Content-based Motion Retrieval Using Vector Space Model

by

Zhunping 'Justin' Zhang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2008

Author ....
Department of Electrical Engineering and Computer Science
August 29, 2008

Certified by ....
Jovan Popović
Associate Professor
Thesis Supervisor

Accepted by ....
Terry P.Orlando
Chairman, Department Committee on Graduate Students

Content-based Motion Retrieval Using Vector Space Model

by

Zhunping 'Justin' Zhang

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2008, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

Abstract

Motion retrieval is the problem of retrieving highly relevant motions in a timely manner. The principal challenge is to characterize the similarity between two motions effectively, which is tightly related to the gap between the motion data's representation and its semantics. Our approach uses vector space model to measure the similarities among motions, which are made discrete using the vocabulary technique and transformation invariant using the relational feature model. In our approach, relational features are first extracted from motion data. then such features are clustered into a motion vocabulary. Finally motions are turned into bag of words and retrieved using vector-space model. We implemented this new system and tested it on two benchmark databases composed of real world data. Two existing methods, the dynamics time warping method and the binary feature method, are implemented for comparison. The results shows that our system are comparable in effectiveness with the dynamic time warping system, but runs 100 to 400 times faster. In comparison to retrieval with binary features, it is just as fast but more accurate and practical.

The success of our system points to several additional improvements. Our experiments reveal that the velocity features improve the relevance of retrieved results, but more effort should be dedicated to determining the best set of features for motion retrieval. The same experiments should be performed on large databases and in particular to test how this performance generalizes on test motions outside the original database. The alternative vocabulary organizations, such as vocabulary tree and random forest, should be investigated because they can improve our approach by providing more flexibility to the similarity scoring model and reducing the approximation error of the vocabulary. Because the bag of words model ignores the temporal ordering of key features, a wavelet model should also be explored as a mechanism to encode features across different time scales.

Thesis Supervisor: Jovan Popović
Title: Associate Professor

Acknowledgments

I want to thank my advisor, Prof. Popovic, for introducing me into this fascinating area and guided me in overcoming the challenges with great advice, patience and encouragement. Without him this work would never be done so enjoyably. I also want to thank my labmates in MIT Computer Graphics Group for their support on both research and life. Especially I want to thanks Sara, Tilke, Eugene, Yeuhi for helping improving the English of my thesis. Finally I want to thank my parents and my sister for their unconditional support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Character animation is a powerful tool to create arbitrary video sequences for story telling and entertainment purposes. This work concerns the problem of human motion retrieval, a central problem in inventing the next generation of character animation solutions. While existing solutions rely on skillful animators and cumbersome hardware, this new solution centers on the concept of data-driven animation, or reusing existing motion. For example, to create a fighting sequence for a new movie, we simply go through sequences from previous movies, find the one that best fits our new story, and use it with minor modifications. This is nothing less than "standing on the shoulder of giants" and will definitely revolutionize the whole industry.

The most traditional way to generate motion is keyframe animation. In keyframe animation, the animator specifies a sparse list of 3D character key-poses, and then the system interpolate between each adjacent pair of key poses to obtain the animation sequence. This process has been improved with inverse kinematics and other posing systems. However, such a solution still requires skill and manual effort.

Motion capture provides an alternative by capturing motions from real humans. Markers are attached to the human body, and their movements are recorded with a number of cameras in the form of video sequences. From this point, triangulation is used to recover the 3D movements of the markers from these videos. Motion capture creates authentic human movement with little animator input. This makes it popular.

As motion capture gets popular, the amount of motion data increases dramatically.

This leads to the emerging "motion reuse" field. There however is a subtle catch. The data-driven approach favors larger a database over smaller ones. One way to understand it is that the more data we have, the more likely we will find a good motion, and thus the more likely we can get what we want with little effort. The catch is that the larger database also has its downside. It requires more effort to go through and to search. An effective and efficient data retrieval tool is necessary for this data-driven method to work, as illustrated in Figure 1-1. This motion retrieval problem is hard, not well solved, and is the subject of this thesis.

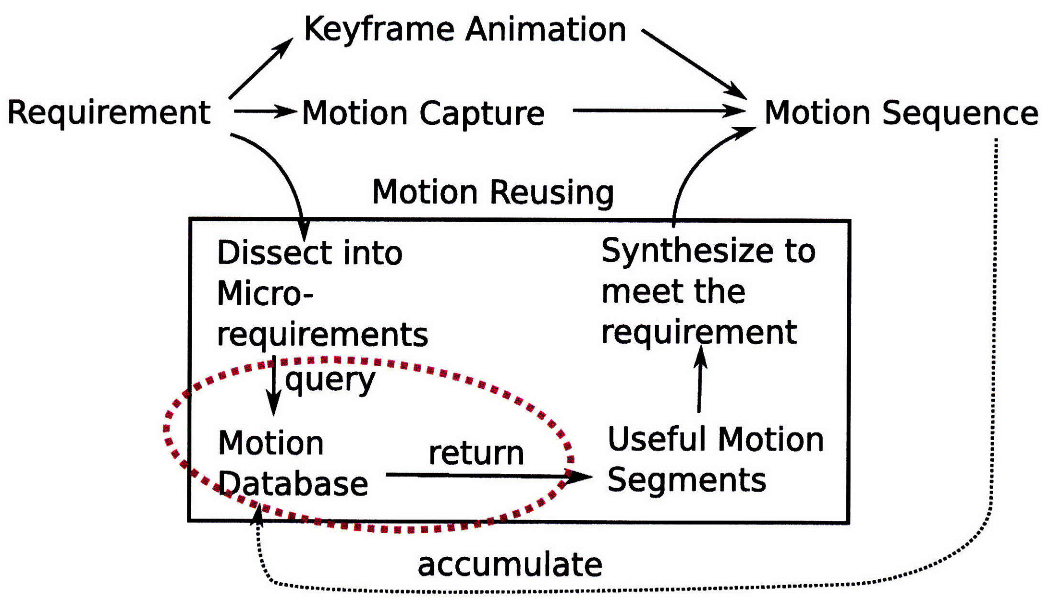

Figure 1-1: The emerging of motion reusing as the future motion creation framework demands a highly effective and efficient motions retrieval system. Here we show three generations of motions creation methods: keyframing, motion capture, and motion reusing. Keyframing was replaced by motion capture due to the latter's lower cost and high authenticity. Compared to motion capture, motion reusing promises more flexibility on the creation of motions and rich content from the large amount of data accumulated from motion capture. In the reusing framework, motion retrieval is an integral subsystem that supplies the system with relevant motion data.

Before we dive into the problem, let us first constrain the problem with two assumptions. First is the content-based retrieval constraint. It assumes that the requirement of the retrieval is given in the form of an example motion. This motion is given by the animator to specify what kind of motion she is looking for. The system's goal is to find motions that are as similar to this motion as possible. This defines the content-based retrieval framework. Second is the manually labeled groundtruth assumption. It assumes that every motions, including the query example, is manually labeled into several categories to serve as groundtruth. Given a query example of label i, the correct returned result is all other motions labelled with i. By comparing the label of the example and the labels of the motions returned by the system, we can evaluate the return list's quality quantitatively. Figure 1-2 shows an illustration of this formulation. Please note that labels are not available to the retrieval system. Labels are only used during evaluation. This is the actual problem being solved by our approach.

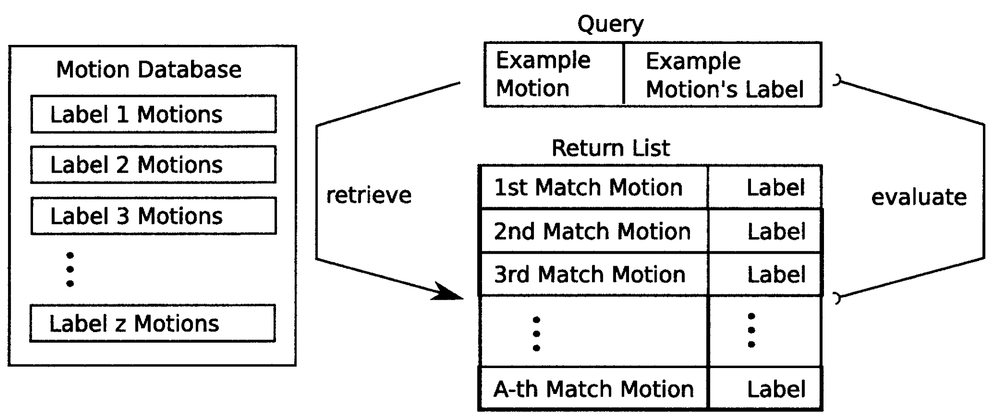

Figure 1-2: This figure shows the content-based motion retrieval problem and its evaluation. The system takes an example motion as the input, and tries to find motions in database that are as similar to the example as possible. The system has no access to the labels, and the shown organization of motions by labels is completely conceptual. In reality motions are organized without the aid of labels. The return lists can be evaluated by comparing its labels with the example's label.

Let us analyze the difficulties of the problem. The system needs to process search orders intelligently enough and fast enough. Missing either one of these will make the system useless. To enable a computer to screen a dataset intelligently, we need to equip it with a reasonable understanding of human motion. This implies to give it the ability to infer semantics from form, a highly non-linear process. This is hard because: first, we don't understand how humans do it; second, even if we understand it, it is likely to be too complicated to be computed efficiently. Our best hope now is to find an approach that is like a sweet point, which is effective enough and efficient enough, such that it can be useful in real animation practice.

The representation of motion data itself involves some of the difficulties. All motions are physically geometric data, represented as Cartesian coordinates or joint angles. Such geometric data does not expose the motion semantics. First, Cartesian coordinates are variant to translation, rotation, and scaling, while semantics is not. Second, the geometric data are in continuous space, while the semantics is ultimately discrete. Third, the motion sequence is simply an assemble of motion frames, while the relation between the frame order and semantics is very complicated. It turns out that in each frame, the geometric representation has much redundant information than its semantics, while on the whole sequence, the simple manner of frame listing ignores much hidden information.

Unfortunately these difficulties cannot be solved by existing techniques. Time alignment aligns two motions in temporal order such that their averaged frame-to-frame difference is minimal. This makes a strong assumption that motions have to be matched monotonically. There have been much efforts on simplifying the single frame representation, using models like PCA for dimension or space partitioning. These will be reviewed in the next chapter. Our goal is to propose a solution that addresses all these difficulties and come up with a system that actually work.

The center of the focus is how to come up with a similarity model that is both effective and fast to compute. Our similarity model breaks up into three parts. First we need to find ways to cancel out transformations that are irrelevant to the semantics. We found out that the relational features is a good method for this purpose [4, 22]. It computes the geometric relations between skeletal parts and removes Euclidean transformation from the data.

Second, we want to compact the space. The output of the relational features is still very high dimensional. We basically have two choices: dimension reduction or discretization. The former reduces the space down to a low dimensional subspace, while the latter partitions the space into a finite number of cells. We go for the latter one because we believe that although human poses are numerous, we should be able to enumerate them into a finite set of thousands or tens of thousands of unique poses. So intuitively, we should be able to have a finite pose vocabulary. We will use a clustering method to partition the real space into a finite number of cells to generate such vocabulary. This packs the motion frames into a finite and discrete set of meaningful components.

