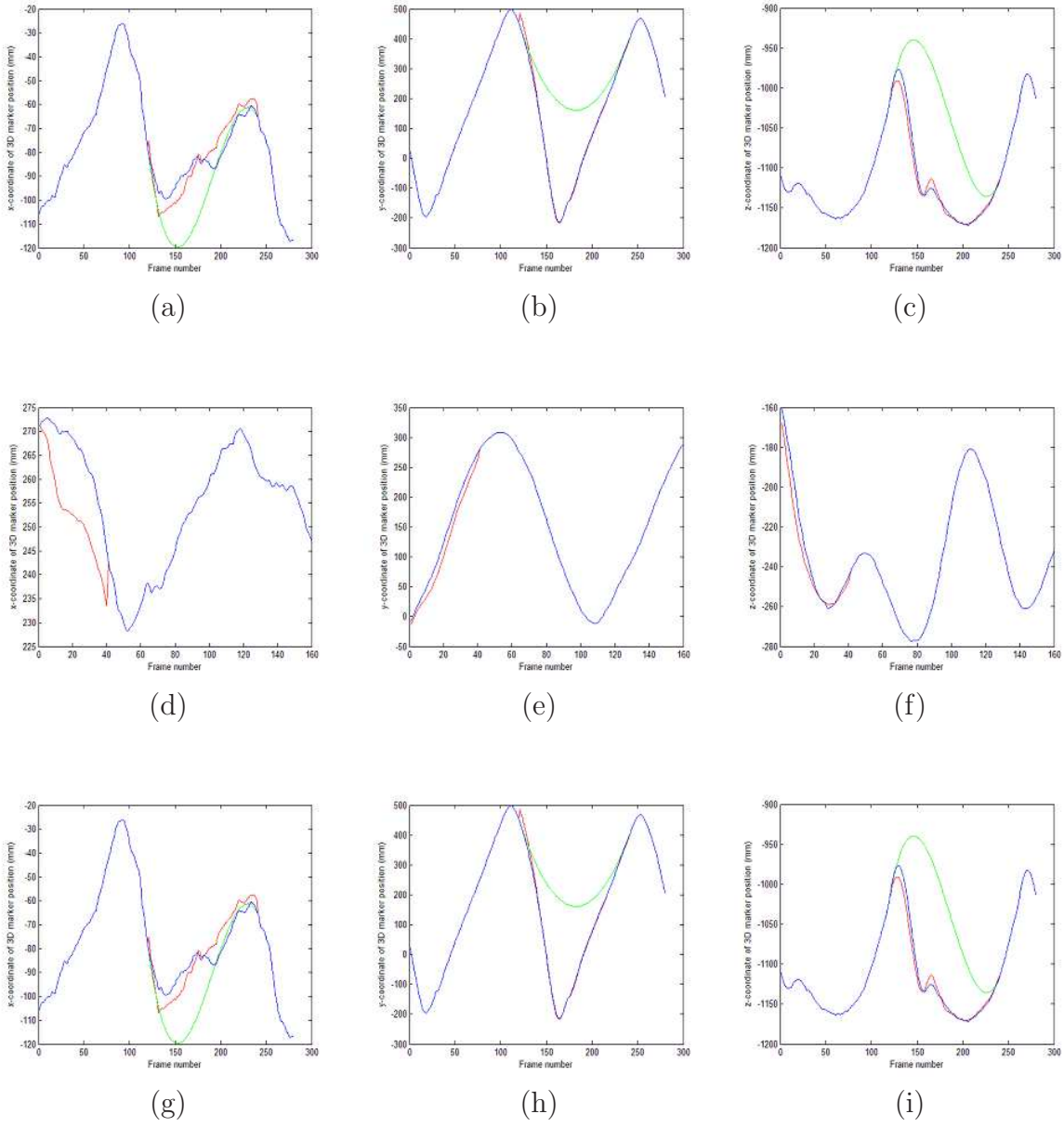


Figure 5.3: Comparison of estimating results (original motion in blue, spline interpolation results in green and my estimation results in red). In each figure the horizontal axis indicates the frame number, while the vertical axis indicates the reconstruction error per marker. The top panel with Figures a, b and c, corresponds to the marker on the right ankle; the middle panel (Figures d, e and f) and bottom panel (Figures g, h and i) correspond to the marker on the left arm. The middle panel only shows the original marker positions and my estimations, while the bottom panel shows the original marker positions, spline interpolations and my estimations, respectively, in a larger scale.

Chapter 6

Motion Sequence Retrieval Based on Behavioral Similarity

Human motion capture data have been widely used in many research fields and applications. However, in order to reuse the existing motion capture data appropriately, one has to be able to efficiently find the suitable motion clips from a motion database. As more and more human motion capture data becomes available, motion databases grow larger and larger. There is an imperative need for the tools to organize large motion databases and support fast retrieval of similar motions.

In designing a scheme to organize a motion database that supports fast motion data retrieval, I have to define similarity among motion sequences. Motion sequences can then be compared and grouped by how similar or dissimilar they are to each other. For time series in general, similarity is often defined by the overall distance between the spatio-temporal curves under a certain distance metric. However, as a special type of time series, motion sequences may be perceived to us as similar even if they may not be closer under certain distance metrics. For example, two walking sequences may be perceived as similar even though they considerably differ in their speeds and styles. I define a similarity measure in terms of *behavioral* similarity perceived by humans, as opposed to *numerical* similarity measured by various distance metrics. Behavior-based similarity metric appears to be more suitable for an activity identification task where the main concern is to determine what behavior the person is engaging, while stylish detail differences are less weighted. My similarity measure is a higher-level abstraction of motion similarity. A similar concept was also considered by (Müller et al., 2005) for content-based retrieval of motion data. They defined logical similarity by using a class of Boolean features to express the geometric relationships among a particular set of joints. However, geometric features have to be selected manually and appropriately,

and the actual quantitative information of motion data may not be fully used to model the data.

I take advantage of the strengths of both numerical and logical similarities. In particular, I apply the piecewise linear modeling approach to modeling human motions at behavioral level and use similarity in behaviors to establish an indexing system for similar human motion sequence retrieval. I first segment long motion sequences into segments of distinct behaviors. I model each motion segment quantitatively with a feature vector derived from the overall statistical properties of the poses. These derived feature vectors serve as a statistical signature for the corresponding motion segments. I then construct a compact but effective indexing scheme in the feature space for efficient retrieval of similar motions. Given a new motion sequence, I segment it into single behavioral segments whose feature vectors are then derived and used to retrieve similar motions from the database. The process of data modeling and the construction of indexing structure is fully automatic. The resulting indexing structure is very compact as compared to the original motion database. This method is also robust to spatio-temporal variations among similar motions, and thus eliminates the use of time-warping techniques. The experiments show that this method is efficient and flexible in organizing, indexing and querying a large motion database based on similarity in behaviors.

The rest of the chapter is organized as follows. I give an overview of the proposed method in Section 6.1. In Section 6.2 I describe the indexing scheme constructed by the piecewise linear modeling approach. In Section 6.3 I discuss the a process of querying a motion database indexed by this method. I then present the experimental results in Section 6.4. Finally, in Section 6.5 I conclude with discussions.

6.1 Overview

My goal is to construct a motion database indexing scheme to support fast queries for similar motions. I first model the motion sequences and then establish a hierarchical index structure to facilitate the motion retrieval. The following is a brief overview on each module of my method (Figure 6.1), with more details given in the later subsections.

Normalization. Normalization converts all the poses from different coordinated systems onto a universal model-rooted coordinate system, where all the translational and orientational effects are removed from the poses through appropriate transformations.

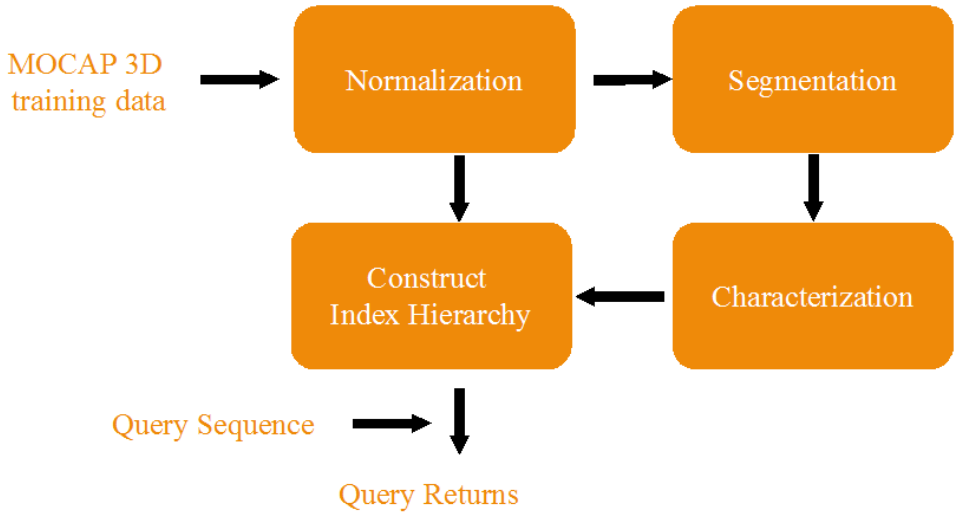


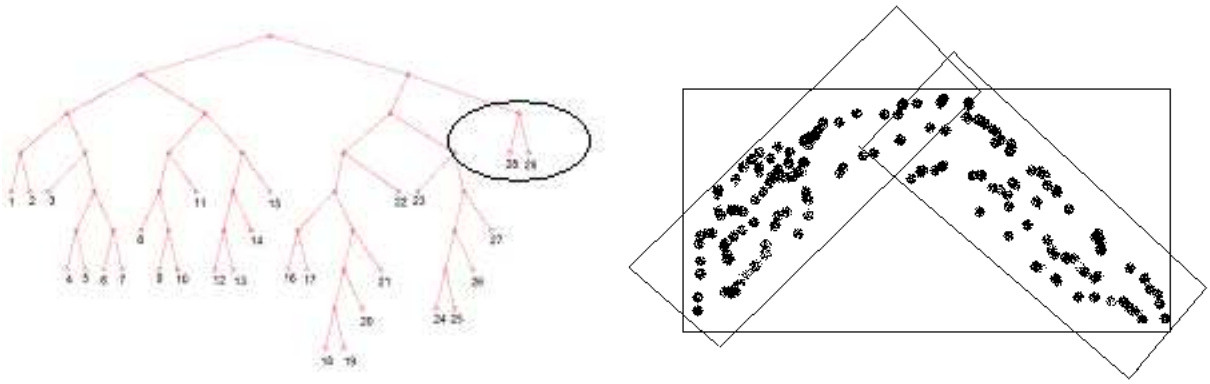
Figure 6.1: Key component diagram of the motion databases retrieval.

Motion segmentation. Motion segmentation divides a complex motion sequence into one or more segments of distinct behaviors. I apply the Probabilistic PCA (PPCA) (Tipping and Bishop, 1999) for motion segmentation.

Motion segment characterization. For each motion segment corresponding to a simple behavior, I characterize it with the statistical distribution of its poses. In particular, I extract a feature vector from the mean vector and the covariance matrix of each motion segment. Motion segments associated with similar behaviors have similar feature vectors.

Indexing. I use feature points of the segments as modeling primitives to construct a compact indexing scheme of a motion database. I take a divisive clustering approach, which recursively partitions the points in feature space into subsets until a preset threshold is reached. At each partition level, I compute minimum bounding rectangles, i.e. MBRs, that not only enclose all the data points associated with the node, but also orient along the directions of maximum variance spreads. This strategy results in much tighter bounds than the minimum bounding envelopes using traditional axis-aligned bounding boxes.

Query. If a query is already a simple motion consisting of a single behavior, I may directly extract a feature vector from the query sequence. This feature vector corresponds to a point in the indexing space, where I then search for the closest matches to the query. If the query is a complex motion containing more than one behavior, I first segment the query motion into distinct behaviors; then, for each segmented simple



(a). (b).

Figure 6.2: Illustration of the hierarchical indexing structure: (a) shows a model tree, with one internal node being circled; (b) shows that associated with the circled model tree branch, all of the branch's data points can be completely bounded by a bigger OBB box and by two smaller and overlapped OBB box.

motion, a set of closest matches are sought from the indexing hierarchy. Finally I rank the returning candidates with their overall closeness to the query sequence.

Since normalization, motion segmenting and characterization have been presented as the key components of a generic PLM modeling framework in Chapter 3, I will not explain those component again in this chapter. Instead, I will describe more details on indexing and query in the following sections.

6.2 Indexing

An efficient indexing scheme may greatly facilitate the retrieval of information. I create an indexing structure that can provide easier, faster and more robust retrieval of human motion data without directly comparing high dimensional motion sequences at all.

My indexing structure is constructed in a feature space, with each feature point representing a motion segment. The closeness of points is measured *via* the Euclidean distance. The closer the two feature points, the more similarity in behaviors between the two motion segments represented by the feature points. The goal is to group

together the closest feature points, i.e., motion segments with similar behaviors. I partition a data set into a hierarchy using a divisive clustering method. I start from the root node corresponding to a whole data set of feature vectors and recursively partition the data evenly into two subsets until the number of the feature points at the node is less than a preset threshold. At each internal node, I conduct PCA and then project the data points onto the first principal axis, i.e. the eigenvector with the largest eigenvalue. Next, I split the data around the median of the projected data. Splitting by the median typically results in a more balanced tree structure. It is also more robust against outliers.

Pruning is a popular strategy to speed up a query process by avoiding expensive searches on apparently unmatched sequences in a database. I take an approach similar to the OBB-Trees (Gottschalk et al., 1996). At each node I compute PCA out of the data points in the feature space, and then form a minimum bounding rectangle (MBR) in the k-dimensional space spanned by the leading k eigenvectors. This typically results in a MBR with a tighter bound of the enclosed data instance. Figure 6.2 illustrates the hierarchical indexing structure, where the feature points are clustered divisively, and the corresponding MBRs are constructed accordingly to each partition level. The partition process continues until a desired granularity is reached at every leaf node.

Once a motion database and the associated indexing structure are established, subsequent updates are allowed and can be done either dynamically or in a batch mode. A batch mode update is recommended to be done only when a significant amount of new data become available and must be added to the database. In this case the indexing hierarchy has to be rebuilt all over again using the feature points of both the old data and the new data. A dynamic update, on the other hand, can be done at any time in between the batch mode updates, with an assumption that the data being updated are small in size as compared to the current database, so that the distribution of the database will not change dramatically with the new data being added. Now I describe two dynamic operations, insertion and deletion, respectively.