Third, we want to uncover hidden information among frames. Instead of using time alignment, we go for the vector-space model [27]. We did not used time alignment because the monotonicity constraint is too strict and it is too costly to compute. The vector-space model treat motion as a bag of words, and computes a histogram from it. Then this histogram is used as the indicator of semantics. This is both proven to be effective in capturing the semantics in text and image retrieval field, and is efficient to compute.

In summary, we explore a motion-retrieval technique that removes the transformation variance from the geometric representation using relational features, then cluster such relational features into a compact finite vocabulary, and finally translate motions into a bag of words and retrieve using the vector-space model.

We evaluate this approach by comparing our approach with two other competing methods: the dynamic time warping method [26] and the binary feature system [22]. The dynamic time warping method computes an optimal time alignment between two motions, and use this alignment to compute a similarity between them. It is highly effective, but runs excessively slowly. The binary feature system computes relational features and then binarizes them into 0s and 1s. Then it indexes motions by their binary features and retrieves them using string matching. It runs very fast, but is less effective because it lost quite much information in the binarization step. Furthermore, because it uses string matching, it is highly sensitive to motion noises. The system requires the user to give a fuzzy matching mask to compensate this. This makes the system not automatic and less practical.

Our system, in comparison, is effective, automatic, and fast. It is more effective than the binary feature system at least because it does not binarize the features. The motion vocabulary approach, in contrast, adapts to the distribution of features and creates a higher quality discretization. It is more practical because the vector space model is not sensitive to noise, thus excluding the need for a manually selected fuzzy mask. It is fast because it discretizes motions into a bag of words, which can the searched efficiently as text. We evaluate all three systems on two benchmark databases. The evaluation consists of precision-recall evaluation which tells the change of the return list's accuracy against its completeness, error matrix analysis which shows the proportions of mis-returned motions by labels, and performance evaluation.

Chapter 2

Related Work

Motion retrieval is a cross field problem. First there has been a collection of works directly on the problem of motion retrieval [22, 18, 6, 20, 18, 5]. Although none of such works gives a practical solution to the problem, they tested many important concepts such as relational features, motion-index tree, time alignment in retrieval, and so on. Computer graphics has gone far in trying to manipulate motion data [17, 1, 2, 4, 26, 13, 14]. These give a strong background in understanding the data we are dealing with, and provide powerful tools such as time alignment. Image retrieval is a closely related and much more thoroughly researched field [10, 7, 21, 8, 9, 29, 24, 25, 23]. Some works in this field discuss the fundamental limitation of computers in approximating human perception. They came up with powerful concepts such as feature-based method and vision words. These concepts have changed our way of thinking about the motion retrieval problem. Text retrieval is an old and sophisticated field that supplies insights and tools on generic informational retrieval[27, 3, 28]. Tools like the vector space model and inverted index are tightly integrated into our method. The rest of this chapter reviews above works individually.

2.1 Text Retrieval

Text retrieval is not perfectly solved but well understood. A text document consists of words and is convenient to search by matching them with keywords. It is also possible to extract the semantic meaning of a text document and search according to its semantics. One example is the vector-space model [27]. The vector-space model computes the word histogram of documents, mapping every document into a high dimension space. Every dimension in this space corresponds to a word in vocabulary, and the dimensionality is the size of the vocabulary. The angles among these vectors are good indicators of their semantical similarities. Two powerful add-ups to the vector space model is the inverted index and stop list. Inverted index maintains pointers from every word to the documents that contain it. This allows the system to quickly find out the set of relevant documents given a set of keywords. Stop list is a list of frequent words that the system ignores. This is useful in lowering the theoretical bound of the retrieval algorithm's complexity.

2.2 Image Retrieval

Image retrieval, in contrast, is a multimedia retrieval problem. It is possible to annotate every image such that the technique of keyword matching can be applied. It takes too much time do so and the annotation can not capture the original meaning perfectly. Content-based image retrieval is widely used as an alternative. As with content-based text retrieval, an example image is given, which is compared to database data for the most relevant ones. The similarity metric used for such finding is non-trivial. Content of image is a high level interpretation of the image. Some vision techniques take insights from the human vision system, which is thought to be divided into two levels: the low level vision that reads details in image, and the high level vision that interpret meaning from the collection of such readings.

The feature based approaches mimic the low level vision system. Details in an image, such as corners and edges, are located and transformed into a convenient mathematical form such as points in high dimension Euclidean space. Harris et. al. invented a detector that finds edges and corners in image [10]. Freeman et. al. introduced an architecture for synthesizing filters of arbitrary orientation, and presented its use in edge detection [7]. Lowe introduced a feature filter that is invariant to image scaling, translation and rotation [21].

Then it is up to the higher level system to infer semantic similarities from extracted features. One way to do this is to compute a matching between two sets of features. Constructing a bipartite graph with features as the graph vertices and distances between features as the edge weights, the optimal assignment and hence the matching cost is obtained by solving the bipartite graph matching problem [15].

Another way to do it is the vision word approach. It partitions the feature space into voxels, and matches sets within this structure. The purpose of this is that then a feature set can be approximated by its histogram over all voxels, and matching two histograms is trivial. Grauman et. al. introduced an architecture in which the feature space is partitioned hierarchically by a uniform grid. Feature sets are superimposed over such a grid and their histograms are computed. The similarity among feature sets is computed by combining the histogram overlaps with weights [8]. In essence, this technique gives an approximation of the optimal assignment with a fraction of the cost. Grauman et. al. introduced a partition that is adaptive to the feature distribution. This is shown to reduce the error when the space dimensionality is high [9].

Sivic et. al. constructed a video frame retrieval system using feature vector quantization . In this system, features are extracted using the filters proposed by Lowe [21]. Then the features are clustered and each feature is represented by its cluster-id, translating every image into a bag of integer cluster-ids. Finally such a bag of integers is retrieved by vector-space model as text documents. Such cluster-ids are also called "words" [29]. Nister et. al. introduced a vocabulary tree architecture which allows a larger and more discriminatory vocabulary to be used efficiently. The hierarchical k-means algorithm clusters the features hierarchically. Each feature thus not only corresponds to a word, but a word path in the tree. A scoring system is developed to give weights to words by their depth in the tree . The results show that such tree structure improves the retrieval effectiveness[24]. Philbin et. al. showed a comparison of using flat k-means, hierarchical k-means, and random forest in quantizing features . The random forest approach randomly selects a dimension from a few dimensions with the largest variance and divides the space into two, and continues this process on both cells until a certain variance threshold is met. Several such random space partitions are used complementarily to mitigate quantization errors. The random forest technique is shown to outperform hierarchical k-means in certain tasks [25]. In another related work, Li et. al. showed a framework to extract video sequence semantics using spatial-temporal interest points [23].

2.3 Motion Retrieval

In contrast to image retrieval, motion retrieval needs to take the time dimension into account. The content of a motion is determined by both the skeleton poses and their ordering over time. Time adds complexity to the problem because motions do not match each other uniformly on time. This demands a time alignment between motions. The dynamic time warping algorithm is a classic algorithm to compute an optimal temporal alignment between two sequences [26]. It works by running a dynamic programming algorithm over a pose-to-pose distance matrix between two motions. Although it gives optimal alignment, its monoticity assumption is overly strict and the quadratic time complexity is expensive for retrieval purposes.

Lin stated that it is only necessary to compare the peaks of two curves [18]. Keogh et. al. pointed out the uniform scaling problem in dynamic time warping and proposed a solution using bounding envelopes [12] . Kovar et. al. proposed an approach to locate locally optimal frame correspondences within a database, grow optimal time alignment between such correspondences, and connect them into a web structure for retrieval [13]. Arikan et. al. applied dynamic programming at different levels to create motions according to user's intuitive specification over time [2]. Arikan et. al. proposed a method to synthesize motion from a dataset using randomized search [1]. Lee et. all. proposed a method to synthesize real time motion from a dataset that has been processed by adding transition edges and by clustering similar segments [17].

All time alignment algorithms require an objective function to define what is an optimal alignment. The objective function is usually the averaged aligned frame-to-frame distance. The computation of the distance between two frames is non-trivial. If two frames are represented in joint angles, their distance in the high-dimensional joint angle space is one such metric. If the poses are represented in joint positions and are normalized on translation, rotation and scaling, then their distance in the high-dimensional Cartesian joint coordinate space is another candidate after alignment [14]. These two metrics are not very discriminative and are slow to compute.

A feature-based approach is a powerful alternative. Similar to the feature-based image retrieval solutions, feature-based motion retrieval approaches extract information from sequence based on highly discriminative models. In such models, local information is extracted into high-dimensional feature space, which is discriminative and convenient for computation. Combinatorial geometry features measure the qualitative properties of groups of geometry features that are invariant over certain transformations [4]. Mueller et. al. proposed a set of 31 such combinatorial measures on joints of a skeleton. Examples are the distance between two hands and the angle of the knee joint. In the systems , such features are binarized into 0 and 1 and are shown to be effective in recognizing poses [22]. Lin proposed a set of five features that characterize the orientation and motion of five separate body parts: the torso and the four limbs. Such features are discretized into 0,1,and 2 [18]. Forbes et. al. proposed a weighted PCA-based pose representation [6]. Liu et. al. used principal component analysis to find a subset of principal markers that are representative for the skeleton, essentially projecting pose down to a joint location subspace [20].

A different approach than time alignment is to discretize motions into meaningful discrete components, and then retrieve them as text. This has the advantage of not having the monoticity constraint and is more friendly for fast processing. The model Mueller et. al. proposed essentially divides motions into lists of segments, and then retrieves them using string matching [22]. Liu et. al. proposed a technique that divides motions into piecewise linear components [20].

As a summary of the the integrated system Mueller et. al. reported, the system computes binary relational features of motion, and then divide them into segments.

Such lists of segments are then organized in an inverted index. Given a query example, exact string matching is found by unioning the corresponding inverted indices. The system also allows a feature mask to adjust the fuzziness in the string matching. The system is tested on the CMU motion database [16] and is shown to be effective in several retrieving tasks [22]. Liu et. al. proposed a motion index tree structure and built a retrieval system on it. The motion index tree uses the hierarchical structure of the skeleton to organize the database. The retrieval using such structure is shown to be effective in a mid-sizes labeled database [19]. Chiu et. al. reported a retrieval system that finds a set of candidate motions through a structure termed "index maps" and the uses dynamic time warping to locate the best matches. Such index maps are essentially a segment-posture retrieval subsystem. The skeleton is divided into segments, and the posture of segments are clustered into a look up structure. Given a query example, the system puts its start and end frame into index maps to narrow the candidates and then search with traditional methods [5].

Chapter 3

Approach

Our approach has three steps. In the first step, the system extracts relational features from the motions, generating 19-dimensional feature vectors. In the second step, the system learns a motion vocabulary from features in the dataset and translates features into integers. In the third step, the integer format motion data is retrieved as text using the vector space model. Figure 3-1 illustrates this pipeline. Overall this method has quadratic preprocessing complexity and $O(n \log n)$ querying complexity. Table 3.1 summarizes the notations used.

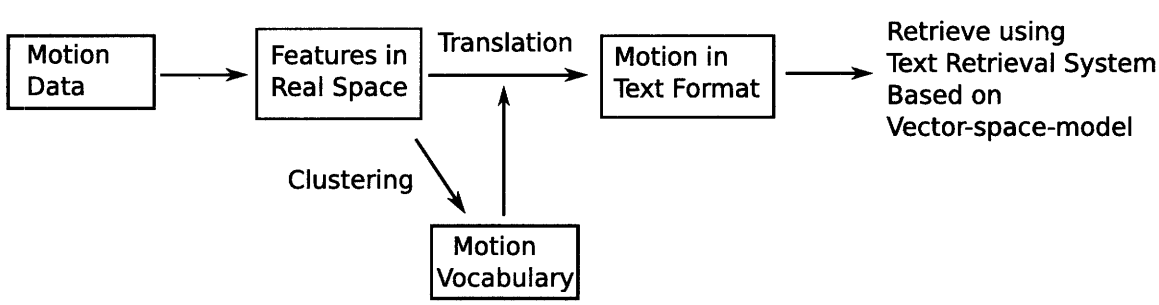

Figure 3-1: Our approach has three steps. First, the system computes relational features from the motion data. Second, the system learns a vocabulary from the features and uses this vocabulary to translate motions into text form. Third, the text formed motions are retrieved using the vector space model.

Symbol	Meaning	Type
$\mathcal{D}$	Whole Motion Database	set
S	One Motion	matrix
$\mathcal{C}$	The Set of All Feature Vectors	set
$\mathcal{V}$	The Motion Vocabulary	set
n	Number of Motions in the Database	integer
m	Total Number of Frames in the Database	integer
α	Dimensionality of Feature Space	integer
k	Number of Motion Words in the Motion Vocabulary Dimensionality of Histogram Vector Space	integer

Table 3.1: Notations

3.1 Feature Extraction

Our approach first extracts relational features from motion. Relational features describe the geometric relation between different parts of the skeleton. Figure 3-2 shows a side-by-side comparison of the Cartesian coordinates representation and the relational features. Cartesian coordinates depend on the coordinate basis and transformations, while relational features do not depend on the coordinate basis and are invariant to transformations. Such features eliminate irrelevant transformation information and are shown to capture pose semantics well [4, 22].

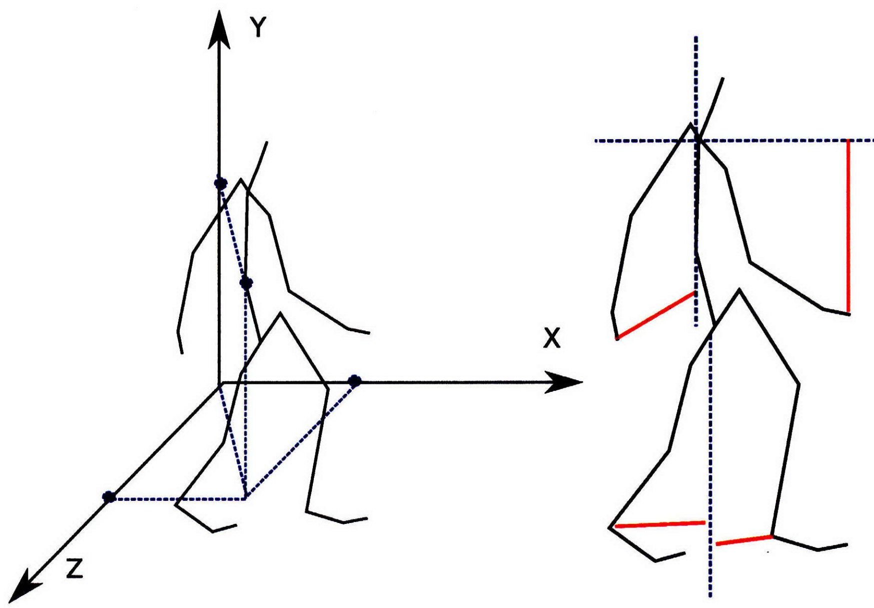

Figure 3-2: Here is a side-by-side comparison of the Cartesian coordinates representation (left) and the relation features (right). In the right figure, the relational feature examples being shown are: the distance between the vertical axis and the hand, the distance between the vertical axis and the two ankles, and the distance between the right hand and the horizontal axis at neck level.

Our system uses 19 relational features in total, as listed in Table 3.2. These features concern the hands' and feets' relative positions to the body. Features 1 through 6 capture the two hands' relative positions to the upper body. This essentially sets up a coordinate system on the torso and projects the hands into it. Features 14 through 19 capture the feets' relative positions to the lower body. This again sets up the lower body coordinate system and represents the feet with it. When we compute how far the left foot is to the front, i.e. feature 14, we use a coordinate system determined by the hips and the right leg. Features 7 through 13 capture the distance between the hands and other body parts. For instance, feature 8 measures the distance between the left hand and both legs. When the left hand is 2 units to the left leg and 1.5 units to the right leg, this feature value is 1.5 units.

Features	Description
1,2	Left/right hand to the front
3,4	Left/right hand raised
5,6	Left/right hand sideways
7	Distance between two hands
8,9	Left/right hand distance to both legs
10,11	Left/right hand distance to head or neck
12,13	Left/right hand distance to hip area
14,15	Left/right foot to the front
16,17	Left/right foot raised
18,19	Left/right foot sideways

Table 3.2: Here is the list of the 19 relational features used in our approach. Features 1 through 6 summarize the two hands' positions in regards to the upper body, features 14 through 19 summarize the two feets' relative position to the lower body, and features 7 through 13 summarize the relation between the hands and other body parts.

This feature extraction maps every motion into a time series in a 19 dimensional real space $\mathbb{R}^{19}$. Figure 3-3 shows an example of a dance motion in this space. Only the first two dimensions are shown in the figure.

We can also include first derivatives of distances into the features, getting 38-dimensional features. This includes more information of the motion but also introduces more noise. In later chapters we experiment with both choices and make a comparison between them. For convenience, we use α to denote the total number of features.

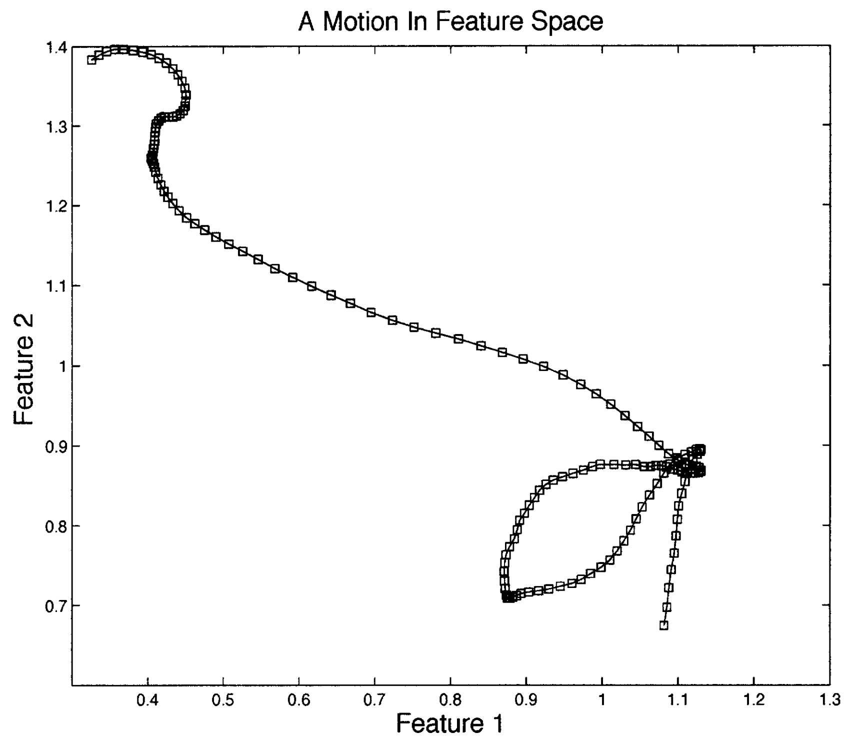

Figure 3-3: A dance motion in feature space. Only the first two dimensions of the 19-dimensional space are shown.

3.2 Learning Vocabulary

Feature extraction turns every motion in a datasetinto a time series in α-dimensional space. Then the system concatenates these time series together into a feature sequence for the whole dataset. This giant time series then has its order discarded and is used as a set. We use $\mathcal{C}$ to denote this set. Figure 3-4 shows such a feature set obtained from a dance database.

Then the system uses the k-means clustering algorithm to cluster $\mathcal{C}$ into a number of clusters. These clusters are words in the motion vocabulary. In the rest of the section, we first introduce the k-means algorithms, and then explain it using an example.

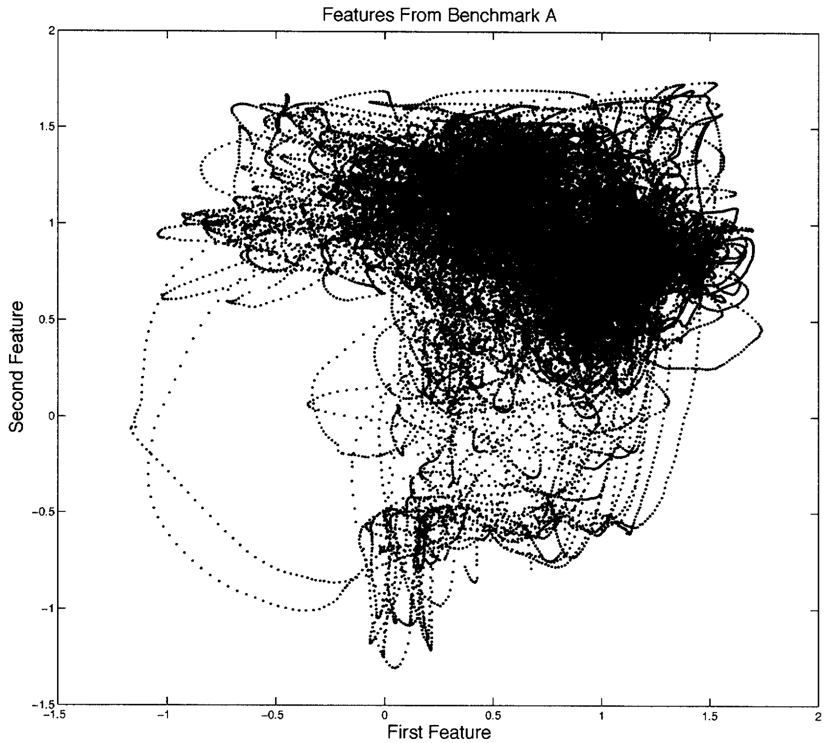

Figure 3-4: Here shows the features from all motions in a dance motion database. Only the first two dimensions of the feature space are shown. A motion vocabulary will be learned from the feature set to help translate the motions into text form.

K-means algorithm

Given a number k, the demanded number of clusters, and a point set with m points $\mathcal{C} = \{c_1, ..., c_m\}$, the algorithm tries to find a k-fold partition of $\mathcal{C}$ that minimizes the averaged variance within each partition. The algorithm makes an initial guess of the cluster centers, and iteratively refines them until convergence. In the following pseudo-code, **centers** represents the cluster centers and **cluster** represents the id of the closest cluster center to each feature.

1. **centers** $\leftarrow$ k random locations in $\mathbb{R}^{19}$
2. **cluster**$[i]$ $\leftarrow$ x s.t. $\|\textbf{centers}[x] - c_i\| = \min_{1 \le j \le k} \|\textbf{centers}[j] - c_i\|$
3. **centers**$[i]$ $\leftarrow$ $\frac{\sum_{1 \le j \le m} \delta(i, \textbf{cluster}[j]) c_j}{\sum_{1 \le j \le m} \delta(i, \textbf{cluster}[j])}$
4. If **clusters** changed during 2, goto 2, otherwise, return **centers, clusters**.