Dynamic Insertion. A dynamic insertion adds in feature points, one at a time, to the indexing tree. When a new motion sequence becomes available and needs to be added to an existing motion database as well as the associated index hierarchy, I first segment the motion sequence into motion segments of single behaviors. Then for each motion segment, a feature vector is derived from the mean vector, and the covariance matrix of the poses in the segment. Then I project this new feature vector to the lower-dimensional indexing space. The projection then becomes a new feature point in

the indexing space and is ready for the dynamic insertion. Starting from the root node, I recursively insert the feature point to the most appropriate node. At each node, with the associated cluster of feature points, I perform a dynamic insertion as follows:

1. Subtract the cluster mean of a node from the feature point to be inserted.

2. Project the mean-centered feature point to the principal component space spanned by the principal component axes associated with the cluster.

3. Check if the projection of the feature point is out of the original minimum bounding rectangle. If so, I update the boundaries of the affected minimum bounding rectangle to include the inserted feature point. If the node is an internal node, meaning it has two children, I then project the inserted feature point to the principal space of each child cluster. I then insert the feature point to the child node whose minimum bounding rectangle increases the least after inserting the feature point. The update stops if the node is a leaf node.

Dynamic Deletion. First, I simply delete the feature point from the leaf node to which it belongs. I then update the minimum bounding rectangle if any of its boundaries are affected (decreased) by the deletion of the feature point. Next, I find its parent node and perform the same deletion operation. The process continues until the root node is reached.

6.3 Query

A query motion sequence may be either a short simple motion sequence that corresponds to a single behavior or a long complex motion sequence including multiple distinct behaviors. I discuss the query process under these two scenarios, separately, in the following subsections.

6.3.1 Query for single-behavior motions

Querying for a single-behavior motion is simpler than querying a multi-behavior motion, since segmentation of motion sequences is not necessary. I first retrieve a feature vector from the mean and the covariance matrix from the query motion and project this feature vector onto the indexing space where it becomes a feature point. I then search

in the indexing space for those feature points that are the closest to the query point under the Euclidean distance metric.

In order to avoid unnecessary point-by-point comparisons and to improve the search efficiency, I prune the indexing hierarchy by setting up a search radius r for the query point. Starting from the root node, I recursively search through the indexing tree for the feature points that are within the search radius. The searching procedure is described as follows:

1. At each node project the query point to the principal component space associated with the node.

2. Check to see if the corresponding oriented minimum bounding box overlaps with the search area, i.e. the hypersphere defined by the query point and the search radius r. Stop further searching, and return no candidate if it doesn't overlap; otherwise, go to step 3.

3. If the node is an internal node, repeat step 1 for each of its two child nodes; otherwise, go to step 4.

4. Compute the distance of each feature point in the leaf node to the query point, and return the points whose distances to the query point are within the search radius r.

The returned candidates are then ranked based on their distances to the query point. The motion segments associated with these returned feature points are the matched sequences to the query.

6.3.2 Query for multiple-behavior motions

If the query is a long and complex motion sequence containing more than one behavior, I segment the query sequence into a series of motion segments of distinct behaviors before any query operation. For each segmented simple subsequence, I retrieve their feature vectors and project them onto the indexing space to get the corresponding feature points, and then use these feature points to query the indexing database for a set of closest matches applying the strategy described above. For each returned candidate, I identify its corresponding motion sequence in the database. I then align pairs of the query and each candidate sequence by matching them to the maximum proportion

on the basis of segment similarity. The optimal alignment can be cast into a string matching problem and solved by dynamic programming.

The term *Dynamic programming* was coined by Bellman (1957). Dynamic programming (DP) solves an optimization problem by first solving smaller subproblems and caching the optimal scores for the solutions of each subproblem instead of recalculating them. Every dynamic programming algorithm has three steps: initialization, matrix filling (scoring) and backtrack (alignment). In string matching by dynamic programming, the optimal alignments are computed for every substring and those scores are saved in a matrix. Then a *Backtrack* step is used to determine the actual alignments that result in the maximum score. As follows I will briefly describe these three steps in solving string alignment.

Initialization. Given two strings $s1$ and $s2$ of lengths M and N, I first construct a score matrix A with $M+1$ columns and $N+1$ rows. Then the entries in the first row and in the first column are filled with zero.

Matrix filling (scoring). At this step, for each entry of the matrix A, we attempt to find the maximum score, representing a substring alignment scenario. Start from the left-top corner, we define the score at the matrix position (i, j) as

$$A_{i,j} = max(A_{i,j} + S_{i,j}, A_{i,j-1} + w_M, A_{i-1,j} + w_N),$$

where $S_{i,j}$ is defined as the diagonal match/mismatch score, w_M and w_N are the gap penalty scores of strings $s1$ and $s2$, respectively.

Backtrack (alignment). The backtrack step determines the actual alignments that result in the maximum score. Starting from the right-bottom corner, the backtrack algorithm proceeds as follows:

1. Start from position $(N+1, M+1)$, i.e. the right-bottom corner, set $(N+1, M+1)$ as current position. Go to step 2.

2. Add current position (i, j) to the candidate position list. Go to step 5 if it is already the left-top corner; otherwise, go to step 3.

3. From its immediately adjacent neighbors, find the position (m, n) with the highest score. Go to step 4.

4. Set (m, n) as the current position. Go to step 2.

5. Compute and output the alignment from the candidate position list.

Candidate	D	A	B	E	C
Query		a	b		c

Table 6.1: Alignment of candidate sequences with a query sequence.

In order to match motion sequences using dynamic programming, I first need to define *match* between motion segments. A pair of motion segments are considered a *match* if their distance in the indexing space is less then a pre-set distance threshold. I convert each motion sequence into a string by assigning each motion segment a label. for example, A query sequence consists of three ordered segments, labeled "abc". One candidate sequence has five segments, also labeled "DABEC". Two labels are considered to be equivalent if they represent a pair of *matched* motion segments. I assign different scores for matched and mismatched scenarios. I also penalize gaps by assigning gap penalty scores. Now I am able to cast the motion sequence comparison problem into a string matching problem, which can be solved effectively by dynamic programming. Table 7.1 illustrates how two aforementioned label sequences are aligned, where (a, A), (b, B) and (c, C) are matching segment pairs, respectively. In this example, a motion sequence represented by "ABEC" should return as an answer for the query "abc". I finally rank the returns according to their overall similarity to the query motion sequence.

6.4 Experiments

I evaluated my compression method with Carnegie Mellon University's Graphics Lab motion capture database available at http://mocap.cs.cmu.edu. I prepared a large and heterogeneous human motion database consisting of 609 sequences with total 949,076 frames collected from 45 different actors over 300 minutes. The motion database contains a variety of motions such as walking, running, climbing stairs, cartwheeling and boxing, to name a few. Even for the same motion, I included different styles from different actors. The sequences in the database were segmented into 1,570 segments with lengths varying from 97 to 6,961 frames (mean: 605; standard deviation: 550; median: 440). Hierarchical clustering of segments, according to their feature vectors, yielded 160 clusters, each comprised of similar behaviors. The indexing structure was just 1.2M bytes vs. 675M bytes of the entire motion sequence database. I demonstrated my method with two types of queries: queries for simple motion sequences without segmentation and queries for general motion sequences with segmentation.

6.4.1 Query for simple sequences

Simple sequences are defined as those that require no segmentation to distinct behaviors. In other words, each simple sequence corresponds to a single behavior. Since motion segments obtained from the sequences in the database are such simple sequences, I ran the experiments with these segments as queries. I processed a query for each motion segment in the motion database, requesting various number of returns. I verified the similarity between the queried motion and the returned candidate motions by visual inspection of the motion sequence videos and by checking the sequence annotations from the original motion sequence database. The results in Table 6.2 show that (1) the average query time was about 1 second, and (2) the percentage of correct returns was over 90%. Dissimilar motions were occasionally returned but usually at the lower rank among the returned candidates. The queried motions themselves are always ranked the first in the returned list.

Returns per query	Search radius	Percentage of behaviorally similar motion	Average query time (Sec.)
1	0.02	100%	0.76
2	0.1	98.76%	1.03
5	0.15	97.4%	1.08
10	0.2	92.24%	1.07

Table 6.2: Summary of querying simple motions

I also showed two query examples, walking and cartwheeling, in Figures 6.3 and 6.4. I chose a representative pose from each returned sequence and plotted them together. In the walking query example, 20 matched sequences were returned, and all of them were walking motions. Most of the returns for a cartwheel motion query were also motion sequences with very similar behaviors, either cartwheels or flips. Only two out of 19 returned motions were not quite similar to the query motion. The appearance of dissimilar motions may be due to fewer similar motions available in the motion database. Additionally, the stylistic differences may overwhelm the behavioral similarity between two motions. Nevertheless, the returned motions with high degree of similarity usually ranked higher than the more dissimilar motions.

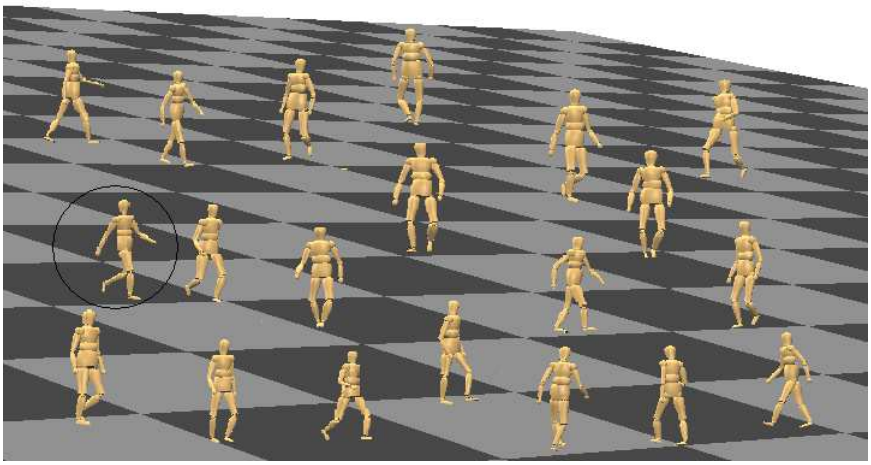


Figure 6.3: Selected frames from query returns for a walking motion. The query is highlighted with a circle.

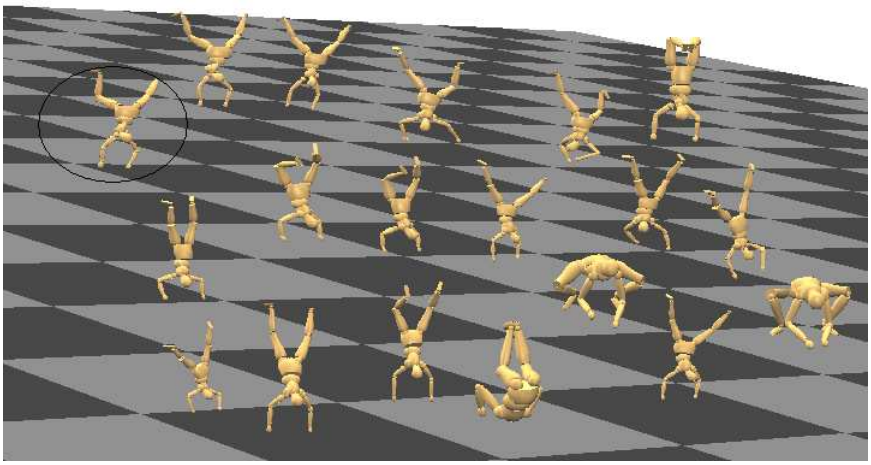


Figure 6.4: Selected frames from query returns for a cartwheeling motion. The query is highlighted with a circle.

6.4.2 Query for sequences with segmentation

In this experiment I considered each sequence in the database as a query and segmented the sequence before searching in the indexing space. The segmentation of a query motion sequence can be done either manually by the user or automatically by the algorithm. In my experiment I automatically segmented the query sequences into motion segments by the PPCA segmentation algorithm. The query performance on 609 sequences is given in Table 6.3. A query example of soccer kicking motion sequence was shown in Figure 6.5. As compared to the query for simple sequences, there was no

Returns per query	Percentage of behaviorally similar motions	Average query time (Sec.)	Segmentation time (Sec.)
10	88.26%	7.26	6.06

Table 6.3: Summary of querying complex motions with segmentation.

significant change in the percentage of returned similar motions and the rank of queried motion in the returning list (see Table 6.3). Although the average query time did increase to 7.26 seconds compared to about 1 second in the single behavior motion query, most of the query time (6.06 seconds) was spent on the PPCA segmentation. PPCA segmentation with larger steps Δ may shorten the segmentation time with no significant drops in query performance. In addition, a user can always choose to segment the motion manually before querying the database.

I ran my experiments in Matlab 7 on a Dell Inspiron Laptop, with 1.4GHz CPU and 512M physical memory. A more powerful computer and more efficient code implementation may push the performance higher to make the motion segmentation.

6.5 Conclusions

I applied a data-driven, piecewise linear approach to modeling human motions at a behavioral level. Based on this modeling approach, I designed an indexing scheme for efficient retrieval of behaviorally similar motion sequences from a large motion database. I believe that my method is a step forward in the study of human-perceived similarities among human motions. My method breaks down long motion sequences into segments of single-behavior motions. It then extracts equal-length feature vectors from the distributions of poses in the motion segments and uses them as the modeling primitives in a newly parameterized space. By doing so I can encapsulate the essence

of motions in a very compact but effective data structure. This model is immune to spatio-temporal variation among similar motions and is able to group together similar motions with different styles. The process of model construction and data query is efficient and scales well with data size and dimensionality.

Although my method does not guarantee that the returns for any particular query are in an absolute right order, more similar motion sequences do rank higher than the other *not-so-similar* motion sequences in most cases. I believe that my method not only provides an efficient way to organize and categorize a larger human motion database, but also can be used independently for motion retrieval tasks such as assisting the composition of animation sequences in video games.