Step 2 calculates the closest centroid to every feature in the feature set. This partitions the feature set into k subsets, with i-th subset being the features closest to i-th cluster centroid. Step 3 recalculates the centroids by taking the average of each cluster and step 4 checks if the process converges. The result is sensitive to the choice of initial cluster centers. The common solution is to run the algorithm several times and use the solution with the smallest average feature-to-centroid error.

Here we use an example to explain the algorithm. Because the clustering happens in α dimensional space, and this cannot be visualized in 2D, we illustrate the process with an artificial 2D dataset. Figure 3-5 (left) shows the dataset and Figure 3-5 (right) shows the clustering results with k being 8.

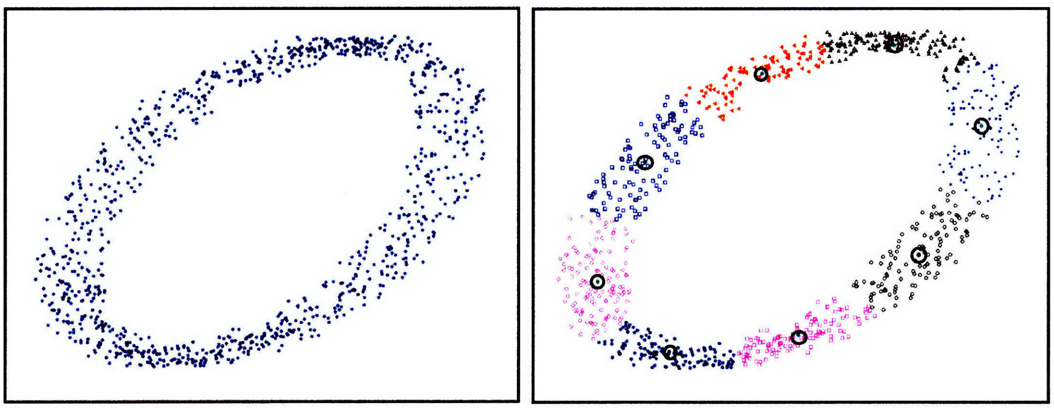

Figure 3-5: Because of the high dimensionality of the real feature set, we use a 2D artificial dataset to explain the algorithm. The left figure shows the example feature set in 2D. The right figure shows the clustering results with k equalling 8.

Such clustering results correspond to a Voronoi diagram where the vertices are the cluster centroids. Figure 3-6 (left) shows the Voronoi diagram of the clustering in Figure 3-5 (right). We call each cell a *motion word* and enumerate them from 1 to k . In the follow text, we use motion words exchangeably with numbers 1 to k. All k words together are called the *motion vocabulary*, denoted by $\mathcal{V}$:

$$\mathcal{V} = \{1, 2, ..., k\}$$

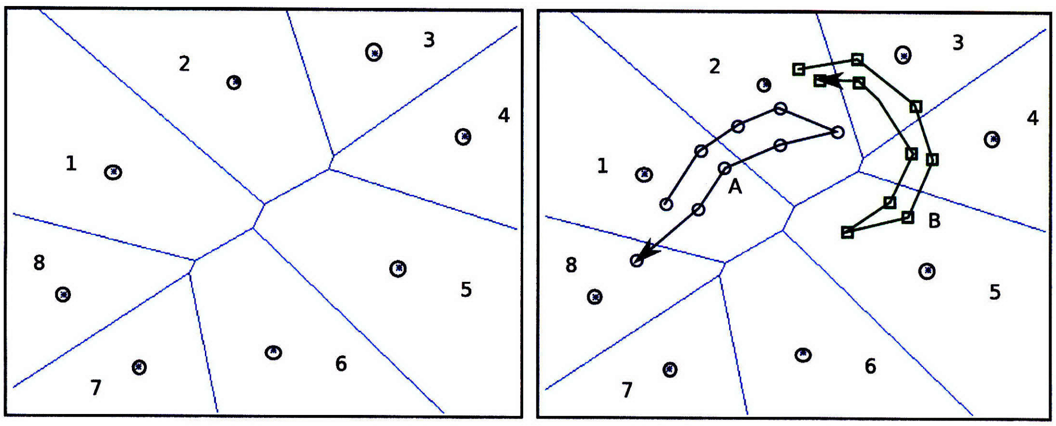

Figure 3-6: The left figure shows the Voronoi diagram corresponding to the clustering in Figure 3-5 (right). Each circle is the centroid of one cluster. The right figure shows how the feature sequences of two motions are translated into motion words. Here the Voronoi diagram is superimposed onto the two feature sequences, where every feature falls into a cell. Then our approach replaces every feature with its cell-id, obtaining an integer list for every motion. In this example, motion A is translated into the list (1,1,2,2,2,2,1,1,8) and motion B is translated into the list (2,3,3,4,5,5,5,4,3,2).

Using this vocabulary, the system translates every feature in a sequence into an integer motion word. For any given feature vector, we can easily determine its motion word by finding the cluster centroid that is closest to it, or equivalently by finding the Voronoi diagram cell that contains it. By finding the motion words for every feature in a motion, the motion is turned into a walk of the cells.

Figure 3-6 right shows an example of how the feature sequences of two motions are translated into motion words. In the figure motion A is a walk of the cells (1,1,2,2,2,2,1,1,8) and motion B is a walk of cells (2,3,3,4,5,5,5,4,3,2). After transforming motions into lists of integers, the system treats them as text documents and retrieves them using a vector-space model.

3.3 Vector Space Model

The vector-space model represents every motion as a histogram vector and measures the difference between motions solely by their vector orientations. These vectors are vectors in k-dimensional space where k is the size of the vocabulary. Every coordinate stands for the number of occurrences of one word in the motion.

Given a query example, it is first transformed into its histogram vector, and then measured against histogram vectors of the dataset. The distance between two vectors is the cosine of their spanned angle. The system sorts the dataset according to the distance between the motions and the example and returns the best ones. Term-frequency and inverse-document-frequency (TF-IDF) are used to improve the histogram vector weights. This process is briefly shown in Figure 3-7.

For the example in Figure 3-6 (right), the histogram of motion A is (4,4,0,0,0,0,0,1) and the histogram of motion B is (0,2,3,2,3,0,0,0). Both vectors are in 8-dimensional vector space. In this 8-dimensional space, every coordinate tells the frequency of one word in the motion. Histogram vectors are normalized into unit vectors, and this does not change the result of cosine similarity metric which only depends on the vectors'

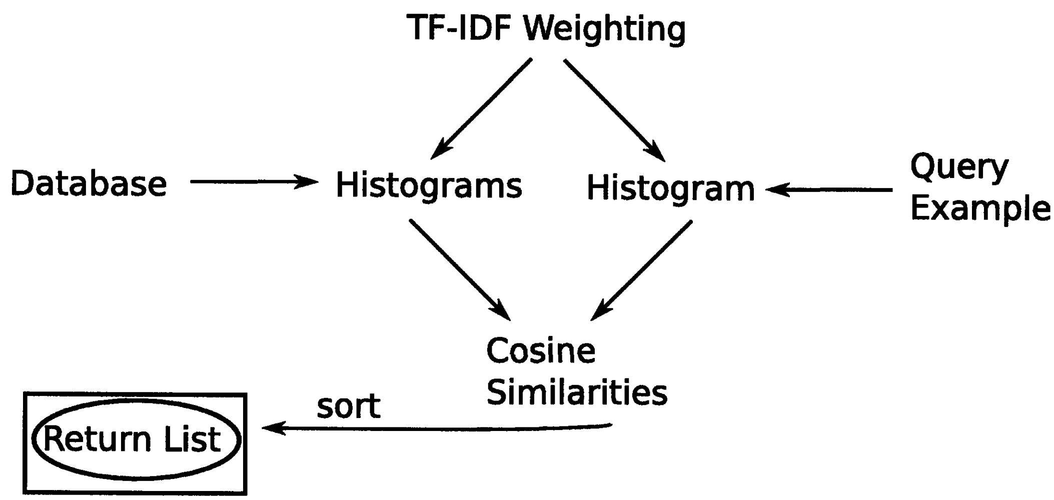

Figure 3-7: The vector space model in our approach consists of three steps. First the system computes the histogram vectors of the database motions and the query motion. Then, it weights such vectors using TF-IDF weighting. Then it computes the cosine similarities between the motions in database and the example. Finally the system sorts the dataset by their similarities to the example and return the best ones.

orientation. This normalization step gives the *term frequency* (TF):

$$\mathrm{tf}_{i,j} = \frac{e_{i,j}}{\sum_l e_{i,l}}$$

where $e_{i,j}$ is the frequency of word j in motion i.

TF-IDF weighting computes *Inverse document frequency* (IDF) to represent the discriminativeness of each words:

$$\mathrm{idf}_i = \log \frac{n}{n_i}$$

where n_i is the number of motions that contain word i and n is the total number of motions. The word-by-word product of the TF weighting and the IDF weighting is the TF-IDF weighting:

$$\mathrm{tfidf}_{i,j} = \mathrm{tf}_{i,j} \cdot \mathrm{idf}_i.$$

TF-IDF weighting uses a balanced measure of the popularity and the discriminativeness of words. The more frequent a word is in a motion, the higher its TF value. The more motions that contain the word, the less discriminative it is and its IDF weight is smaller. In the computation, every motion has a TF weighting vector, and the whole database has a IDF weighting vector:

$$\mathbf{tf}_i = (\mathrm{tf}_{i,1}, \mathrm{tf}_{i,2}, ..., \mathrm{tf}_{i,k})$$
$$\mathbf{idf} = (\mathrm{idf}_1, \mathrm{idf}_2, ..., \mathrm{idf}_k).$$

A motion's TF-IDF vector is defined as the element-to-element product between the IDF vector and its TF vector:

$$\mathbf{tfidf}_i = (\mathrm{tf}_{i,1} \cdot \mathrm{idf}_1, \mathrm{tf}_{i,2} \cdot \mathrm{idf}_2, ..., \mathrm{tf}_{i,k} \cdot \mathrm{idf}_k).$$

Once the TF-IDF vectors of the motions have been determined , the system computes the similarities between every motion in the database and the example. Finally our system sorts motions in the database in descending order by their similarities to the example and returns the highest ones. The similarity used in this process is defined as:

$$\cos\left(\frac{\mathbf{tfidf}_1 \cdot \mathbf{tfidf}_2}{\parallel \mathbf{tfidf}_1 \parallel \parallel \mathbf{tfidf}_2 \parallel}\right).$$

3.4 Complexity

The complexity of the system is best analyzed individually for the preprocessing subsystem and the querying subsystem. The preprocessing subsystem computes a TF-IDF vector for every motions in the database, and the querying subsystem does so for a query example and looks for the closest motions in a database given this example.

3.4.1 Complexity for Preprocessing

• In the feature extraction step, our system spends $O(m)$ time computing relational features for motions in the database, where m is the total number of frames in the database.

• Our system usually spends less than $O(m^2)$ time learning the motion vocabulary. It first runs the k-means clustering algorithm to cluster the features. This usually takes much less than $O(m)$ steps to converge, where each step costs $O(m)$ time to update cluster centers and feature indices. The total time, the product of the number of steps and the cost of each step, is usually much less than $O(m^2)$.

• Our system spends $O(m)$ time translating motions into word sequences and then into histogram vectors where m is the total number of frames in database.

• Our system spends $O(nk)$ time computing the motion's TF-IDF vectors. This operation takes place in k-dimensional vector space, and the cost is the product of n, the total number of motions, and k, the dimensionality of the space.

3.4.2 Complexity for Querying

• In the first step of querying, our system computes the feature sequence from the given example in $O(r)$ time where r stands for the number of frames in the example.

• In the second step, our system transforms the feature sequences into a histogram vector in $O(r)$ time. Motion vocabulary is generated and used as a middle step in $O(r)$ time.

• In the third step, our system computes the motion example's TF-IDF vector in $O(k)$ time.

• In the fourth step, our system computes the cosine similarities between the example and every motion in the database in $O(n)$ time.

• In the fifth step, our system sorts the similarities and returns the best ones in $O(n \log n)$ time.

Our system has an overall preprocessing time complexity of $O(m^2 + nk)$ and an overall querying time complexity of $O(r + k + n \log n)$. Assuming that the number of frames per motion is bounded by a constant: $O(m) = O(n)$ and $O(r) = O(1)$, and the size of the vocabulary k is a constant, then the preprocessing complexity reduces to $O(n^2)$ and the querying complexity to $O(n \log n)$.

3.5 Extension

The algorithm can be further enhanced to improve its accuracy and performance. First we can introduce an incremental learning algorithm to improve the generalization ability of our technique. Because both the motion vocabulary and the IDF vector are trained from the database, it might not generalize well to the query example. If our system incrementally updated the vocabulary using the query example, then the generalization issue would be solved.

The query speed of the approach can be improved by two tricks. The sorting cost in query can be reduced by selecting only a small fraction of motions with top similarities to be sorted. The other speedup is using a combination of inverted index and stop list, which decouples the query cost from the size of the database.

Chapter 4

Experimental Results

Performance is difficult to evaluate because of the enormous variety in motion. We evaluate systems on two manually labeled datasets, both of which were collected from other projects: a Lindy-hop dance dataset and a locomotion dataset with basic human movements. We tested the vector-space model approach against two competing approaches: dynamic time warping and binary feature search. In order to do so, we compute precision-recall evaluation to evaluate the effectiveness of the systems. In addition, we measure our system's sensitivity to feature set selection, and we compute the cross-category retrieval error to see the difference between how systems handle different type of motions. Finally, we instrumented all systems for speed performance. As a snapshot, the results show our method to have a superior combination of effectiveness and efficiency over dynamic time warping and binary feature search.

4.1 Benchmark Datasets

We evaluated the systems with two datasets. The first dataset contains eight different categories of Lindy Hop dancing motions [11]. The second dataset contains five different basic locomotion motions with 77 irrelevant ones from the CMU motion capture database [16]. Table 4.1 summarizes the two benchmarks and Table 4.2 lists the semantical meaning of the Lindy Hop categories.

Benchmarks	A	B
Content	Lindy Hop Dancing	Subset from CMU Database
Number of clips	288	353
Number of frames	54,280	368,654
Framerate	30	120 and 60
Number of joints in skeleton	18	31
Number of categories	8	5
Number of motions In each category	90,72,18,18, 36,18,18,18	46,148,42,10,30

Table 4.1: Basic database information for the two benchmark databases. The first contains eight different categories that indicate the basic dance steps of Lindy Hop [11], the second contains five different basic locomotions from the CMU motion capture database [16].

Step No.	Step Detail
1	Open → close
2	Close → open
3	Open → close crosshand
4	Close crosshand outside turn to open crosshand
5	Close → close
6	Close inside turn to open
7	Close outside turn to open
8	Open crosshand inside turn to close

Table 4.2: This table lists the details of 8 dance steps in benchmark B. Open/close stands for an open or close stance.

4.2 System Implementation

4.2.1 Vector-Space Model-based Retrieving System

The vector-space model (VSM) system implements the design described in chapter 3 with minor improvements. It uses the 19 features in Table 3.3 as the basic set of features and their first order derivatives as optional features. During the clustering stage, the system uses a stratified clustering with k-means approach, but only keeps the leaf nodes. Essentially it does a flat k-means clustering. The system uses three layers of stratified clustering with branching factor six, resulting in 206 clusters. The system runs the k-means algorithm ten times and picks the best one.

4.2.2 The Dynamic Time Warping System

In preprocessing, the dynamic time warping (DTW) system computes the relational features as in the feature computation stage of the vector-space model system. Then the system runs the dynamic time warping algorithm to compute a time alignment between the query example and every motion in the dataset [26]. For each alignment, the system measures a distance by summing the aligned pose-to-pose distance. Finally, the system sorts the datasets descendingly by their distance to the example and returns the top motions. DTW is the most traditional way to solve this problem, and is used as a ruler to measure the effectiveness of the vector-space model.

4.2.3 The Binary Feature System

The binary feature system computes binary features from motions and uses a combination of inverted indices and string matching to find the relevant motions [22]. It uses fuzzy retrieval to loosen the string matching restrictions. We implemented the fuzzy matching and apply a full matching mask when exact matching is tested. One problem with the fuzzy version of the binary feature system is that it requires the user to select a matching mask, which is costly for large scale testing. Instead our system randomly generates 300 masks and retrieves using each of them, then visualizes the results to compare the performance.

4.3 Precision-Recall Evaluation

Precision-recall evaluation allows us to evaluate the quality of a return list by measuring how many useful hits versus how many irrelevant hits have been returned [3]. This method is applied to our problem and indicates that the vector-space model system outperforms the other two. We also show that the effectiveness of the vector-space model is sensitive to the choice of features.

4.3.1 Methodology

Suppose there are R relevant motions in the database. Given a query example, the system returns A motions, among which R_a are relevant. The precision and the recall are defined as:

$$\text{Precision} = R_a/A$$

$$\text{Recall} = R_a/R$$

For any query, the higher the precision and the recall the better. For a ranked hitlist, we get a precision-recall pair for every prefix of the hitlist. This gives us a PR curve. If we draw this curve in 2D coordinates, with the horizontal axis as the recall and the vertical axis as the precision, then for any ranked hitlist, the higher the curve, the better.

To have a single number evaluation of a query instead of a curve, we calculate the *averaged precision*. For every prefix of the hitlist that ended with a relevant motion, we calculate its precision. If Precision_i is the precision of the i-th prefix of the hitlist that ended with a relevant motion, then the averaged precision is defined as:

$$\text{Averaged-Precision} = \frac{1}{R_a}\sum_{i=1}^{R_a}\text{Precision}_i.$$

Our system queries the dataset using its own motions as examples, and reports the average precision-recall performance. It does this for each category separately.

4.3.2 Results

Figure 4-1 shows the precision-recall evaluation results of Benchmark A. The center sub-plot in Figure 4-1 evaluates the querying of dataset A using its category 5 motions. The stars that denote the binary feature system's performance are high on precision, but they are low on recall. This is undesirable because the hit-lists do not include most of the relevant items in the database. There is one star high on recall and low on precision. This corresponds to a hit-list containing all documents in the database.

This hit-list contains all the relevant documents, but it is not useful because it contains all the irrelevant ones as well.

For the PR curves of the VSM system, the precision is high at low recall, meaning the top results in the hit-lists are mostly relevant ones. This is a good because most users only check the first several returns for any given query. The curve roughly exhibits the shape of a line. This implies that as the hit-list gets longer, the precision decreases linearly. In comparison, a concave PR curve would be worse, and a convex PR curve would be better. In the figure, the PR curve of the VSM system in the "Query of Step 6" is a concave PR curve, while the same curve in "Query of step 4" is convex.

Overall, the figure shows that our approach is the most effective method among the three except in the first two cases. In the first two cases, our approach is still very comparable to the dynamic time warping system. In all cases, our approach is more effective than the binary feature approach.

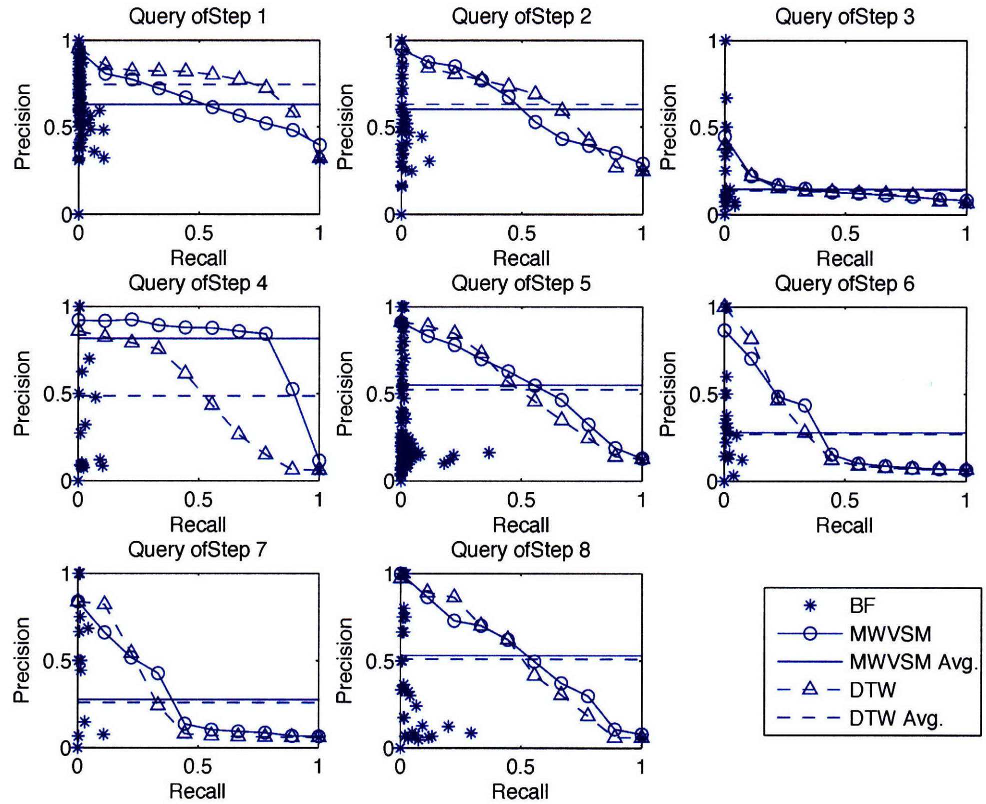

Figure 4-1: Precision-recall evaluation on benchmark A. In the figure, every subplot shows the system's precision-recall evaluation in retrieving motions of a particular category. For the DTW system and the VSM system, the ranked return lists result in PR curves, while for the BF system the unranked return lists result in PR pairs or PR points. In the figure, stars stand for PR points from the BF system, where every star corresponds to the querying using a particular mask, Circle-solid-curve/triangle-dashed-curve are PR curves from the VSM/DTW systems, and solid-line/dashed-line are averaged precisions from the VSM/DTW systems. From the figure we can see that the vector-space model is in general the most effective one. The vector-space model system uses 38-dimensional features.

Figure 4-2 shows the results on benchmark B. In all the plots, the vector-space model's lines are higher than the dynamic time warping approach's. This is good because the dynamic time warping is used as a performance groundtruth. The stars from binary feature system are on the axises except in the first plot. This means the binary feature system either returns all the motions in dataset or returns almost nothing. Overall the vector-space model gives the best result.