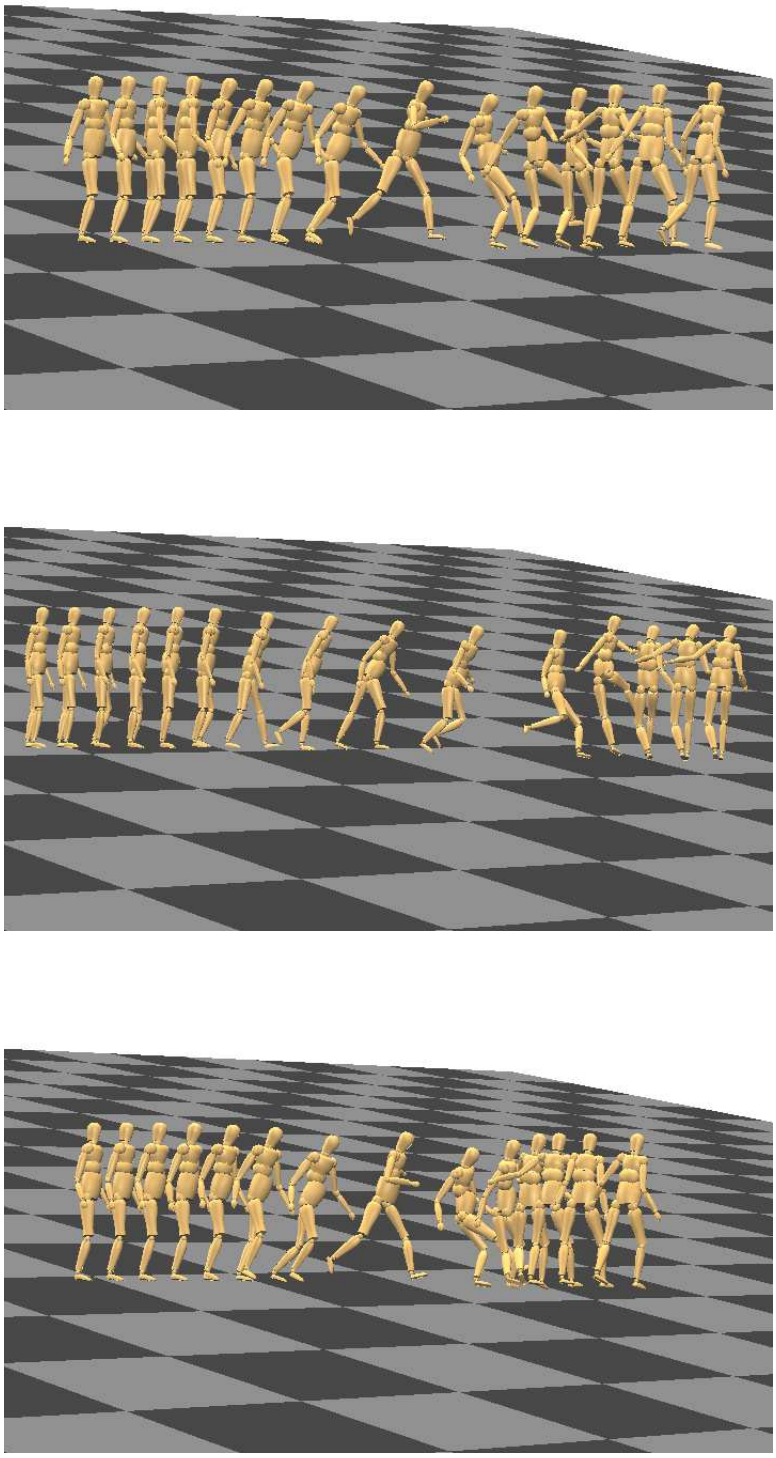


Figure 6.5: A snapshot of a soccer motion query. The first sequence is the query. The other two sequences are the returning results.

Chapter 7

Segment-Based Human Motion Compression

Human motion data have been used in many research fields and applications, such as animating human-like computer characters in video games, driving avatars in virtual reality environments, and generating special effects in movies. In particular, online video games often use motion data to interactively control the game characters from a remote site across the internet. As more and more motion data become available, compressing motion data for compact storage and fast transmission becomes imperative.

In order to achieve a greater compression ratio while still being able to retain high fidelity to the original motion sequences, I propose a motion compression method that exploits both spatial and temporal coherences inherent in a human motion sequence. First, I segment a motion sequence into segments of simple motions. Poses from each motion segment lie near a space with low linear dimensionality. I then compress these motion segments individually by PCA approximation. I compute PCA from each motion segment and approximate pose position vectors by their projections onto the space spanned by the principal components. This segment-based PCA compression typically needs fewer principal components than compression using global PCA on a whole sequence to achieve similar distortion ratio. To take advantage of temporal coherence and further compress the PCA projections of the poses of each motion segment, I adaptively select and store only the key frames from each motion segment and use them as the control points for the cubic spline interpolation in the principle component space. This motion sequence compression method is efficient and easy to implement, with a corresponding decompression process that is simple and fast as well.

The rest of the chapter is organized as follows. In Section 7.1 I give an overview of the proposed method. In Section 7.2 I develop an algorithm to compress motion segments by PCA. I then describe how to achieve further compression by selecting the key frames in Section 7.3. In Section 7.4 I provide a decompression algorithm. I present

the experimental results in Section 7.5 and finally conclude with discussions in Section 7.6.

7.1 Overview

Figure 7.1 shows a flow chart of the compression and decompression pipeline. An overview of each component in the compression/decompression process is given below.

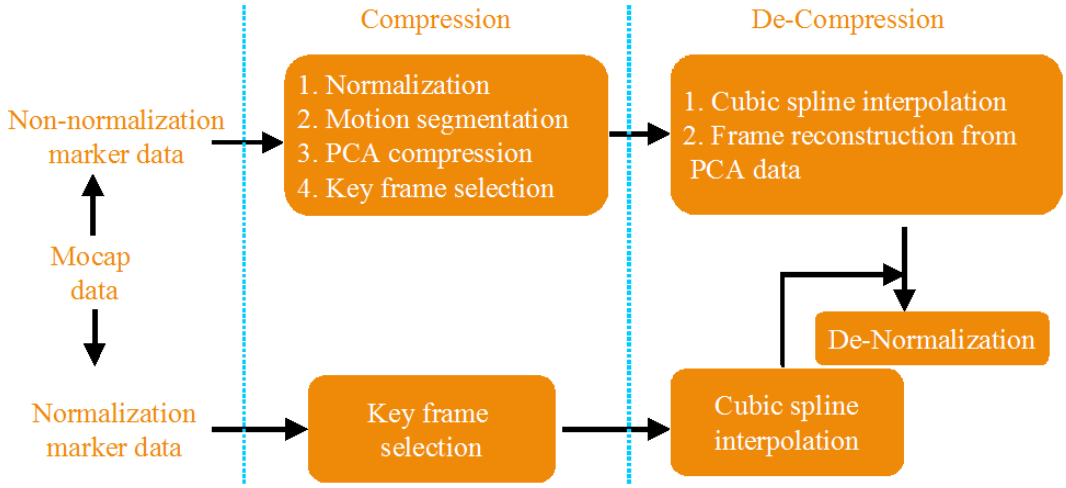


Figure 7.1: A diagram of motion data compression.

Normalization. In order to achieve high compression performance, I apply a normalization procedure (already described as a key component in PLM modeling pipeline in Chapter 3) to convert all the poses from different coordinated systems onto a universal, model-rooted homogeneous coordinate system, where all the translational and orientational effects are removed from the poses through appropriate transformations. I used three markers for normalization, namely, the markers at STRN, the left and right shoulders. These three *normalization* markers are crucial to the quality of the full pose configuration. I compress these three special markers separately from the rest of the markers, which I call the *non-normalization* markers.

Motion segmentation. I segment the normalized motion data sequence into subsequences of simple motions whose poses lie near a low-dimensional linear space.

Compression of segments by PCA. For each motion segment, I approximate the pose position vectors by their projections onto the space spanned by the leading principal components.

Key frame selection for spline interpolation. Given PCA projection data of each frame of a motion segment, I adaptively select and store only the key frames as control points, such that the spline interpolation of the rest of the frames yields an approximation error below a preset threshold.

Decompression. In decompression I use spline interpolation to recover the positions of the *normalization* markers, as well as the PCA projections of the non-key frames for the non-normalization markers. I then reconstruct the positions of the *non-normalization* markers in the normalized coordinate system.

Denormalization. Denormalization transforms all the normalized poses back to their original coordinate systems.

Since normalization and segmentation have been presented in Chapter 3, I will not discuss those components again in this chapter. Instead, I will describe in more detail the rest of the key components in the following sections.

7.2 Compression of Segments by PCA Approximation

PCA is a dimensionality reduction technique which retains those characteristics of a data set that contribute most to its variance. For a motion segment whose pose position vectors lie near a much lower-dimensional space, PCA is a very effective method of finding that low-dimensional space. I compute PCA for each motion segment and keep the leading k eigenvectors, such that the residual variance covered by the discarded eigenvectors is less than a preset threshold. The projections of the pose position vectors onto the k-dimensional principle component space are used as the approximations of the original poses. A motion segment is represented by a k-dimensional trajectory over time, with k being varied in different motion segments.

The reconstruction errors of all the mocap markers are not perceived on the same scale by my sensing system. Human vision tends to be more sensitive to errors on certain body parts than the others. For example, even very small errors at the foot marker positions could be detected and perceived as a major artifact called sliding feet or skating effect (Kovar et al., 2002b; Ikemoto et al., 2006). To address this issue, I compress the foot markers separately from the rest of the markers with a tighter error tolerance.

7.3 Key Frame Selection for Spline Interpolation

Human motion capture data demonstrate strong temporal coherence, as most time series do. We can reliably estimate a frame with its temporally adjacent neighbors. I opt to select and store only the PCA projections of the key frames from a motion segment to achieve further compression. In decompression I can apply the cubic spline interpolation to recover the non-key frames, using the saved key frames as control points for the spline function.

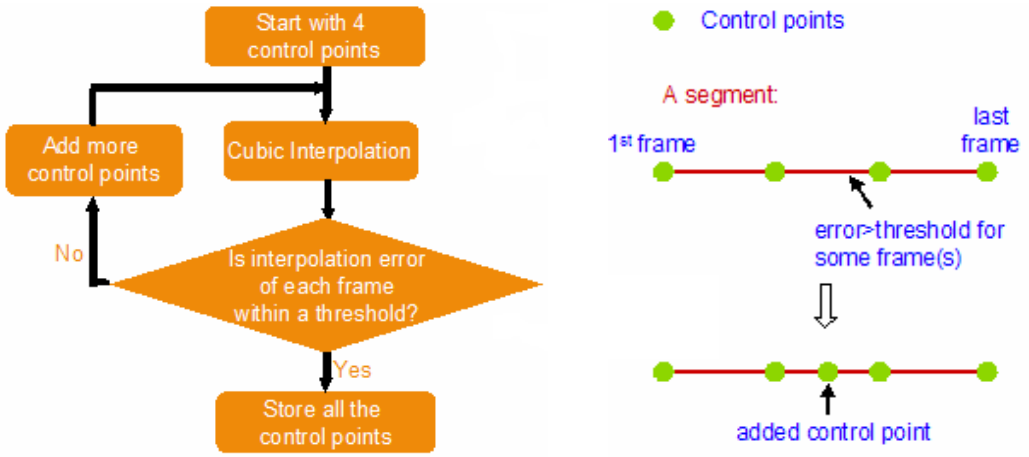


Figure 7.2: Adaptive key frame selection. (a) is a diagram of the key frame selection process; while (b) illustrates how the initial control points are selected and how the subsequence control points are added.

I adopt the cubic spline interpolation approach because of its computational simplicity, good approximation property and implicit smoothness (minimum curvature property). Selection of the key frames as control points is an adaptive process. As shown in Figure 7.2, I start by fitting each PCA-projected trajectory by a cubic spline function with four evenly distanced control points, two at both ends of the motion segment and the other two at the $\frac{1}{3}$ and $\frac{2}{3}$ temporal positions of the segment. I then interpolate all of the frames using the cubic spline functions and compute the interpolation errors. For each frame the approximation error of spline interpolation is calculated as the $L2$ norm of the difference between the k-dimensional interpolated vector and the original projection vector. If the approximation error of the interpolation exceeds a

preset threshold for any frame between the two existing control points, then the frame in the middle of those two control points is selected as a new control point and is added to the list of the existing control points. I continue adding control points and interpolating the frames until the interpolation errors of all the frames are within an error threshold.

As mentioned earlier the three *normalization* markers only go through one-level compression *via* the key frame selection. Since these three markers are crucial to the de-normalization of the rest of the markers during decompression and may ultimately affect the final decompression result, I apply a more stringent error tolerance in selecting the key frames.

For each compressed motion segment, I need to store the key frames for the three *normalization* markers: one mean vector, the k principal component vectors and the PCA projections of the key frames for the *non-normalization* markers.

7.4 Decompression and Denormalization

Decompression of motion data sequences is conducted separately for the *normalization* markers and *non-normalization* markers as follows.

Normalization markers: Since I only compress the motion data of the normalization markers in their original measurement space by selecting the key frames as control points for the cubic spline interpolation, I simply need to reconstruct the non-key frames with spline interpolation using those key frames as control points.

Non-normalization markers: Given the PCA projections of the key frames in each segment, I run cubic spline interpolation using those key frames as control points to estimate the PCA projections of the other frames. Then I reconstruct the marker positions in the normalized coordinate system using the principal component vectors associated with each of the motion segments. Finally, I transform the normalized mocap marker data back to the original marker coordinate system using the reconstructed positions of the normalization markers.

An inherent shortcoming with the local linear modeling approach is the temporal discontinuity at the transitions between PCA models, manifested as visible jerkiness in the reconstructed motion. For example, if I approximate two temporally adjacent motion segments using two different sets of principal component vectors, then it is likely to see jerkiness at the transition frames between these two segments.

In order to address this problem, instead of using only one PCA model to reconstruct

Sequence	Description	# of Frames	Size (KBytes)
1	Jumping Jacks, side twists, bending over, squats	4,592	4,413
2	Long break-dance sequence	4,499	4,324
3	Walk, squats, running, stretches, jumps, punches and drinking	10,590	10,177
4	Walk, squat, Stretching, kicking, and punching	9,206	8,847

Table 7.1: Used motion capture data sequences.

the poses at the transition of local PCA models, I use a mixture of PCA models associated with the particular pose and its neighboring poses. Let b_{ti} be a column vector containing the reconstructed $3D$ positions of markers at pose t based on PCA model i, I estimate the marker positions y_t, as

$$y_t = \sum_i w_i b_{ti}, \tag{7.1}$$

where $w_i = r_i/(h+1)$ is a weight for the i^{th} PCA model, and r_i is the number of frames corresponding to the i^{th} model among the frames $t - h/2$ to $t + h/2$. Basically, I put more weight on the model that is favored by more of the $h + 1$ poses. In my experiments, $h = 10 - 20$ works very well.

7.5 Experiments

I evaluated my compression method with Carnegie Mellon University's Graphics Lab motion capture database available at http://mocap.cs.cmu.edu. I used the 3D motion data on a set of 41 markers. The sampling rate is 120 frames per second. Table 7.1 shows the basic information of the motion sequences.

As discussed previously, the reconstruction errors may not be perceived on the same scale for different markers, so I chose different error tolerances according to the importance of markers. The PCA residual error threshold was set as 10 mm/marker for the six foot markers and 30 mm/marker for the other *non-normalization* markers. The spline interpolation error threshold was set as 10 mm/marker for the *non-normalization*

markers and 1 mm/marker for the *normalization* markers. Larger error tolerances typically result in more visible artifacts during the reconstruction. Extreme cases include apparent shifting of certain body parts, as well as jerkiness at the transitions. I assessed the performance of my compression method with regards to compression ratio and reconstruction quality. I visually evaluated the reconstruction results using my motion model viewer and quantitatively evaluated the reconstruction errors with two distortion measures. The first one is a distortion rate d similar to Keogh et al. (2004), which measures the quality of reconstruction for the whole motion sequence, and is defined as

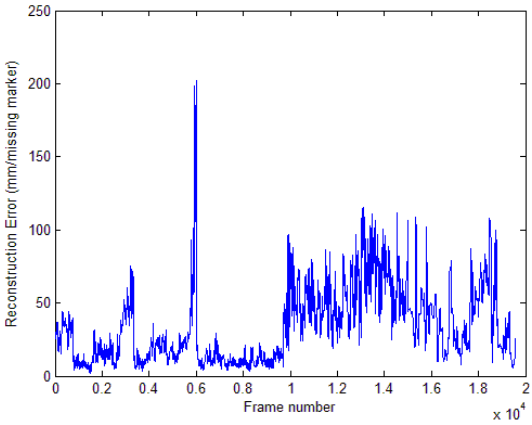
$$d = \frac{\|A - \tilde{A}\|}{\|A - E(A)\|}, \tag{7.2}$$

where A is a $3m \times N$ data matrix containing the original motion sequence collected from m markers over N frames. $\tilde{A}$ is the reconstructed result of the same motion sequence after decompression. $E(A)$ is an average matrix in which each column consists of the average marker positions for all the frames. The second distortion measure is the per-frame distortion, which is defined as the Root-Mean-Squared (RMS) error (mm/marker) in each frame.

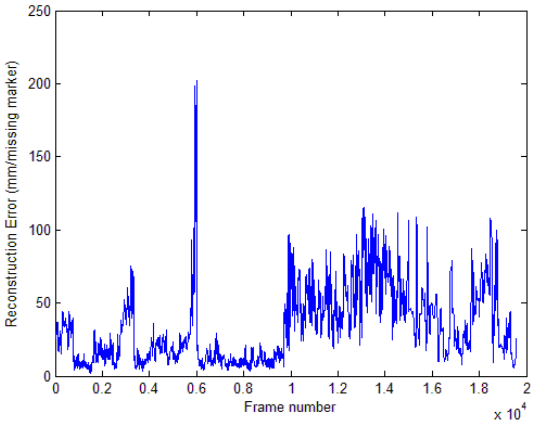
The compression and reconstruction results of four motion sequences were presented in Table 7.2. The *frame-by-frame* distortions were shown in Figure 7.3. These results show that my compression method achieved a high compression ratio with low distortion rate. The reconstructed motion sequences are of reasonably good quality with fairly smooth transitions from one PCA-modeled segment to another. Also by using a tighter PCA residual error tolerance for the foot markers instead of a uniform tolerance for all the markers, I greatly reduced the sliding-feet artifact and thus significantly improved the perceived visual quality of the reconstructed motion sequences. Some reconstructed sample frames are given in Figure 7.4.

As motion sequences became more complicated, more principle components and key frames were needed to ensure the reconstruction quality. For example, the long sequence of breakdancing was quite fast-paced and sophisticated, thus it required larger number of principle components and key frames than the other sequences to obtain comparable reconstruction quality. Therefore, the resulting compression rate is not as high as the other motion sequences.

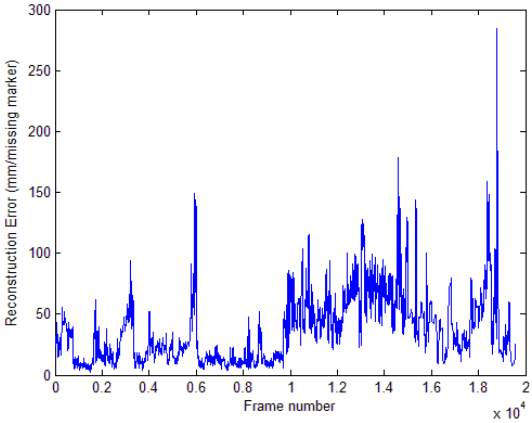
I compared my method to the global PCA method as well as the segment-based piecewise PCA method without spline interpolation (Table 7.3). It showed that the



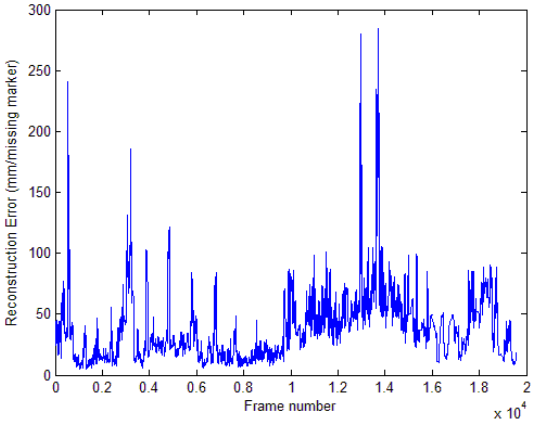
(a). Sequence 1



(b). Sequence 2



(c). Sequence 3



(d). Sequence 4

Figure 7.3: Reconstruction errors of decompression.

Sequence	1	2	3	4
# of segments	6	9	10	9
Average # of Principal components	4.6	10.9	3.1	4.2
# of control points	117	231	235	221
Compression ratio	1:55.2	1:18.4	1:61.7	1:56.0
Distortion rate d (%)	5.1	7.1	5.1	5.4
Compression time (ms/frame)	1.3	1.4	1.3	1.2
decompression time(ms/frame)	0.7	0.7	0.7	0.7

Table 7.2: Compression and decompression results

piecewise PCA compression method improved the compression performance for all the sequences, as compared to the global PCA compression method. However, the most significant improvement came from its use, combined with the spline interpolation, which efficiently incorporated the temporal coherence of the sequences. Note that 3 out of 41 markers were used as *normalization* markers in my experiment, and only 38 markers actually went through the segment-based PCA compression, so the compression effect of piecewise PCA may not be fully demonstrated. However, I expect significant advantages for segment-based PCA compression when the number of markers increases substantially.

I ran the experiments in Matlab V7 on a Dell Inspiron Laptop, with 1.4GHz CPU and 512M physical memory. Both of my compression and decompression algorithms are very fast and scale linearly with the number of frames in motion sequences. As Table 2 shows, the compression time is about 1.3 ms/frame for all four motion sequences, while the decompression time for each sequence is 0.7 ms/frame.

7.6 Conclusions

I presented a novel method to compress human motion data sequences based on the data-driven, piecewise linear modeling approach. I exploited both spatial and temporal coherences of motion data to achieve a considerably high compression rate with low degree of distortion. The experimental results showed that it is important to segment a long and complicated motion sequence into subsequences of short and simple motions, and to compress each of these subsequences separately. My segment-based approach requires fewer principal components to achieve the same error threshold than traditional global PCA compression, and this leads to a higher compression rate and

Sequence	Method	Compression Ratio	Distortion Rate (%)
1	Global PCA	1:5.3	4.2
	Piecewise PCA	1:8.0	4.1
	My method	1:55.2	5.1
2	Global PCA	1:3.7	5.8
	Piecewise PCA	1:5.0	5.4
	My method	1:18.4	7.1
3	Global PCA	1:6.4	4.9
	Piecewise PCA	1:9.4	4.3
	My method	1:61.7	5.1
4	Global PCA	1:5.3	4.5
	Piecewise PCA	1:8.4	4.5
	My method	1:56.0	5.4

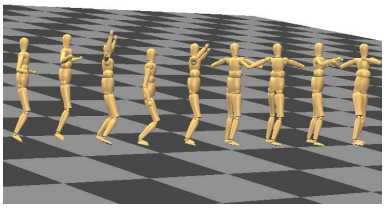
Table 7.3: Comparison of different compression methods.

better reconstruction results. PCA approximation is an effective way to characterize the correlations among mocap markers and significantly reduces the dimensionality of the marker data without the loss of important aspects of the human motions. Spline interpolation on top of PCA approximation further improves the compression rate by taking into account the correlations among temporally adjacent frames.

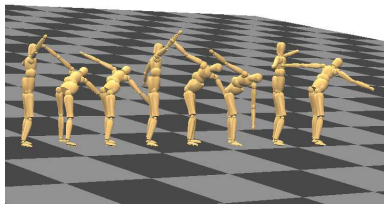
This compression method is a lossy compression scheme. If more accuracy is demanded, quantization can be used to store the differences between the reconstruction results and the original true values, so that the actual errors can be further reduced to the quantization errors only. However, doing so means we have to allocate extra space to store these quantized errors, and the trade-off would be the reduced compression ratio. The experimental results showed that few visual artifacts appear when the information reflected by the residual errors is discarded, so the use of quantization may not be necessary.

Motion capture data are often represented as joint angles in animation research. Since joint angles are a hierarchical representation, it is difficult to achieve high quality compression by directly compressing the joint angle data due to the accumulation of errors along the chain of the joints. As a solution we can first convert joint angles into joint positions and then compress these joint positions instead. In a concurrent work by Arikan (Arikan, 2006), the conversion to positional data of so-called *virtual markers* was also involved in the compression of joint angle data. It is worth examining how effective my method is, as compared to Arikan's approach in handling the joint angle data.

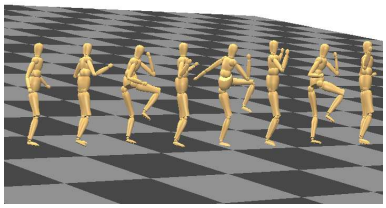
While I demonstrated the utility of this compression method with full-body human motion capture data, in my future work I would like to evaluate how well my method performs in compressing other types of human motion data, such as facial expressions. In addition, the application of this compression method to other formats of data such as animation meshes, sequences of point clouds and range scan data also merits further study.



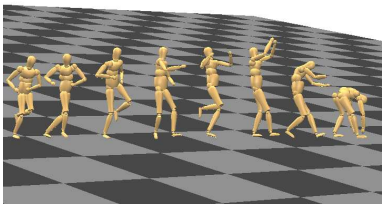
Sequence 1:
(a) jumping & twists

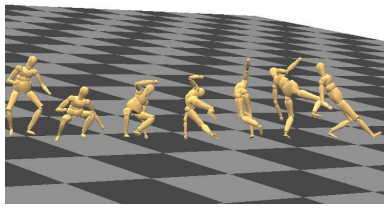


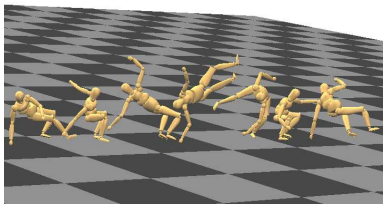
(b) bending over



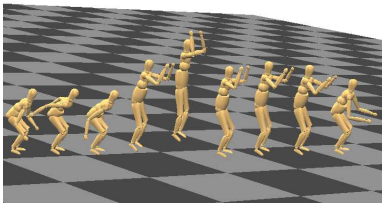
(c) stretching



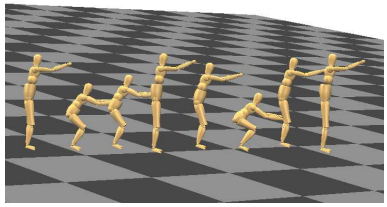




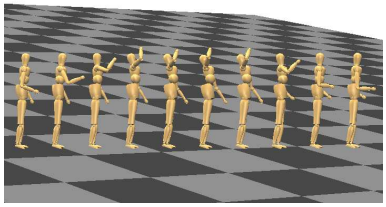
Sequence 2: breakdance



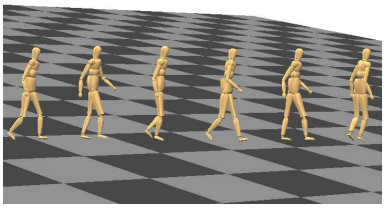
Sequence 3: (a) jumping



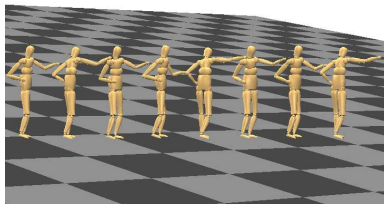
(b) squats



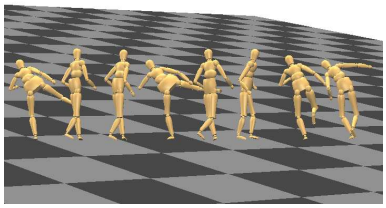
(c) drinking



Sequence 4: (a) walking



(b) punching



(c) kicking

Figure 7.4: Sample frames from the decompressed motion sequences.

Chapter 8

Conclusions and Future Work

Human motions are a special type of high-dimensional time series. Traditional modeling approaches typically rely on prior assumptions, domain knowledge and physical simulations to construct often over-simplified models. This strategy was appropriate during the early days when data acquisition techniques had not matured enough, and so motion capture data were difficult and expensive to acquire. However, nowadays motion capture data have been widely used in more and more applications. Since many applications demand highly realistic motions with so many details and subtleties, it becomes much harder for these traditional models to satisfy the needs of those applications. On the other hand, data-driven modeling approaches, by adopting a *Let the data tell the story* strategy, construct appropriate sample-based models by analyzing the sampled data, with only minimal assumptions and domain knowledge. They applied data mining and modeling techniques to construct models that are accurate enough to capture the essence and subtleties of a data set with the least human interventions and bias. Due to minimal constraints imposed from the assumptions, data-driven models are often flexible enough and adapt very well to various types of data, as well as adverse and noisy environments.

During the past decade, we have experienced great advances in data acquisition, analysis, modeling and computational technology that have made many elegant but computationally expensive and time consuming algorithms run within a reasonable time frame. These advances have spawned more and more methods that are based on the data-driven modeling approach. The segmented-based, piecewise linear human modeling approach presented in this dissertation makes a conscious effort to systematically apply the principles of data-driven approaches in modeling human motions.

8.1 Synopsis

In this dissertation I have presented the framework of a data-driven, segment-based, piecewise linear modeling approach to modeling human motions. In particular, I discussed the segmentation of motion sequences into subsequences of distinct behaviors and low dimensionality. I also described the characterization of motion segments by their statistical distributions. I presented a divisive clustering method to group similar motions together. I then explained how to construct a local linear model out of the poses from each group of similar motions. Finally, I discussed the classification strategy to identify the most appropriate local linear model, given a frame or a segment of motions. In order to demonstrate this modeling approach's usefulness in modeling human motions, I applied the framework of this approach to four driving problems carefully chosen from a broad range of human motion related applications, namely, human motion estimation from a reduced marker set; missing marker estimation from available markers; motion retrieval from a large, heterogenous database; and motion sequence compression.

In the remainder of this chapter, I will discuss the strengths and weaknesses of this modeling approach, especially when being applied to the aforementioned four driving problems. I will then discuss some interesting applications as well as open problems that may merit further investigations in the future work.