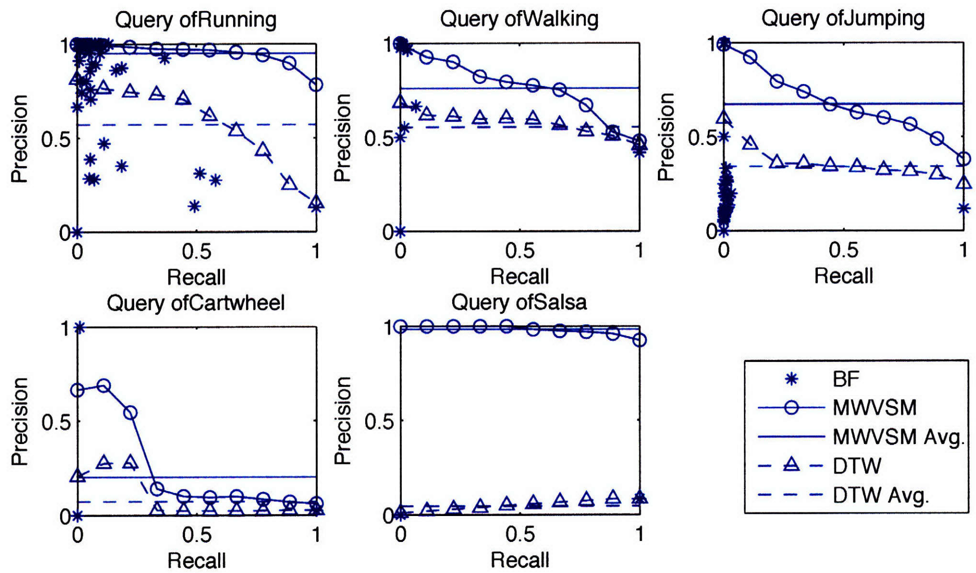

Figure 4-2: Precision-recall evaluation on benchmark B: Stars are precision-recall pairs from the Binary Feature system, Circle-solid-curve/triangle-dashed-curve are PR curves from VSM/DTW. Solid-line/dashed-line are averaged precisions from VSM/DTW. In all five cases, our approach is shown to be the most effective. The vector-space model system uses 38-dimensional feature to get the results in this figure.

4.3.3 Sensitivity to Choice of Features

To examine the effects of including the first derivative information, i.e. the velocity information, among the features, we plot the precision-recall results of the vector-space model system with and without the velocity information. Figure 4-3 and 4-4 show these results.

In Figure 4-3, all the PR curves with velocity are higher than the curves without, indicating that including velocities in the feature improves the performance. In Figure 4-4 however, the curves with velocity are lower than the curves without except in the last plot, indicating that including velocities drops the performance. From these we see that the system's performance is sensitive to the choice of features and it is unclear whether including velocities improves or drops the performance.

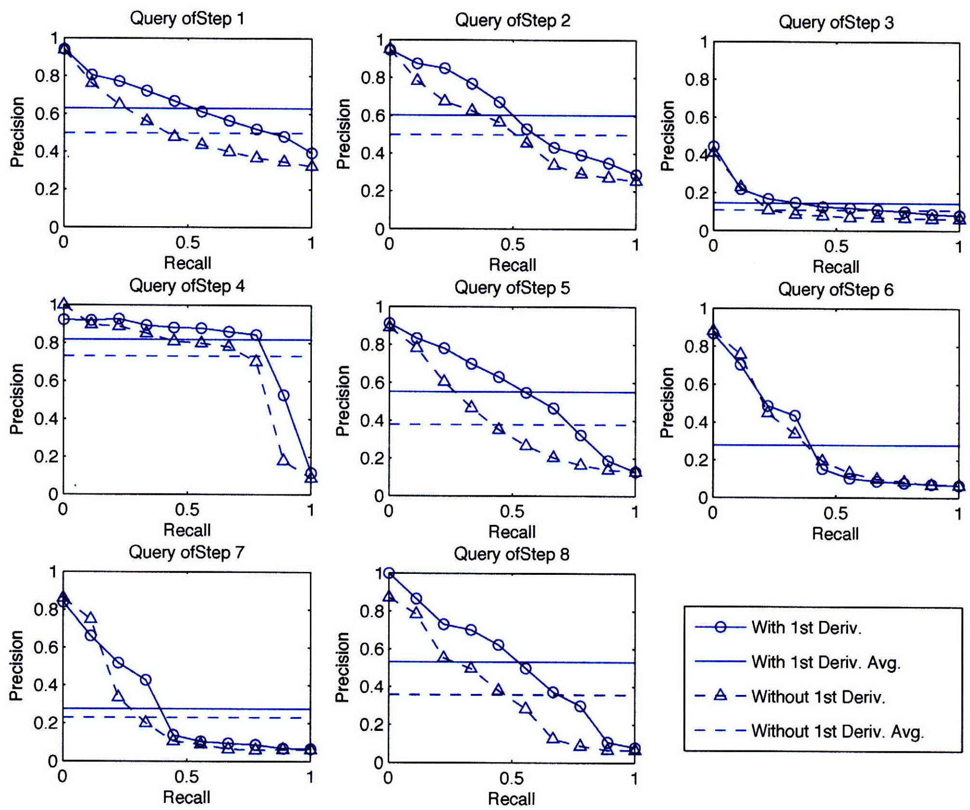

Figure 4-3: Plots of precision-recall evaluation of our system on benchmark A using 19-dimensional and 38-dimensional features. The 19-dimensional features do not include the velocity information while the 38-dimensional features do.

4.4 Cross-Category Mistakes

We evaluate the systems' cross-category mistakes by creating an error analysis matrix, a t-by-t matrix where t is the number of motion categories in the database. Element (row i, column j) is the averaged percentage of category j motion in the top ten hits returned from a query using category i motion, otherwise stated as category j motions mis-returned as category i motions. We compute it as following:

Suppose for a query of category p, the first ten returned motions are

$$R = \{s_1, ..., s_{10}\},$$

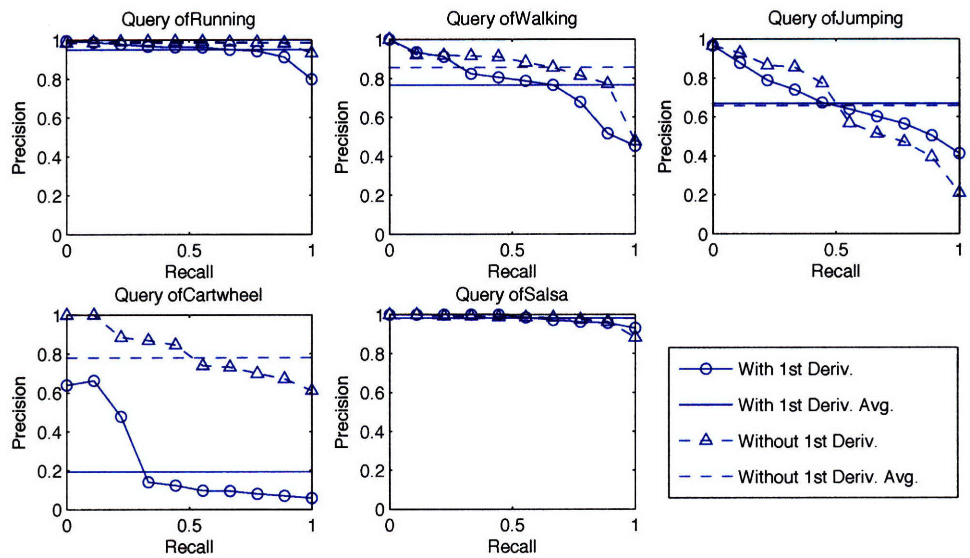

Figure 4-4: Plots of precision-recall evaluation of our system on benchmark B using 19-dimensional and 38-dimensional features. The results show that the velocity information is not helpful on benchmark B.

among which there are h_1 category 1 motions, h_2 category 2 motions, and so on. By dividing h_i by 10, we get the percentage of labels within the first ten returns. In the ideal case, the system would return only relevant motions, resulting in h_i satisfying the following property:

$$h_i = \begin{cases} 1 & i = p \\ 0 & i \neq p \end{cases}$$

We perform the query using all motions of category 1, getting such a percentage vector for each of them. By averaging these percentage vectors, we put the averaged percentage vector into the first row of the error matrix. For the second row, we use the value from queries using category 2 motions, and so on.

Figure 4-5 shows the error analysis matrix on Benchmark A using the dynamic time warping system and the vector-space model system. The systems give mostly relevant returns in the first 10 hits, but make visible mistakes in several spots. Both systems makes two major mistakes. They return category 1 motion in queries of category 3 motions, and confuse category 6 with category 7. Referring to Table 4.2, we find out that category 6 and 7 only differ from each other in the direction of turn, and category 1 and 3 are the same except that the hands cross at the end of step 1. Both of these cases are confusing even for humans. This is a fascinating observation as it suggests that both systems show some resemblance to human cognition.

Figure 4-6 shows the error matrix on benchmark B. The error matrix from vector-space model has higher diagonal values than that from dynamic time warping method. This indicates that the vector-space model has higher overall accuracy, conforming with the precision-recall results.

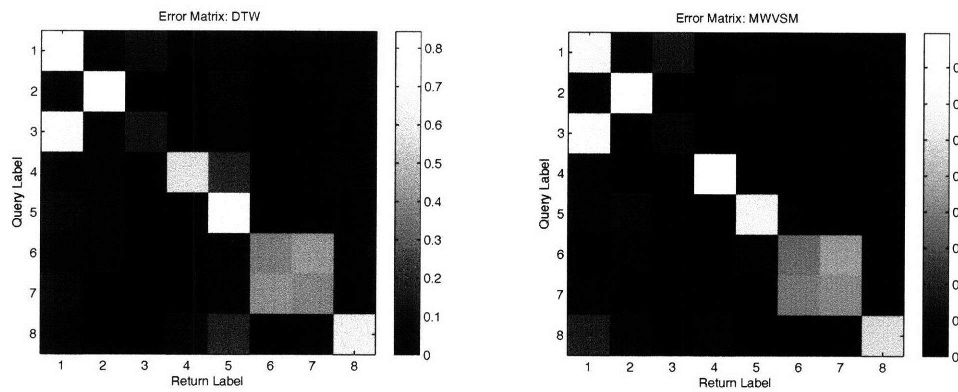

Figure 4-5: Here shows the error matrices on benchmark A using the dynamic term warping system and the vector-space model system. In both figures, the block at row i and column j stands for what percentage of category j motions are returned within the top 10 hits on retrieval using an example from category i.

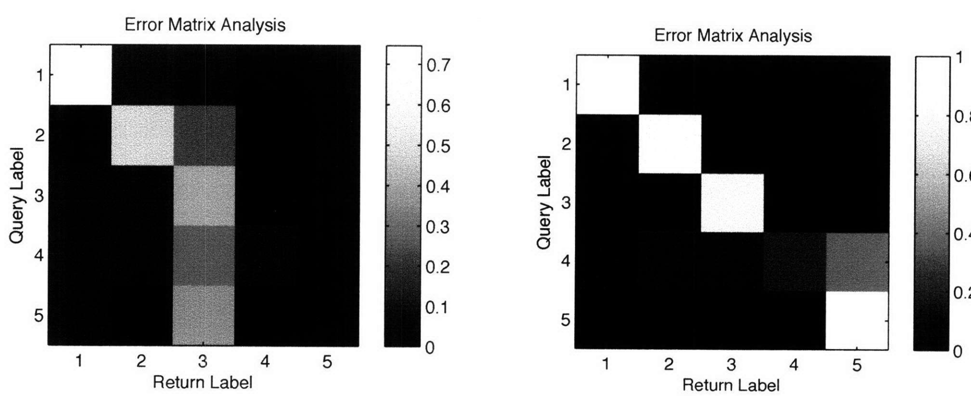

Figure 4-6: Here shows the error matrices on benchmark B using the dynamic term warping system and the vector-space model system. From the results we see that our system is more effective on benchmark B.