8.2 Strengths and Weaknesses

This modeling approach requires minimal human intervention during the modeling process. Traditional human modeling approaches rely heavily on human knowledge of the domain of the data. However, this knowledge may not always be available. It is more often that a pre-assumed model becomes very complicated and inevitably big but still seems inadequate in describing all the subtleties exhibited by the data. In contrast, my data-driven approach doesn't need to assume any human skeleton model using prior assumptions and domain knowledge. This property makes this approach very flexible in modeling human motion data under various environments and conditions where traditional modeling approaches may not be very effective nor feasible. My approach applies a very simple and natural concept that assumes that human motions lie in a piecewise linear space. Poses of each group of similar motion segments are near a very low-dimensional space and can be sufficiently modeled with a lot fewer parameters.

These local linear models can collectively provide a compact but sufficiently accurate and robust model for a large and heterogenous motion dataset. By using an appropriate modeling resolution based on motion segments, we can capture the essence of human motions in a relatively simple and compact model while ignoring insignificant details and noise exhibited in human motion data. Since domain knowledge is not explicitly involved in the modeling process, my approach, without any significant modification, can be potentially applied to a dataset other than human motions. For example, as I will show in the next section of future work, we can apply the principle of this modeling approach to model the molecular dynamics of protein, despite the fact that this type of data set bears no resemblance to human motions.

Everything has two sides. There is no exception here. A data-driven, sample-based modeling approach relies heavily on the training data, as does my modeling approach. An inherent weakness of this approach is the limitation to extrapolation. In other words it can only reproduce motions similar to what have been seen in the training set and would fail if used to hallucinate new motions that bear no similarity to the motions in the training set. For example, a model trained by only walking and running motions would not be able to produce novel motions such as boxing or dancing, etc. In order to reliably recover various motions under different conditions, this approach needs a large heterogenous motion database of a wide range of human motions.

Another common drawback of a data-driven modeling approach is the *overfitting* problem. Unlike traditional modeling approaches where we may well know the underlying model or how many sub-models exist, a data-driven modeling approach like ours, usually seeks a model of the data which will give us, on average, the best possible predictions for novel data. We are more likely to have overfitting problems if we have relatively few data points. In the opposite case, where we have essentially an infinite number of data points, we are not usually in danger of overfitting the data, as the noise associated with any single data point plays a small role in my overall fit. However, overfitting may still occur in a large but imbalanced motion database, where we may have more than enough samples for some motions but limited samples for certain other motions. In order to construct a robust and effective model using this approach, it is important to be consciously aware of this overfitting issue and to provide large, heterogenous and balanced motion data as a training set, where every distinct motion is well represented with sufficient samples. In this dissertation study, I also implemented some mechanisms proven to be effective in the modeling process to mitigate the overfitting problem. For example, one reason that leads me to ap-

ply the Random Forest classification rather than other classification techniques is that Random Forest has an inherent resistance to overfitting with the prediction-by-voting strategy and cross-validation mechanism (Breiman, 2001). The experimental results showed that this classification technique effectively addressed the overfitting issue in the classification phase.

Throughout this dissertation I model human motions in the mocap marker space instead of in the derived joint angle space with an assumed human skeleton. Modeling in the marker space has advantages over modeling in the joint angle space. Significant aspects of human motions are not captured *via* joint angles, in particular, shape deformations due to the motions of muscles and other fleshy areas. Generally, evidence of this sort of motion is available from the measured data, i.e. raw marker data. However, such subtle information may have been smoothed out during the conversion onto the joint angle space. Modeling human motions in the marker space pushes motion data processing a step closer to raw data measurements. Doing so also streamlines the modeling process by eliminating skeleton estimation and skeleton calibration, which are otherwise needed when modeling human motion data in the joint angle space. The $3D$ marker position data are also easy to manipulate because they are parameterized as spatial coordinates rather than angles, thus allowing for a simple Euclidean metric, which otherwise is not applicable in joint angle space. Modeling in marker space also appears to be a more natural choice in working on some applications and often has a more straightforward modeling process, which often produces simpler but more effective human motion models. For example, in the first driving problem, i.e. *human motion estimation from a reduced marker set*, I estimate the remaining marker positions from a small set of principal markers. A joint angle based approach would have to construct a model in the joint angle space and estimate from the principal marker positions all the joint angle values, which in turn produces the full configuration of a pose. It would make more sense and seems to be more straightforward to directly model in the marker space and infer from the principal marker positions the positions of the remaining markers. Similar arguments could be made in the other two driving problems, namely, *missing marker estimation* as well as *motion sequence compression*. On the other hand, there may be a normalization issue with direct use of marker data due to size differences among human subjects. Nevertheless, my experiments showed that the performance of the proposed method was not sensitive to the normal variations in subjects' sizes, as long as the human subject whose motions to be estimated falls in a normal range of the representative human subjects in the motion database. In the

experiments equivalent motions from different subjects tend to lie in the same local linear space, so the corresponding mapping function is actually computed based on data from different subjects. Calibration of subjects of different sizes does not appear to be essential with this marker-based approach. However, more experiments are needed in this regard.

The framework that I presented in this dissertation is a piecewise linear approach. As compared to other traditional nonlinear modeling methods, such as neural network, Hidden Markov Models, Gaussian process latent variable models (GPLVM), etc., piecewise linear modeling is simple and straightforward. Furthermore, human motions do exhibit local linearity and piecewise linear models would fit well with such type of data. This modeling approach models human motions as a collection of local linear models. Each local linear model is very compact but effective in capturing the subtleties of human motions. PCA-based low-dimensional local linear mapping function is computationally more efficient than the optimization-based searching methods that may be sometimes trapped at the local minimums in the searching space. As demonstrated in the experimental results in the four driving problems, piecewise linear modeling achieved better modeling performance in terms of more accurate and plausible motion estimation, as well as higher motion compression ratio, than the methods adopting the global modeling approaches. On the other hand, piecewise linear modeling has its inherent limitations as well. Temporal discontinuity at the transitions between linear models often causes visible artifacts in the motions reconstructed from a piecewise linear model. I have discussed in the previous chapters that the cause for this temporal discontinuity is due to the change of bias direction of the reconstruction errors. Although this inherent limitation can't be completely eliminated, I did provide a solution by using a mixture of linear models to mitigate the artifacts caused by the temporal discontinuity, so that they are not visible to the human perception.

My approach is a segment-based modeling approach. Data-driven approaches, even piecewise linear modeling, have been explored before in some fields and are thus not new ideas. However, it is important to construct a piecewise linear model based on motion segments instead of individual poses. Human motion data are a special high-dimensional time series. Human perception is very sensitive to temporal discontinuity. One side effect of piecewise linear modeling, with individual poses as the modeling primitives, is that it tends to group similar poses from different motion sequences of different behaviors into the same local linear model while partitioning temporally adjacent frames from the same motion sequence into different local linear models. This

may lead to many unnecessary model transitions when reconstructing a simple and single-behavior motion. Too many unnecessary transitions between linear models may cause temporal discontinuity manifested as visible jerkiness, an artifact in the reconstructed human motions. In data modeling it is important to choose an appropriate modeling resolution. By modeling human motions at the resolution of motion segments, we guarantee poses of the same motion segment to be in the same local linear model. This modeling strategy closely resembles in spirit how human perception works and can drastically reduce unnecessary transitions between models. Thus it may improve the overall quality of the estimated motions.

8.3 Future Work

I have demonstrated my approach's usefulness in the four challenging driving problems chosen from a wide range of human motion applications. There are many other interesting problems that may be well worth the investigation under this modeling framework. In the following subsections, I will discuss a few interesting problems that may benefit from applying this human motion modeling approach.

8.3.1 Relieving ambiguity in marker labeling

Marker labeling is an important step during a motion capture process. Labeling is basically the assigning of specific names to specific markers. Once we have more than one entity that we're trying to track, we need to know which is which by naming them, and this is where labeling comes in. It is more than often that during a marker labeling process, we may encounter a so called *correspondence problem*, because when they occur, we do not know which blob in a camera image *corresponds* to which marker since the markers are just dots on an image. Occlusion and ghosting are two primary sources of marker ambiguity. Occlusion occurs when a performer turns around, and two or more markers are eclipsed (or occluded) by their bodies. When they re-appear it is no longer certain which is which. Ghosting occurs when two markers are seen by only two cameras, and the cameras and the markers lie in the same plane. In this case there is an ambiguity about exactly where the markers are. Again, while several means are available, it may not be possible to automatically re-establish the identity of the markers. Instead, human interventions are often needed for the marker re-identification task.

Re-establishing marker identity (or *correspondence*) across the gaps of motion data requires hours of manual labor doing what is referred to as *cleanup*. Also, while adding cameras is desirable for accuracy and enlargement of the captured volume, it tends to increase the opportunities for ghosting or occlusion, and thus makes re-identifying the markers to each camera harder. This optical data cleanup problem is the source of the early (and correct at the time) belief that it took almost as long to capture and clean up motion data as it would to hand-animate it.

We can greatly ease the burden of the marker re-identification task by applying the same piecewise linear modeling approach to this problem as applied to the missing marker problem. Once a marker is classified as occluded or ambiguous, it is declared as *missing*, and its position would be recovered from the other available marker positions. Once we recover the missing marker positions, we compare the estimated marker positions with the ambiguous blobs and assign them to the marker whose position has the closest distance to the estimated marker positions. By doing so we could dramatically speed up the marker labeling process as compared to the existing marker labeling methods.

8.3.2 Markerless motion capture

The past two decades have seen great progress in marker-based motion capture methods, resulting in robust systems that can produce accurate results. On the other hand, markerless motion capture methods have only seen a decade's worth of research. Even though promising results have been reported, these systems have not provided the same robustness or precision as their marker-based competitors. While marker-based methods might be used to give satisfactory and accurate results for the purposes of some of the applications outlined above, the process of wearing special clothing and/or markers is generally unpleasant and time consuming, thus favoring the use of markerless mocap systems. Additionally, some applications demand that there is no intrusion to the subject's body whatsoever. It is therefore clear why development of robust markerless tracking algorithms is desirable.

In recent years there have been studies on markerless motion capture in the computer vision community. In markerless motion capture, human images are acquired through some passive sensing mechanisms and then reconciled into kinematic motions. General approaches to markerless motion capture are to first assume a human skeleton as a rooted hierarchy of bones and joints, and then to use silhouettes or other image

information to estimate or infer the joint positions on the human skeleton. However, it is not always easy to make a sufficiently accurate estimation in a timely manner on all the joint positions from the available image cues at anytime. On the other hand, it is conceivable that at different positions, only certain joint positions can be more accurately estimated than the other joint positions.

A strategy to tackle this problem entails estimating all the joint positions solely from the silhouettes of captured human figures. We only estimate a subset of the joints that can be accurately estimated from the silhouettes at a current particular position. At the next stage, we infer from the newly estimated joint positions the rest joint positions, the same way as we approach the missing marker problem. By this two-stage approach, we can recover all the joint positions more reliably than the existing markerless mocap methods with much less time and computing power. However, it is a nontrivial problem to adopt the piecewise linear modeling approach in estimating the joint positions from the silhouette. Normalization, for example, may be an issue. Nevertheless, there is great promise that my approach can be applied to this problem, in principle, to quickly and accurately recover the human motions in markerless motion capture system.

8.3.3 Discrimination of abnormal motions from normal motions

In some applications such as evaluating the progress of physical therapy and patient monitoring, it is important to be able to discriminate abnormal motions from normal motions. Healthy people typically have similar normal poses as well as normal motions. In normal motions, although they often fluctuate due to different body shapes, styles and other noises, major coordinations (couplings) among body parts are preserved, i.e. invariant. In abnormal motions, in contrast, certain coupling invariability may have been violated while new couplings may have formed.

A typical human motion sequence consists of three components: generic components, style component and noise components. The generic component captures the major coupling invariability, while the style and noise components only count for a small variation from the generic poses. Motion sequences with the same generic motion tend to be similar and also tend to have similar motion transition trajectories. On the other hand, abnormal motions differ dramatically from any of the normal motions since they are considered to be derived from different generic motions.

Motion discrimination is inherently a classification problem. In a large motion

dataset, normal and abnormal motions often tangle with each other in a high-dimensional space. It is impossible to find a global classifier to identify normal and abnormal motions. The piecewise linear modeling approach adopts a divide-and-conquer strategy to partition motion sequences into smaller and simpler motion segments that can be described with local linear models. Each of the resulting motion segments can then be characterized by the statistical properties from which a signature may be derived. These signatures can be later used to train a classifier to discriminate between normal and abnormal motions for a given pose or motion clip.

A path can also be drawn from an abnormal motion to its intended normal counterpart motion with sufficient data support. This path would be useful in monitoring and quantitatively evaluating patients' rehabilitation from diseases that have severely damaged their motor systems and consequently altered their motion patterns.

8.3.4 Automatic motion pattern mining and annotation

Human motions are highly coordinated among different body parts. There exist consistent patterns in many different motions. These patterns are invariant among similar motions under certain resolutions. Annotation is a technique of giving a descriptive summary on the identifiable motions with certain patterns. Annotation can be applied at different modeling resolutions, for example, at a lower (finer) level where simple motion strokes such as lifting a foot, waving a hand, etc., are the annotation primitives. On the other hand, at a higher level of resolution, more abstract and behavioral motion segments, such as walking, running, jumping, sitting, etc., compose the vocabulary of the annotations. Annotations of higher abstraction-level motions can always be rephrased with the vocabularies in the lower-level motion annotations. For example, walking, can always be equivalently described as *alternate swinging between hands and feet*.

Currently, most annotations are most done manually. In particular, human expertise is heavily relied upon to find and define meaningful patterns as the building blocks at various modeling resolutions. As greater amounts of motion data become available, we need to develop an effective and efficient strategy to find the patterns and annotate various motions with only minimal human intervention.

My piecewise linear modeling approach can be applied to achieve this goal. At a higher and more abstract level of resolution, human motions are to be categorized as a collection of short, simple, and most of the time, single-behavior motion clips,

i.e. segments. We have already been able to find such simple motions by segmenting long and complicated motion sequences into distinct behaviors and by grouping similar motions together by the similarities in their statistical distributions. Signatures can also be derived from the statistical properties of the motion segments. Standard data mining methods can then be applied to search for meaningful patterns.

In order to find primitives for low-level annotations, each behavioral motion segment is further divided into simple-strokes, in which the geometric relationships among certain key body parts remain invariant. We can first select a subset of key body parts with a strategy similar to the principal marker selection presented in Chapter 4, and then encode each frame of the original single-behavior motion segments with the pairwise geometric relationships among the key body parts. With this strategy we can annotate motion sequences at multiple modeling resolutions automatically or with only minimal human intervention.

8.3.5 Modeling protein dynamics

Protein structures are not static. Instead, proteins are dynamic molecules that often undergo conformational changes while performing their specific functions, such as an enzyme reaction or ligand binding. Many of the bonds in a protein can rotate and flex, and entire structural segments of the protein can move on a variety of timescales. The types and timescales of motions that the protein experiences can play a significant role in the way that the protein functions. The dynamic properties intrinsic to a protein structure may provide information on the location and the energetics of the conformational change process, and are thus the focus of many biophysical studies. Protein dynamics are essential for specific biological functions (Huitema and van Liere, 2000).

The piecewise modeling approach has promises to be useful in modeling protein dynamics, which can be treated as a special high-dimensional time series. For example, the principal marker selection method can be applied to capture the most relevant aspects that influence the binding process and determine the affinity with which a potential drug candidate binds to its protein target. As compared to the global modeling approach used in the existing protein dynamics modeling methods, the local linear modeling strategy may produce a more compact but more accurate model with a lot fewer parameters. It can be used for fast protein molecular modeling that may capture the essence of a variety of protein dynamics in a timely manner with a high accuracy

but relatively low cost.

8.3.6 Summary

I have briefly discussed a few interesting problems that could potentially benefit from my data-driven, piecewise linear modeling approach and outlined a strategy on each problem discussed. I believe that this approach provides a viable, and perhaps better alternative to the traditional modeling approaches under certain circumstances. This data-driven, segment-based, piecewise linear approach would find great utilities on many more applications within and beyond human motion modeling.

Appendix A

Source Code

I present here the core source code developed during the course in this dissertation. I will first present the generic modules for constructing segment-based, local linear models. I then present the additional source code used in addressing each of the four aforementioned driving problems. All the source code was written in Matlab version 7.

A.1 Segment-based, Piecewise Linear Modeling

A.1.1 Normalization

```
0001 %##########################################################################
0002 %The following function is used to normalize the mocap frames. M stores   #
0003 %each frame's full marker positions. FrM stores only the positions of the #
0004 %three markers used for the normalization.                                 #
0005 %##########################################################################
0006 function N_M = Normalize(FrM, M)
0007
0008 nFrame = size(M,1);
0009 nPnt = size(M,2)/3;
0010 N_M = zeros(size(M));
0011 for i = 1:nFrame
0012     Origin = FrM(i,[1:3]);
0013     Lsho = FrM(i,[4:6]);
0014     Rsho = FrM(i,[7:9]);
0015     Xaxis = (Rsho - Lsho) / norm(Rsho - Lsho);
0016     Zaxis = [0,0,1];
0017     Xaxis = Xaxis - dot(Xaxis, Zaxis)*Zaxis;
0018     Xaxis = Xaxis / norm(Xaxis);
0019     Yaxis = cross(Zaxis, Xaxis);
0020     frameTran21 = [Xaxis; Yaxis; Zaxis];
0021     frameTran12 = inv(frameTran21);
0022     for j = 1:nPnt
0023         N_M(i,j*3-2:j*3) = (M(i,j*3-2:j*3) - Origin) * frameTran12;
0024     end
0025 end
```

A.1.2 Motion segmentation and characterization

```
0001 %#########################################################################
0002 %The following function is the main function of the PPCA motion sequence  #
0003 %segmentation algorithm. It iteratively calls its sub function to segment #
0004 %a motion sequence into subsequences.                                      #
0005 %#########################################################################
0006 function ppcaSegMain(dFolder, sFolder, per, block, delta, trange)
0007
0008 load NormIdx;
0009 fname = [dFolder, '/M*N*.mat'];
0010 allFiles = dir(fname);
0011 cnum = 0;
0012 tnum = 0;
0013 clus = [];
0014 segs = [];
0015 segNames = [];
0016 index = 0;
0017 flist = [];
0018 for i=1:size(allFiles, 1)
0019     matName = [dFolder, '/', allFiles(i).name];
0020     newName = allFiles(i).name(1:end-4);
0021     flist(i).name = newName;
0022     tmp = load(matName);
0023     MN = tmp.M_N;
0024     M = MN(:, NormIdx);
0025     sizem = size(M,1);
0026     current = 1;
0027     offset = 0;
0028     indx = 0;
0029     segIdx = [];
0030     while (1)
0031         sIdx = ppcaSegSub(M(current:end,:), per, block, delta, trange);
0032         indx = indx + 1;
0033         segIdx(indx,:) = [offset+1, offset+sIdx];
0034         offset = offset+sIdx;
0035         if (offset < sizem)
0036             current = offset+1;
0037             continue;
0038         else
0039             break;
0040         end;
0041     end
0042     segs(i).segIdx = segIdx;
0043     numOfSegs = size(segIdx,1);
0044     segs(i).segs = [index+1:index+numOfSegs];
0045     for j=1:numOfSegs
0046         if (j < 10)
0047             fname = [sFolder, '/', newName, '0', int2str(j)];
0048             index = index+1;
0049             segNames(index).name = [newName, '0', int2str(j)];
0050         else
0051             fname = [sFolder, '/', newName, int2str(j)];
0052             index = index+1;
0053             segNames(index).name = [newName, int2str(j)];
0054         end;
0055         M_N = MN(segIdx(j,1):segIdx(j,2), :);
```

```
0056         save(fname, 'M_N');
0057         numOfFs = size(M_N, 1);
0058         cnum = cnum + 1;
0059         st = tnum + 1;
0060         ed = tnum + numOfFs;
0061         clus(cnum,:) = [st, ed];
0062         tnum = ed;
0063     end
0064 end
0065 fname = [sFolder, '/clus'];
0066 save(fname, 'clus');
0067 fname = [sFolder, '/segs'];
0068 save(fname, 'segs');
0069 fname = [sFolder, '/segNames'];
0070 save(fname, 'segNames');
0071 return;

0001 %#############################################################################
0002 %The following function is the child function of the PPCA motion          #
0003 %segmentation algorithm.                                                   #
0004 %#############################################################################
0005 function len = ppcaSegSub(M, per, block, delta, T)
0006
0007 R =500;
0008 [sizem, numOfDims] = size(M);
0009 if (sizem <= block + T)
0010     len = sizem;
0011     return;
0012 end;
0013 index = 0;
0014 max1 = -1000000000;
0015 max2 = max1;
0016 min1 = 1000000000;
0017 status = 1;
0018 for K=block:delta:sizem-T
0019     Nmean = mean(M(1:K,:),1);
0020     N = M(1:K,:) - repmat(Nmean, K, 1);
0021     Cinv = ppcaMod(N, per);
0022     sumh = 0;
0023     for j=1:T
0024         centered = (M(K+j,:)-Nmean);
0025         sumh = sumh + centered * Cinv * centered';
0026     end
0027     index = index + 1;
0028     sumH(index) = sumh / T;
0029     continue;
0030     if (status == 1)
0031         if (sumH(index) > max1)
0032             max1 = sumH(index);
0033             status = 2;
0034         end;
0035     elseif (status == 2)
0036         if (sumH(index) < max1 && sumH(index) < min1)
0037             min1 = sumH(index);
0038             status = 3;
0039         else
0040             if (sumH(index) > max1)
```

```
0041                max1 = sumH(index);
0042                status = 2;
0043            end;
0044        end;
0045    elseif (status == 3)
0046        if (sumH(index) < min1)
0047            min1 = sumH(index);
0048            status = 3;
0049        else
0050            if (sumH(index) > max2)
0051                max2 = sumH(index);
0052                status = 4;
0053            end;
0054        end;
0055    else
0056        if (sumH(index) < max2)
0057            if (max2 - min1 >= R)
0058                len = K;
0059                xaxis = [1:index] * delta;
0060                plot(xaxis, sumH);
0061                return;
0062            else
0063                max2 = -1000000000;
0064                if (sumH(index) < min1)
0065                    min1 = sumH(index);
0066                end;
0067                status = 3;
0068            end;
0069        else
0070            max2 = sumH(index);
0071        end;
0072    end;
0073 end
0074 len = sizem;
0075 return;
```

```
0001 %#############################################################################
0002 %The following function is used to derive a feature vector from each      #
0003 %motion segment by making a weighted concatenation of the elements of     #
0004 %the mean vector and the upper triangle of the covariance matrix.          #
0005 %#############################################################################
0006 function compFeatures(dFolder, fvFolder, w)
0007
0008 load NormIdx;
0009 fname = [dFolder, '/M*N*.mat'];
0010 allFiles = dir(fname);
0011 fvec = [];
0012 for i=1:size(allFiles, 1)
0013     %read the ith file
0014     matName = [dFolder, '/', allFiles(i).name];
0015     newName = allFiles(i).name(1:end-4);
0016     tmp = load(matName);
0017     MN = tmp.M_N;
0018     M = MN(:, NormIdx);
0019     fmean = w * mean(M,1);
0020     cv = cov(M);
0021     rvec = diag(cv)';
```

```
0022     svec = [];
0023     for j=1:size(rvec,2)
0024         svec(j) = sqrt(rvec(j));
0025     end
0026     fvec(i,:) = [fmean, svec];
0027 end
0028 fname = [fvFolder, '/fvec'];
0029 save(fname, 'fvec');
0030 return;
```

```
0001 %#########################################################################
0002 %The following function is used for dimensionality reduction of the       #
0003 %segment feature vectors. It computes PCA out of all the feature vectors  #
0004 %and keeps the leading principal components. It then projects each feature#
0005 %vector to this principal component space and uses these projections to   #
0006 %approximate the feature vectors.                                         #
0007 %#########################################################################
0008 function dimRedux(fvFolder, pcper, pcsize)
0009
0010 fname = [fvFolder, '/feavec'];
0011 tmp = load(fname);
0012 feavec = tmp.fvec;
0013 numOfFs = size(feavec, 1);
0014 fmean = mean(feavec, 1);
0015 [evec,eval] = empca(feavec', pcsize);
0016 cenvec = feavec - repmat(fmean, numOfFs, 1);
0017 clear feavec;
0018 tmp = 0;
0019 for i=1:numOfFs
0020     tmp = tmp + dot(cenvec(i,:), cenvec(i,:));
0021 end
0022 totalvar = tmp / (numOfFs - 1);
0023
0024 for i=pcsize:-1:1
0025     if (sum(eval(1:i)) / totalvar > pcper)
0026         continue;
0027     else
0028         dim = i+1;
0029         vportion = sum(eval(1:dim)) / totalvar
0030         break;
0031     end;
0032 end
0033 fvec = cenvec * evec(:,1:dim);
0034 fname = [fvFolder, '/fvec'];
0035 save(fname, 'fvec');
0036 fname = [fvFolder, '/fmean'];
0037 save(fname, 'fmean');
0038 fname = [fvFolder, '/evec'];
0039 save(fname, 'evec');
0040 fname = [fvFolder, '/eval'];
0041 save(fname, 'eval');
0042 fname = [fvFolder, '/vportion'];
0043 save(fname, 'vportion');
0044 return;
```

A.1.3 Construction of model hierarchy

```
0001 %##########################################################################
0002 %The following function is the main function of the divisive clustering   #
0003 %algorithm. It recurively calls its sub function to construct a model     #
0004 %hierarchy using the dimension-reduced feature vectors as the modeling    #
0005 %primitives.                                                              #
0006 %##########################################################################
0007 function clusteringMain(fvFolder, mFolder, vtol)
0008
0009 global numOfLeaves;
0010 global leaf;
0011 leaf = [];
0012 numOfLeaves = 0;
0013 fname = [fvFolder, '/fvec'];
0014 tmp = load(fname);
0015 M = tmp.fvec;
0016 numOfFs = size(M, 1);
0017 mlist = [1:numOfFs];
0018 root = fcluSN(M, mlist, '0',  vtol);
0019 fname = [mFolder, '/clusterTree'];
0020 save(fname, 'root');
0021 fname = [mFolder, '/numOfLeaves'];
0022 save(fname, 'numOfLeaves');
0023 fname = [mFolder, '/leaf'];
0024 save(fname, 'leaf');
0025 return;
```

```
0001 %##########################################################################
0002 %The following function is the child function of the divisive clustering  #
0003 %algorithm.                                                                #
0004 %##########################################################################
0005 function node = clusteringSub(localM, mlist, code, vtol)
0006
0007 global numOfLeaves;
0008 global leaf;
0009 node.code = code;
0010 numOfFs = size(localM, 1);
0011 localMean = mean(localM, 1);
0012 node.Mean = localMean;
0013 maxVar = -1000000;
0014 for i=1:numOfFs
0015     cvar = norm(localM(i,:) - localMean);
0016     if (cvar > maxVar)
0017         maxVar = cvar;
0018     else
0019         continue;
0020     end;
0021 end
0022 node.radius = maxVar;
0023 node.mlist = mlist;
0024 if (maxVar <= vtol) % leaf node
0025     numOfLeaves = numOfLeaves + 1;
0026     node.class = numOfLeaves;
0027     node.leftFlag = -1;
0028     node.left = [];
0029     node.rightFlag = -1;
```

```
0030     node.right = [];
0031     leaf(numOfLeaves).mlist = mlist;
0032 else
0033     mat = [localM(1, :); localM(numOfFs,:)];
0034     [clusterIdx, C] = kmeans(localM, 2, 'start', mat, 'maxiter', 200);
0035     node.KmeanTime = toc;
0036     lIndex = 0; % the number of frames on the left branch
0037     rIndex = 0; % the number of frames on the right branch
0038     for i=1:numOfFs
0039         if (clusterIdx(i) == 1)
0040             lIndex = lIndex + 1;
0041             lMlist(lIndex) = i;
0042         else
0043             rIndex = rIndex +1;
0044             rMlist(rIndex) = i;
0045         end;
0046     end
0047     if (lIndex > 0)
0048         node.leftFlag = 1;
0049         node.left = fcluSN(localM(lMlist, :), mlist(lMlist), ...
0050             [node.code, '0'], vtol);
0051     else
0052         node.leftFlag = -1;
0053     end;
0054     if (rIndex > 0)
0055         node.rightFlag = 1;
0056         node.right = fcluSN(localM(rMlist, :), mlist(rMlist), ...
0057             [node.code, '1'], vtol);
0058     else
0059         node.rightFlag = -1;
0060     end;
0061 end;
0062 return;
```