4.5 Time Performance Evaluation

We evaluate the time spent during the preprocessing and the querying phase. All systems are implemented in MATLAB and run on a dual-core Xeon 3.4Ghz machine with 8GB memory. Table 4.3 and Table 4.4 summarize the results and Figure 4-7 and Figure 4-8 show the visualization. The results show that the querying phase of the vector-space model system is as fast as the binary feature system, and is about 100 to 400 times faster than the dynamic time warping system. The difference shown in the preprocessing phase is less significant.

	DTW	BF	VSM	Feature	Clustering	Histogram
Benchmark A	16.13	15.51	54.44	16.13	36.65	1.66
Benchmark B	1.31	1.32	6.26	1.31	4.70	0.25

Table 4.3: Preprocessing time evaluations in minutes. The last three columns are the segmented costs of the vector-space model system. All values are in minutes.

	DTW	BF	VSM
Benchmark A	1438.89	2.60	3.74
Benchmark B	34.70	0.26	0.31

Table 4.4: Averaged querying time in seconds. Both the vector-space model system and the binary feature systems query 100 to 400 times faster than the dynamic time warping system.

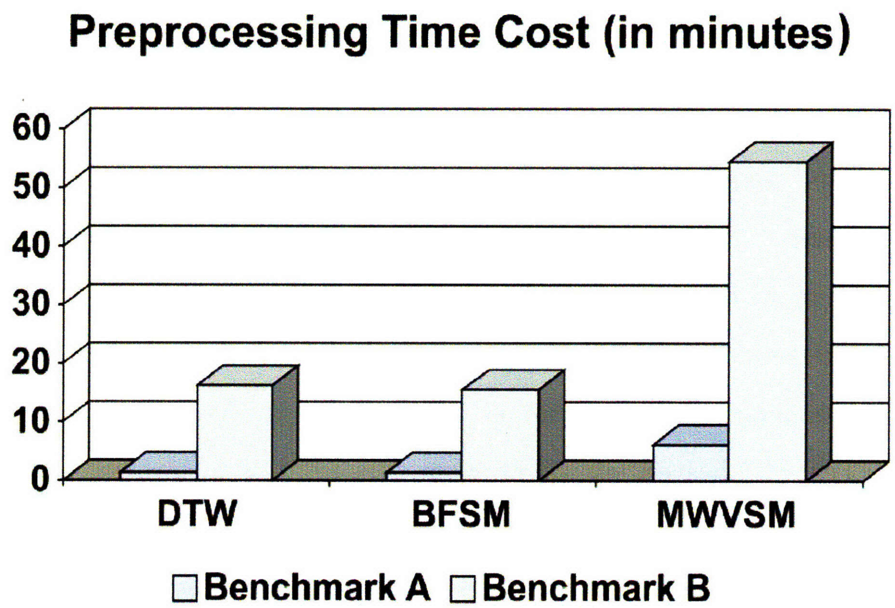

Figure 4-7: This figure shows the preprocessing time costs. It shows that the vector-space model system uses roughly 3 times more preprocessing time than the other two systems.

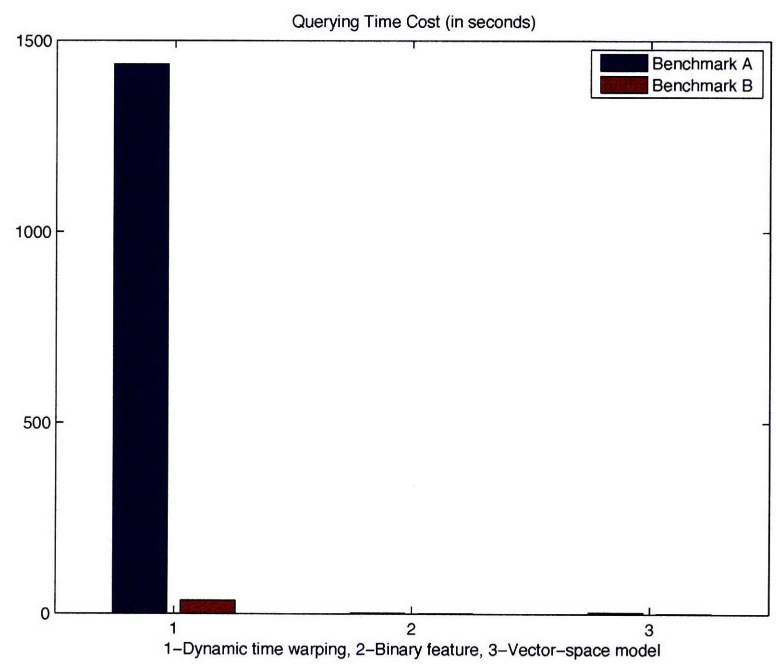

Figure 4-8: This figure shows the querying time costs. The time cost of binary feature system and vector-space model system are low and barely visible in the figure.

4.6 Observation and Discussion

Overall, we found the vector-space model system to be both effective and efficient, while the dynamic time warping system is not efficient and the binary feature system is not effective. Table 4.2 summarizes such observations. This leads us to conclude that our approach is better than the other two. The binary feature system is not effective enough partly because the binarizing process discards too much information. The resulting binary features are coarse. Although such discreteness allows the system to retrieve using fast string matching, the effectiveness is reduced. In our system, the different path we take, i.e. letting the system discretize motions using a learned vocabulary, is more effective.

	Relatively Effective	Relatively Efficient
Dynamic time warping	Yes	No
Binary feature	No	Yes
Vector-space model	Yes	Yes

Table 4.5: This table summarizes the observations from the experiments results. It shows that the vector-space model system is overall a better system as it is more effective than the binary feature system and more efficient than the dynamic time warping system.

Chapter 5

Conclusion

Data-driven motion production is changing the way people create animation. Its promise of inexpensive and rich production makes it a promising future production model. This promise, however, is undermined by the the limitation of motion retrieval techniques. This paper proposed a motion retrieval framework that is both accurate and effective, making it more practical to create motions from a captured database. The core of the problem demands two somewhat contradicting goals: an "as sophisticated as possible" similarity model to capture human's recognition of motions, and an "as high as possible" retrieval speed. These two goals are reconciled by a tool we proposed called motion words. It is an adaptive and discrete guess at the basis of human motion, upon which motions are effectively captured by vectors in high dimensional space. We implemented a system with this concept, and compared it with two competing methods on two real world datasets. The results shows that our method is the most desirable among the three because it is the only one that is both effective and fast.

The new approach applies three pieces from three different fields together to form an effective framework. These three pieces are relational features, adaptive vocabulary generation, and a vector-space model. Relational features compute the relations between parts of a skeleton, excluding irrelevant transformation informations. Adaptive vocabulary generation clusters features in real space, creating a vocabulary that is adapted to the distribution of features in the database. A vector-space model maps every motion into a vector in histogram space, and effectively measures the similarities among motions by their vector orientations. For every given query example, the system maps it into the vector space, and then computes its similarities to the motions in the database, and finally return the most similar ones.

We implemented the system along with the dynamic time warping system [26] and the binary feature system [22] for comparison. All three systems are tested on two benchmark databases consisting of real world motion capture data. Three evaluations are performed: the precision-recall evaluation, the error matrix evaluation, and the performance evaluation. The precision-recall evaluation shows that our systems are in most cases the most effective, and in some cases comparable to the dynamic time warping system. The precision-recall results also shows that velocity information does not necessarily improves the retrieval effectiveness. The error matrix evaluation shows that out system are more likely to make mistakes in two cases that are hard for humans. The performance evaluation shows that both our system and the binary feature system are orders of magnitude faster than the dynamic time warping system. Overall, we observed that our system is comparably as effective as the dynamic time warping system, and its efficiency are comparable to the binary feature system. This makes it the most practical one among the three.

Our system is more effective than the binary feature system because it does not binarize the features. The motion vocabulary approach we take, in contrast, adapts to the distribution of features and creates a much finer discretization. It is more practical because the vector space model is not sensitive to noise, thus excluding the need for a manually selected fuzzy mask. It is fast because it discretize motions into bags of words, which can be searched efficiently as text. We observe that our system requires more preprocessing time. Such cost, however, is not a big problem because the database only needs to be preprocessed once, and the preprocessing time is within an acceptable range.

5.1 Future Work

The success of our approach points to several additional improvements. These include determining the best set of features, experimenting on large datasets especially on the generalization ability of our approach, alternative vocabulary organizations, and the wavelet model. In the following we discuss each of them:

5.1.1 Determining The Best Set of Features

Our experiments reveals that the velocity features improve the relevance of retrieved results, but more effort should be dedicated to determining the best set of features for motion retrieval. It might be fruitful to look into how velocity information changes the internal system behavior. It might be true that some motions should be mainly recognized by there pose, while other require the velocities. If this is the case, then it is necessary to design a system that can automatically decide whether to include velocity information for a particular query example.

5.1.2 Large Dataset and Generalization Ability

More experiments should be performed on large databases and, in particular, to test how this performance generalizes on test motions outside the original database. In this work the motion vocabulary is learned from the database, and might not generalize well to the query motion. It might also be possible to design an incremental learning algorithm that retrains the motion vocabulary with a modest cost every time a query example is given.

5.1.3 Alternative Organization of Vocabulary

The alternative vocabulary organizations, such as vocabulary trees and random forests, should be investigated because they can improve our approach by providing more flexibility to the similarity scoring model and reducing the approximation error of the vocabulary. In this work the system uses a flatly organized vocabulary. A vocabulary tree, in contrast, retains the hierarchical structure in hierarchical k-means, providing more flexibility in the similarity scoring model. A random forest is another possible vocabulary organization for our framework. It creates several vocabularies to reduce the approximation error of the discretization.

5.1.4 Wavelet

Because the bag of words model ignores the temporal ordering of key features, a wavelet model should also be explored as a mechanism to encode features across different time scales. One such example of temporal coherence is the periodic movements in motions. Human motion exhibits great periodicity in some cases, such as walking and running. If such information can be captured by the features, then the representation of the motions would be more compact and the system would retrieve more effectively. A wavelet model is one promising candidate model to capture such frequential-temporal information. It gives rich and highly structured information about the motion that is better at handling motion semantics.

5.1.5 Long term goals

The long term fundamental goals of motion retrieval are always to improve effectiveness and efficiency. Improving effectiveness demands a more sophisticated statistical model, a more sophisticated query model, and more data. Improving efficiency demands more structured organization of data, and tight integration with hardware advancement such as multi-core architectures. It is likely that these two will be bundled together by some new technique similar to motion words. The technique will need to deeply capture the semantic mapping process and still maintain its compactness for scalable search.

Another long term direction is the integration of motion search as part of an motion creation system. Such a system would integrate motion capture, key-frame sketching, motion storage and search, and motion generation. It could boost motion effects productivity to unprecedented levels. The motion search module will not only search the database fast and accurately, but also be integrated with the authoring interface. One possibility is that it could take query input from a sketch interface and summarize the retrieved data in a ready to use sequence.