A.1.4 Local linear modeling

```
0001 %#########################################################################
0002 %The following function is used to construct a local linear model for the #
0003 %segments grouped together. It retrieves the mean and the principal       #
0004 %components of the poses in the same cluster. It then builds mapping      #
0005 %functions from a space spanned by the available markers to the principal #
0006 %component space as well as to the full marker space.                     #
0007 %#########################################################################
0008 function localLinearModel(dFolder, mFolder, projDim)
0009
0010 load NormIdx;
0011 fname = [mFolder, '/leaf'];
0012 tmp = load(fname);
0013 leaf = tmp.leaf;
0014 numOfLeaves = size(leaf, 2);
0015 fname = [dFolder, '/M*N*.mat'];
0016 allFiles = dir(fname);
0017 index = 0;
0018 for i=1:numOfLeaves
0019     S = [];
0020     numOfSegs = size(leaf(i).mlist, 2);
```

```
0021     mlist = leaf(i).mlist;
0022     for j=1:numOfSegs
0023         fname = [dFolder, '/', allFiles(mlist(j)).name];
0024         tmp = load(fname);
0025         S = [S; tmp.M_N(:,NormIdx)];
0026     end
0027     sizes = size(S, 1);
0028     smean = mean(S, 1);
0029     NS = S - repmat(smean, sizes, 1);
0030     Cov = cov(NS);
0031     [pc,latent,explained] = pcacov(Cov);
0032     proj = NS * pc(:,1:projDim);
0033     mp(i).pcs = pc(:,1:projDim)';
0034     mp(i).smean = smean;
0035 end
0036 fname = [mFolder, '/mp'];
0037 save(fname, 'mp');
0038 return;
```

A.2 Principal Marker Selection

```
0001 %#############################################################################
0002 %The following function is used to group the correlated mocap features    #
0003 %together. It first computes the principal components out of the poses in #
0004 %the data set. It then calls a clustering method to cluster the features  #
0005 %by their weights on the principal component space.                       #
0006 %#############################################################################
0007 function [clusters, mcIdx, C, V] = pfaFeatureClustering(M, EnergyRatio)
0008
0009 nFrm = size(M,1);
0010 VarDim = size(M,2);
0011 M_mean = mean( M, 1 );
0012 for j = 1: nFrm
0013     M(j,:) = M(j,:) -  M_mean;
0014 end;
0015 cov = M' * M;
0016 [pc,latent,explained] = pcacov(cov);
0017 q = 0;
0018 Additional = 2;
0019 EnergySum = 0;
0020 for i = 1 : VarDim
0021     EnergySum = EnergySum + explained(i);
0022     if ( EnergySum / 100 >= EnergyRatio  )
0023         q = i;
0024         break;
0025     end;
0026 end;
0027 V = zeros( VarDim, q);
0028 for i = 1 : VarDim
0029     V(i,:) = abs(pc(i,[1:q]));
0030 end;
0031 K = q + Additional;
0032 iSelect = 0;
0033 Idxs = zeros(1, VarDim);
0034 [IDX,C] = kmeans(V, K, 'replicates', 400, 'maxiter', 6000);
```

```
0035 iSelect = 1;
0036 Idxs(1,:) = IDX';
0037 [clusters, mcIdx] = FeaturePFA2( C, V );
0038 return;
```

```
0001 %#########################################################################
0002 %The following function is used to cluster mocap features and group       #
0003 %correlated features together.                                             #
0004 %#########################################################################
0005 function [clusters, mcIdx] = pfaFeatureClustering(C, V)
0006
0007 K = size(C, 1);
0008 VarDim = size(V, 1);
0009 nPrn_IDX = zeros(1, K);
0010 for i=1:K
0011     cls(i).list = [];
0012     cls(i).distances = [];
0013 end
0014 for  j = 1:VarDim
0015     minDis = 1e+10;
0016     for i = 1:K
0017         dis = norm(V(j,:) - C(i,:));
0018         if dis < minDis
0019             minDis = dis;
0020             minIdx = i;
0021         end;
0022     end
0023     mcIdx(j) = minIdx;
0024     cls(minIdx).list = [cls(minIdx).list, j];
0025     cls(minIdx).distances = [cls(minIdx).distances, minDis];
0026 end;
0027 for i=1:K
0028     [sortDis, idx] = sort(cls(i).distances);
0029     clusters(i).sortedIdx = cls(i).list(idx);
0030     clusters(i).distances = sortDis;
0031 end
0032 return;
```

```
0001 %#########################################################################
0002 %The following function takes clusters of correlated marker features as   #
0003 %input and selects a set of principal markers by their importances.       #
0004 %#########################################################################
0005 function pfIdxList = selectPrincipalMarkers(clusters, mcIdx, C, V, numOfMarkers)
0006
0007 feature = mcIdx;
0008 numOfDims = 3;
0009 for i=1:numOfMarkers
0010     tmp = [];
0011     v = [];
0012     for j=1:numOfDims
0013         tmp = [tmp, feature((i-1)*numOfDims+j)];
0014         v = [v, norm(V((i-1)*numOfDims+j,:) - C(feature((i-1)*numOfDims+j),:))];
0015     end
0016     stmp = sort(tmp);
0017     index = 1;
0018     for j=2:numOfDims
0019         if (stmp(j) == stmp(j-1))
```

```
0020             continue;
0021         else
0022             index = index + 1;
0023         end;
0024     end
0025     mWeight(i) = index;
0026     MD(i) = markerDis(stmp, v, V, C);
0027 end
0028 numOfClusters = size(clusters, 2);
0029 for i=1:numOfClusters
0030     numOfMs = size(clusters(i).sortedIdx, 2);
0031     currentM = -1;
0032     clusters(i).features = 0;
0033     for j=1:numOfMs
0034         marker = clusters(i).sortedIdx / 3;
0035         if (currentM ~= marker)
0036             clusters(i).features = clusters(i).features + 1;
0037             currrentM = marker;
0038         end;
0039     end
0040 end
0041 markers = [mWeight', MD'];
0042 [sortedMs, sortedMidx] = sortrows(markers);
0043 prnMarkers = [];
0044 for i=1:40
0045     markerIdx = sortedMidx(i);
0046     flag = 0;
0047     for j=1:numOfDims
0048         clusterIdx = feature((markerIdx-1)*3 + j);
0049         restFeatures = clusters(clusterIdx).features - 1;
0050         if (restFeatures <= 0)
0051             flag = 1;
0052             break;
0053         end;
0054     end
0055     if (flag == 0)
0056         for j=1:numOfDims
0057             clusterIdx = feature((markerIdx-1)*3 + j);
0058             clusters(clusterIdx).features = clusters(clusterIdx).features - 1;
0059         end
0060     else
0061         prnMarkers = [prnMarkers, markerIdx];
0062     end;
0063 end
0064 pfIdxList = [];
0065 for i=1:size(prnMarkers, 2)
0066     for j=1:numOfDims
0067         pfIdxList = [pfIdxList, (prnMarkers(i)-1)*3 + j];
0068     end
0069 end
0070 return;
```

A.3 Motion Compression

```
0001 %##########################################################################
0002 %The following function is used to compress a motion segment. It first    #
0003 %compress frames by their projections onto the linear space spanned by the#
0004 %leading principal components. It then adaptively select key frames and   #
0005 %only stores these key frames' PCA projections.                           #
0006 %##########################################################################
0007 function CM = compressSub(M, e, ers)
0008
0009 [numOfFs,dim] = size(M);
0010 covx = cov(M);
0011 Mean = mean(M,1);
0012 [acs, latent, explained] = pcacov(covx);
0013 se = 0;
0014 total = sum(latent);
0015 for i=1:dim
0016     se = se + latent(i);
0017     if (sqrt((total-se)/40) <= e)
0018         pdim = i;
0019         break;
0020     end;
0021 end
0022 pcs = acs(:,1:pdim);
0023 projs = (M - repmat(Mean,numOfFs, 1)) * pcs;
0024 knot(1) = 1;
0025 kvalues(1:pdim,1) = projs(1,:);
0026 tmp = round(numOfFs/3);
0027 knot(2) = tmp;
0028 kvalues(1:pdim,2) = projs(tmp,:);
0029 tmp = round(numOfFs/1.5);
0030 knot(3) = tmp;
0031 kvalues(1:pdim,3) = projs(tmp,:);
0032 knot(4) = numOfFs;
0033 kvalues(1:pdim,4) = projs(numOfFs,:);
0034 numOfKnots = 4;
0035 newKnot = [];
0036 knotFlag = 1;
0037 while (knotFlag > 0)
0038     newKnot = [];
0039     pp = spline(knot, kvalues);
0040     v = ppval(pp,[1:numOfFs])';
0041     index = 0;
0042     knotFlag = -1;
0043     for i=1:numOfKnots-1
0044         index = index + 1;
0045         newKnot(index) = knot(i);
0046         for j=knot(i):knot(i+1);
0047             dis = norm((projs(j,:) - v(j,:))/40);
0048             if (dis > ers)
0049                 index = index + 1;
0050                 pos = round((knot(i)+knot(i+1))/2);
0051                 newKnot(index) = pos;
0052                 knotFlag = 1;
0053                 break;
0054             end;
0055         end
```

```
0056     end
0057     index = index + 1;
0058     newKnot(index) = numOfFs;
0059     knot = newKnot;
0060     numOfKnots = size(knot,2);
0061     kvalues = projs(knot,1:pdim)';
0062 end
0063 CM.knot = knot;
0064 CM.kvalues = kvalues;
0065 CM.Mean = Mean;
0066 CM.pcs = pcs;
0067 return;
```

A.4 Motion Retrieval

```
0001 %#############################################################################
0002 %The following function is used to query the motion database for similar  #
0003 %motions.                                                                   #
0004 %#############################################################################
0005 function query(infoname, qFolder, qfile, weigh)
0006
0007 load NormIdx;
0008 load(infoname);
0009 fname = [fvecFolder, '/fmean'];
0010 tmp = load(fname);
0011 fmean = tmp.fmean;
0012 fname = [fvecFolder, '/rdim'];
0013 tmp = load(fname);
0014 rdim = tmp.rdim;
0015 fname = [fvecFolder, '/evec'];
0016 tmp = load(fname);
0017 evec = tmp.evec;
0018 fname = [mbFolder, '/mod*'];
0019 allModels = dir(fname);
0020 fname = [qFolder, '/', qfile];
0021 tmp = load(fname);
0022 M = tmp.M_N(:, NormIdx);
0023 [sizem, numOfDims] = size(M);
0024 current = 1;
0025 offset = 0;
0026 indx = 0;
0027 segIdx = [];
0028 fvec = [];
0029 choice = 1;
0030 tpcs = [];
0031 while 1
0032     sIdx = ppcaSegSub(M(current:end,:), per, block, delta, 150);
0033     st = offset + 1;
0034     ed = offset + sIdx;
0035     cv =  cov(M(st:ed,:));
0036     Mean = mean(M(st:ed,:), 1);
0037     index = 0;
0038     indx = indx + 1;
0039     svec = [];
0040     for j=1:numOfDims
```

```
0041         for k=j:numOfDims
0042             index = index + 1;
0043             svec(index) = cv(j,k);
0044         end
0045     end
0046     feavec = [svec / sum(svec), Mean * weigh];
0047     segIdx(indx,:) = [offset+1, offset+sIdx];
0048     fvec(indx, :) = (feavec - fmean) * evec(:,1:rdim);
0049     offset = offset+sIdx;
0050     if (offset < sizem)
0051         current = offset+1;
0052         continue;
0053     else
0054         break;
0055     end;
0056 end
0057 numOfModels = size(allModels, 1);
0058 while (1)
0059     mflags = -1 * ones(1, numOfModels);
0060     mnums = -1 * ones(1, numOfModels);
0061     for i=1:numOfModels
0062         allMatches(i).matches = [];
0063         fs(i).fids = [];
0064     end
0065     mFolder = [mbFolder, '/', allModels(midx).name];
0066     fname = [mFolder, '/clusterTree'];
0067     tmp = load(fname);
0068     root = tmp.root;
0069     numOfFs = size(fvec, 1);
0070     fids = [];
0071     for i=1:numOfFs
0072         [fids(i)] = fclassSubK(root, fvec(i,:));
0073     end
0074     fs(midx).fids = fids;
0075     match = matching(fids, segIdx, dFolder, mFolder, segFolder, range, ...
0076         tol, fvec, fvecFolder, M, ntol);
0077     numOfRs = size(match, 2);
0078     allMatches(midx).matches = match;
0079 end
0080 return;
```

```
0001 %#########################################################################
0002 %The following function is called by the query function to search in the  #
0003 %motion database index space for the closest matches to the querried      #
0004 %motion segment                                                           #
0005 %#########################################################################
0006 function matches = matching(tcidx, tcb, dFolder, mFolder, segFolder, ...
0007     range, tol, tvec, fvecFolder, x, ntol, dtw)
0008
0009 global fvec;
0010 load NormIdx;
0011 fname = [mFolder, '/clusterTree'];
0012 tmp = load(fname);
0013 root = tmp.root;
0014 fname = [dFolder, '/M*N*.mat'];
0015 allFiles = dir(fname);
0016 fname = [mFolder, '/fs'];
```

```
0017 tmp = load(fname);
0018 fs = tmp.fs;
0019 numOfSeqs = size(fs, 2);
0020 fname = [segFolder, '/segs'];
0021 tmp = load(fname);
0022 segs = tmp.segs;
0023 fname = [fvecFolder, '/fvec'];
0024 tmp = load(fname);
0025 fvec = tmp.fvec;
0026 clear tmp;
0027 numOfTS = size(tcidx, 2);
0028 tflags = -1 * ones(1, numOfTS);
0029 numOfTF = tcb(numOfTS, 2) - tcb(1,1) + 1;
0030
0031 cPercent = [];
0032 for i=1:numOfTS
0033     cPercent(i) = (tcb(i,2) - tcb(i,1) + 1) / numOfTF;
0034 end
0035 [fhits, hsIdx] = fclaMN(mFolder, segFolder, numOfSeqs, cPercent, ...
0036     tvec, tol, range);
0037 numOfH = size(hsIdx, 2);
0038 oid = 0;
0039 match = [];
0040 scores = [];
0041 for i=hsIdx
0042     hits = [];
0043     candIdx = fs(i).cidx;
0044     ccb = segs(i).segIdx;
0045     numOfS = size(candIdx, 2);
0046     fvecSegs = segs(i).segs;
0047     candPer = [];
0048     for j=1:numOfS
0049         candPer(j) = ccb(j,2) - ccb(j,1) + 1;
0050     end
0051     total = sum(candPer);
0052     candPer = candPer / total;
0053     for j=1:numOfS
0054         hits(j).hit = [];
0055         for k=1:numOfTS
0056             if (ismember(k, fhits(fvecSegs(j)).hits))
0057                 hits(j).hit = [hits(j).hit, k];
0058             end;
0059         end
0060     end
0061     index = 0;
0062     for st=1:numOfS
0063         if (size(hits(st).hit, 2) < 0)
0064             continue;
0065         end;
0066         for ed=numOfS:-1:st
0067             if (size(hits(ed),2) < 0)
0068                 continue;
0069             end;
0070             sumc = 0;
0071             sumu = 0;
0072             hit = 0;
0073             tflags = -1 * ones(1, numOfTS);
```

```
0074             for j=st:ed
0075                 sizeh = size(hits(j).hit, 2);
0076                 if (sizeh > 0) t
0077                     sumc = sumc + ccb(j,2)-ccb(j,1)+1;
0078                     for k=1:sizeh
0079                         thit = hits(j).hit(k);
0080                         if (tflags(thit) < 0)
0081                             sumu = sumu + cPercent(thit);
0082                             tflags(thit) = 1;
0083                         end;
0084                     end
0085                 end;
0086             end
0087             totals = ccb(ed,2)-ccb(st,1)+1;
0088             portion = sumc / totals;
0089             if (sumu >= range)
0090                 if (portion >= range &&  sumc <= numOfTF / dtw && sumc ...
0091                         >= numOfTF * dtw)
0092                     perq = sumu;
0093                     perc = portion;
0094                     index = index + 1;
0095                     oid = oid + 1;
0096                     match(oid).seqId = i;
0097                     match(oid).seqname = allFiles(i).name(1:end-4);
0098                     match(oid).seg = candIdx(st:ed);
0099                     match(oid).pos = [st:ed];
0100                     match(oid).mscore = perq * perc;
0101                     match(oid).qscore = perq;
0102                     match(oid).cscore = perc;
0103                     match(oid).cidx = fs(i).cidx;
0104                     scores(oid) = perq;
0105                     hit = 1;
0106                     fname = [dFolder, '/', allFiles(i).name];
0107                     tmp = load(fname);
0108                     y = tmp.M_N(ccb(st,1):ccb(ed,2), NormIdx);
0109                     match(oid).avg = norm(mean(x,1)-mean(y,1));
0110                     fc = fvec(fvecSegs(st:ed),:);
0111                     match(oid).avg2 = cs3(tvec, fc);
0112                     scores(oid) = match(oid).avg2;
0113                 end;
0114             else
0115                 hit = -1;
0116                 break;
0117             end;
0118         end
0119     end
0120 end
0121 [scs, sids] = sort(scores, 'descend');
0122 matches = match(sids);
0123 return;
```

Bibliography

Alexa, M. and Müller, W. (2000). Representing animations by principal components. *Comput. Graph. Forum*, 19(3).

Arikan, O. (2006). Compression of motion capture databases. *ACM Trans. Graph.*, 25(3):890–897.

Arikan, O. and Forsyth, D. A. (2002). Interactive motion generation from examples. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 483–490, New York, NY, USA. ACM Press.

Badler, N. I., Hollick, M. J., and Granieri, J. P. (1993). Real-time control of a virtual human using minimal sensors. *Presence*, 2(1):82–86.

Barbič, J., Safonova, A., Pan, J.-Y., Faloutsos, C., Hodgins, J. K., and Pollard, N. S. (2004). Segmenting motion capture data into distinct behaviors. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, pages 185–194, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada. Canadian Human-Computer Communications Society.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.

Berndt, D. J. and Clifford, J. (1994). Using dynamic time warping to find patterns in time series. In *KDD Workshop*, pages 359–370.

Brand, M. and Hertzmann, A. (2000). Style machines. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 183–192, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Brand, M. and Kettnaker, V. (2000). Discovery and segmentation of activities in video. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):844–851.

Bregler, C. and Omohundro, S. M. (1994). Nonlinear image interpolation using manifold learning. In *NIPS*, pages 973–980.

Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

Chai, J. and Hodgins, J. K. (2005). Performance animation from low-dimensional control signals. *ACM Trans. Graph.*, 24(3):686–696.

Chan, K.-P. and Fu, A. W.-C. (1999). Efficient time series matching by wavelets. In *ICDE*, pages 126–133.

Cipolla, R., Robertson, D., and Boyer, E. (1999). Photobuilder - 3d models of architectural scenes from uncalibrated images. In *ICMCS, Vol. 1*, pages 25–31.

Cohen, I., Tian, Q., Zhou, X., and Huang, T. (2002). Feature selection using principal feature analysis.

Cox, T. F., Cox, M. A. A., and Cox, T. F. (2000). *Multidimensional Scaling, Second Edition*. Chapman & Hall/CRC.

Craig, J. J. (1989). *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Das, G., Gunopulos, D., and Mannila, H. (1997). Finding similar time series. In *PKDD*, pages 88–100.

Dash, M., Liu, H., and Motoda, H. (2000). Consistency based feature selection. In *PAKDD*, pages 98–109.

Debevec, P. E., Taylor, C. J., and Malik, J. (1996). Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH*, pages 11–20.

Dorfmller-Ulhaas, K. (2003). Robust optical user motion tracking using a kalman filter. Technical Report 2003-6, Klaus Dorfmller-Ulhaas.

Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification (2nd Edition)*. Wiley-Interscience.

Faloutsos, C., Ranganathan, M., and Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. In *SIGMOD Conference*, pages 419–429.

Ferrari-Trecate, G. and Muselli, M. (2002). A new learning method for piecewise linear regression. In *ICANN '02: Proceedings of the International Conference on Artificial Neural Networks*, pages 444–449, London, UK. Springer-Verlag.

Fleet, D. J., Black, M. J., Yacoob, Y., and Jepson, A. D. (2000). Design and use of linear models for image motion analysis. *International Journal of Computer Vision*, 36(3):171–193.

Fod, A., Mataric, M. J., and Jenkins, O. C. (2002). Automated derivation of primitives for movement classification. *Auton. Robots*, 12(1):39–54.

Forbes, K. and Fiume, E. (2005). An efficient search algorithm for motion data using weighted pca. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 67–76, New York, NY, USA. ACM Press.

Gortler, S. J., Grzeszczuk, R., Szeliski, R., and Cohen, M. F. (1996). The Lumigraph. In *Proceedings of SIGGRAPH 1996*, pages 43–54, New Orleans.

Gottschalk, S., Lin, M. C., and Manocha, D. (1996). Obbtree: A hierarchical structure for rapid interference detection. In *SIGGRAPH*, pages 171–180.

Grochow, K., Martin, S. L., Hertzmann, A., and Popović, Z. (2004). Style-based inverse kinematics. *ACM Trans. Graph.*, 23(3):522–531.

Guo, S. and Robergé, J. (1996). A high-level control mechanism for human locomotion based on parametric frame space interpolation. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, pages 95–107, New York, NY, USA. Springer-Verlag New York, Inc.

Gupta, S., Sengupta, K., and Kassim, A. A. (2002). Compression of dynamic 3d geometry data using iterative closest point algorithm. *Computer Vision and Image Understanding*, 87(1-3):116–130.

Guskov, I. and Khodakovsky, A. (2004). Wavelet compression of parametrically coherent mesh sequences. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 183–192, New York, NY, USA. ACM Press.

Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57.

Hall, M. A. (2000). Correlation-based feature selection for discrete and numeric class machine learning. In *ICML*, pages 359–366.

Herda, L., Fua, P., Plänkers, R., Boulic, R., and Thalmann, D. (2000). Skeleton-based motion capture for robust reconstruction of human motion. In *CA*, pages 77–.

Hinton, G. E., Revow, M., and Dayan, P. (1994). Recognizing handwritten digits using mixtures of linear models. In *NIPS*, pages 1015–1022.

Hornung, A. and Sar-Dessai, S. (2005). Self-calibrating optical motion tracking for articulated bodies. In *VR '05: Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality*, pages 75–82, Washington, DC, USA. IEEE Computer Society.

Huitema, H. and van Liere, R. (2000). Interactive visualization of protein dynamics. In *VISUALIZATION '00: Proceedings of the 11th IEEE Visualization 2000 Conference (VIS 2000)*, Washington, DC, USA. IEEE Computer Society.

Hyvarinen, A., Karhunen, J., and Oja, E. (2001). *Independent Component Analysis*. Wiley-Interscience.

Ibarria, L. and Rossignac, J. (2003). Dynapack: space-time compression of the 3d animations of triangle meshes with fixed connectivity. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 126–135, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

Ikemoto, L., Arikan, O., and Forsyth, D. (2006). Knowing when to put your foot down. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 49–53, New York, NY, USA. ACM Press.

Iwai, Y., Manjoh, K., and Yachida, M. (2002). Gesture and posture estimation by using locally linear regression. In *AMDO*, pages 177–188.

Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323.

Jolliffe, I. T. (1986). *Principal component analysis*. Springer-Verlag.

Karni, Z. and Gotsman, C. (2004). Compression of soft-body animation sequences. *Computers & Graphics*, 28(1):25–34.

Keogh, E. J. (2002). Exact indexing of dynamic time warping. In *VLDB*, pages 406–417.

Keogh, E. J., Palpanas, T., Zordan, V. B., Gunopulos, D., and Cardle, M. (2004). Indexing large human-motion databases. In *VLDB*, pages 780–791.

Kim, S.-W., Park, S., and Chu, W. W. (2001). An index-based approach for similarity search supporting time warping in large sequence databases. In *ICDE*, pages 607–614.

Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artif. Intell.*, 97(1-2):273–324.

Kovar, L. and Gleicher, M. (2004). Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph.*, 23(3):559–568.

Kovar, L., Gleicher, M., and Pighin, F. (2002a). Motion graphs. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 473–482, New York, NY, USA. ACM Press.

Kovar, L., Schreiner, J., and Gleicher, M. (2002b). Footskate cleanup for motion capture editing. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 97–104, New York, NY, USA. ACM Press.

Kutulakos, K. N. and Seitz, S. M. (2000). A theory of shape by space carving. *International Journal of Computer Vision*, 38(3):199–218.

Lawrence, N. (2004). Gaussian process latent variable models for visualization of high dimensional data. In *NIPS*, pages 329–336.

Lee, J., Chai, J., Reitsma, P. S. A., Hodgins, J. K., and Pollard, N. S. (2002). Interactive control of avatars animated with human motion data. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 491–500, New York, NY, USA. ACM Press.

Lee, S.-L., Chun, S.-J., Kim, D.-H., Lee, J.-H., and Chung, C.-W. (2000). Similarity search for multidimensional data sequences. In *ICDE*, pages 599–608.

Lengyel, J. E. (1999). Compression of time-dependent geometry. In *SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 89–95, New York, NY, USA. ACM Press.

Levoy, M. and Hanrahan, P. (1996). Light Field Rendering. In *Proceedings of SIGGRAPH 1996*, pages 31–42, New Orleans.

Li, Y., Wang, T., and Shum, H.-Y. (2002). Motion texture: a two-level statistical model for character motion synthesis. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 465–472, New York, NY, USA. ACM Press.

Liebowitz, D., Criminisi, A., and Zisserman, A. (1999). Creating architectural models from images. *Comput. Graph. Forum*, 18(3):39–50.

Liu, H. and Motoda, H. (1998). *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, Norwell, MA, USA.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In *5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297.

Magnenat-Thalmann, N. and Seo, H. (2004). Data-driven approaches to digital human modeling. In *3DPVT*, pages 380–387.

McMillan, L. and Bishop, G. (1995). Plenoptic Modeling: An Image-Based Rendering System. In *Proceedings of SIGGRAPH 1995*, pages 39–46, New Orleans.

Mukai, T. and Kuriyama, S. (2005). Geostatistical motion interpolation. *ACM Trans. Graph.*, 24(3):1062–1070.

Müller, M., Röder, T., and Clausen, M. (2005). Efficient content-based retrieval of motion capture data. *ACM Trans. Graph.*, 24(3):677–685.

Ng, A. Y. (1998). On feature selection: Learning with exponentially many irrelevant features as training examples. In *ICML*, pages 404–412.

O'Brien, J. F., Bodenheimer, R. E., Brostow, G. J., and Hodgins, J. K. (2000). Automatic joint parameter estimation from magnetic motion capture data. In *Proceedings of Graphics Interface 2000*, pages 53–60.

Oliver, N., Horvitz, E., and Garg, A. (2002). Layered representations for human activity recognition. In *ICMI*, pages 3–8.

Pavlovic, V., Rehg, J. M., and MacCormick, J. (2000). Learning switching linear models of human motion. In *NIPS*, pages 981–987.

Peyrard, N. and Bouthemy, P. (2002). Content-based video segmentation using statistical motion models. In *BMVC*.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1986). *Numerical recipes: the art of scientific computing*. Cambridge University Press, New York, NY, USA.

Pullen, K. and Bregler, C. (2002). Motion capture assisted animation: texturing and synthesis. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 501–508, New York, NY, USA. ACM Press.

Rose, C., Cohen, M. F., and Bodenheimer, B. (1998). Verbs and adverbs: Multidimensional motion interpolation. *IEEE Comput. Graph. Appl.*, 18(5):32–40.

Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326.

Safonova, A., Hodgins, J. K., and Pollard, N. S. (2004). Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Trans. Graph.*, 23(3):514–521.

Salomon, D. (2000). *Data Compression: The Complete Reference*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Sattler, M., Sarlette, R., and Klein, R. (2005). Simple and efficient compression of animation sequences. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 209–217, New York, NY, USA. ACM Press.

Schölkopf, B., Smola, A., and Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Comput.*, 10(5):1299–1319.

Semwal, S. K., Hightower, R. R., and Stansfield, S. A. (1998). Mapping algorithms for real-time control of an avatar using eight sensors. *Presence*, 7(1):1–21.

Tenenbaum, J. B., de Silva, V., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323.

Tipping, M. and Bishop, C. (1999). Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *Journal of the Royal Statistical Society*, 61(3):611–622.

Torgo, L. and Costa, J. P. D. (2003). Clustered partial linear regression. *Mach. Learn.*, 50(3):303–319.

Tsamardinos, I. and Aliferis, C. (2003). Towards principled feature selection: Relevancy, filters and wrappers. In *the Ninth International Workshop on Artificial Intelligence and Statistics (AI&Stats 2003). Florida, USA*.

Vijayakumar, S. and Schaal, S. (2000). Locally weighted projection regression: Incremental real time learning in high dimensional space. In *ICML*, pages 1079–1086.

Vlachos, M., Gunopulos, D., and Kollios, G. (2002). Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684.

Wiley, D. J. and Hahn, J. K. (1997). Interpolation synthesis for articulated figure motion. In *VRAIS '97: Proceedings of the 1997 Virtual Reality Annual International Symposium (VRAIS '97)*, Washington, DC, USA. IEEE Computer Society.

Winter, D. A. (2004). *Biomechanics and Motor Control of Human Movement*. Wiley-Interscience.

Yi, B.-K., Jagadish, H. V., and Faloutsos, C. (1998). Efficient retrieval of similar time sequences under time warping. In *ICDE*, pages 201–208.

Yu, H., Yang, J., Wang, W., and Han, J. (2003). Discovering compact and highly discriminative features or feature combinations of drug activities using support vector machines. In *CSB*, pages 220–228.

Yu, L. and Liu, H. (2003). Feature selection for high-dimensional data: A fast correlation-based filter solution. In *ICML*, pages 856–863.

Zelnik-Manor, L. and Irani, M. (2001). Event-based analysis of video. In *CVPR (2)*, pages 123–130.