Appendix A

Material Included For Completeness

A.1 Relational Feature Computation Example

Here is an example of relational feature computation. The follow math commands checks how far the right hand is to the front:

$$
\begin{aligned}
\mathbf{n} = &\quad (c_{\mathrm{T}} - c_{\mathrm{RS}}) \times (c_{\mathrm{LS}} - c_{\mathrm{RS}}) \\
d = &\quad \frac{\mathbf{n}}{\| \mathbf{n} \|} \cdot (c_{\mathrm{RH}} - c_{\mathrm{RS}})
\end{aligned}
$$

, where c_{T}, c_{RS}, c_{LS}, c_{RH} are the locations of the thorax, the right shoulder, the left shoulder, and the right hand respectively. $\mathbf{n}$ computes the normal of the plane determined by the thorax and the two shoulders. We dot product it with the vector from the right shoulder to the right hand, $c_{\mathrm{RH}} - c_{\mathrm{RS}}$, getting how far the right hand is to the front. Following commands binarizes this feature. It compares the distance with unit length 2, and set the feature accordingly. It checks if the right hand is to the front for more than 2 unit length.

$$
y = \quad \begin{cases} 0 & d \leq 2 \\ 1 & d > 2 \end{cases}
$$

A.2 Searching with Inverted Index

The problem in the lookup stage is that we need to compute similarity with every motion in the database. This poses a $O(n)$ complexity query time. Inverted index solves this problem by building an index from word to motions. In particular, we have an index $I(w)$ for every word w, that stores all motions that contain w.

During query, we only need to consider those motions that have an overlap with the example. We achieve this by computing the union of inverted indices of all words in the query example:

$$l = \bigcup_{w \in \mathbf{x}} I(w)$$

, where $\mathbf{b}$ is the query example. Then instead of adding the example to the database, we compute its similarity with every motion in l, and return the ones with the greatest similarities sorted descendingly on similarities. This inverted index essentially computes the list of motions that are non-orthogonal to the example in the word space.

Without using inverted index, the complexity of retrieval is $O(n + n \log n)$, where $O(n)$ is the time to compute the example's similarity to every motion in the database, and $O(n \log n)$ is the time to sort these similarities. Using inverted index, only motions that have words in common with — or non-orthogonal in histogram space with — the example are processed. We call this list of non-orthogonal motions non-orthogonal list.

An inverted index based query consists of three steps: get the non-orthogonal list, compute their similarities to the example, sort. Assume the example has p different words, then p lists need to be union-ed to get the non-orthogonal list. Assume the longest list among the p list has q motions. Then this union operation takes $O(pq)$ time.

Suppose we apply stop list approach and remove all words that existed in more than ϕ motions, then q is upper bounded by ϕ. Thus the non-orthogonal list is $O(p)$ long, which leads to $O(p)$ time of similarity computing and $O(p \log p)$ time of sorting, totalling $(p \log p)$. This is much better than $O(n \log n)$ because it is not relevant to the size of the database, but to the number of words in the example.

A.3 Review of Wavelet Filter

Wavelet filter gives localized frequency responses in any resolution. By convolving a signal with a translated and scaled wavelet function, we can get frequency response of the signal in a particular time range and a particular frequency range. Here we convolve every column of the feature sequence with such wavelet function:

$$X = D \otimes \frac{1}{\sqrt{\alpha}} \psi(\frac{t-\beta}{\alpha})$$

, where ψ is the original wavelet function, α is the scaling factor and β is the translating factor. This equation gives the convolution of every column of D with a translated and scaled wavelet. The translation controls the temporal location, and the scaling controls the frequency of the resulting response. More specifically, this gives us the frequency response in the frequency band $[1/\alpha, 2/\alpha]$ and at the location β.

A.3.1 An Example

Here is an example to show the effectiveness of wavelet. As mentioned in the article, one example of temporal coherence is the periodic movements. Figure A-1 shows a plot of the measure of how far the left foot is to the front in a running motion; the plot reveals periodic changes of the signal. Figure A-2 shows the Fourier transform of this signal, revealing a peak in the frequency domain.

Figure A-3 shows the continuous wavelet transform of the signal in Figure A-1. In contrast to the original signal and the frequency domain representation, wavelet model gives a frequential-temporal representation that combines the two.

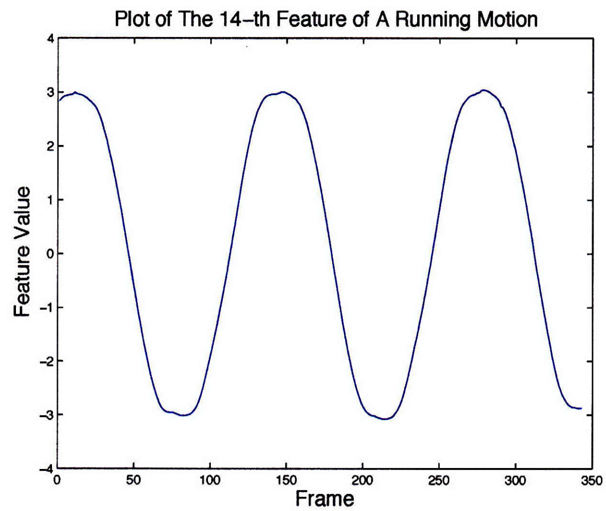

Figure A-1: This is a plot of the measure of how far the left foot is to the front in a running motion over time. The signal shows periodic changes and suggests that the system should exploit the temporal ordering of frames.

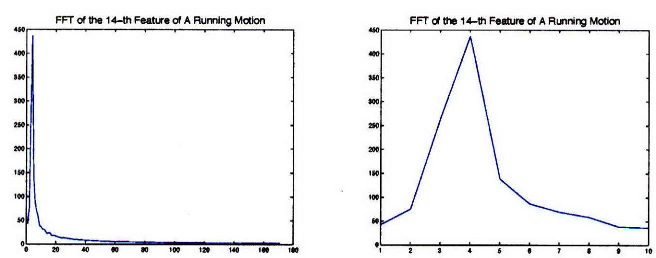

Figure A-2: Here shows the Fast Fourier Transform results on the example signal from Figure A-1. The left figure shows the frequency domain response of the signal. The signal has a major low frequency peak. The right figure is the magnified version of the left over the horizontal axis.

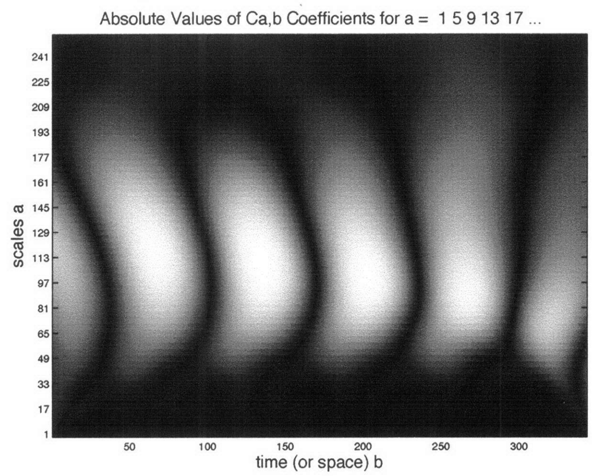

Figure A-3: This is a plot of the wavelet transform of the example signal in Figure A-1; The wavelet filter used is Daubechies 4 filter. In contrast to the temporal signal in Figure A-1 and the frequential responses in Figure A-2, this figure is a combination of the two: the horizontal axis is the time axis and the vertical axis corresponds to period; the higher on the vertical axis, the longer the period the response stands for.

Appendix B

Source Code

B.1 Dynamic Time Warping System

B.1.1 Dynamic Time Warping Code For The Similarity Between Two Motions

```
function [r] = f(c1,c2);
% dynamic time warping distance of two motions, averaged distance
n1 = size(c1,2);
n2 = size(c2,2);

ef = zeros(n1+2,n2+2);   % the frame to frame distance
for i=1:n1,
for j=1:n2,
  ef(i+1,j+1) = norm(c1(:,i)-c2(:,j));
end
end

g=zeros(n1+2,n2+2);
step=zeros(n1+2,n2+2);
for i=2:n1+2,
```

```
for j=2:n2+2,
  tmp=g(i-1,j-1);

  if (j<n2+2 || step(i-1,j)>=n2)
    tmp=min(tmp,g(i-1,j));
  end
  if (i<n1+2 || step(i,j-1)>=n1)
    tmp=min(tmp,g(i,j-1));
  end

  g(i,j)=tmp+ef(i,j);
  stepinc = (i<=n1+1 && j<=n2+1);
  if tmp==g(i-1,j),
     step(i,j)=step(i-1,j)+stepinc;
  elseif tmp==g(i-1,j-1),
     step(i,j)=step(i-1,j-1)+stepinc;
  else
     step(i,j)=step(i,j-1)+stepinc;
  end
end
end

r=g(n1+2,n2+2)/step(n1+2,n2+2);  % averaging
end
```

B.2 Binary Feature-based String Matching

It is an implementation of features in: Mueller et. al. 2005, "Efficient content-based retrieval of motion capture data".

B.2.1 Relational Features Computing

```
function [nbf,hlshw] = f(r3,frate),
% r3: motion clip, every frame is a list of skeleton joints in 3D cartesian space
% frate: framerate
% nbf: non-binary features
% hlshw: humerus length, shoulder width, and hip width;

  n = size(r3,3);
  assert(size(r3,2)==14 && size(r3,1)==3);
  % this version is for 14 joints skeleton; Benchmark A use this skeleton
  % benchmark B uses a skeleton of 31 joints, a subset of 14 joints of that skeletc

  proot=1;
  pneck=2;
  plhip=3;  prhip=4;
  plknee=5;  prknee=6;
  plfoot=7;  prfoot=8;
  plshoulder=9;  prshoulder=10;
  plelbow=11;  prelbow=12;
  plhand=13;  prhand=14;

  if ~exist('frate'),
    frate=30;
  end
  framerate = frate;

d1 = diff(r3,1,3);
d1(:,:,n) = d1(:,:,n-1);   % last frame has the same d1 as second to last frame
d1=d1*framerate;
```

```
nbf=[]; hlshw=[];

for i=1:n,
  vs=r3(:,:,i);

  root=vs(:,1);  % joint locations
  neck=vs(:,2);
  lhip=vs(:,3);  rhip=vs(:,4);
  lknee=vs(:,5);  rknee=vs(:,6);
  lfoot=vs(:,7);  rfoot=vs(:,8);
  lshoulder=vs(:,9);  rshoulder=vs(:,10);
  lelbow=vs(:,11);  relbow=vs(:,12);
  lhand=vs(:,13);  rhand=vs(:,14);

  hlshw (1,i) = norm(lshoulder-lelbow);     % humerus length
  hlshw (2,i) = norm(lshoulder-rshoulder);
  hlshw (3,i) = norm(lhip-rhip);

  % upper body features
  nbf(1,i) = flet_plane ([rshoulder,root,lshoulder, lhand]);
  nbf(2,i) = flet_plane ([rshoulder,root,lshoulder, rhand]);
  nbf(3,i) = flet_nplane ([root,neck,neck, lhand]);
  nbf(4,i) = flet_nplane ([root,neck,neck, rhand]);
  nbf(5,i) = flet_nplane ([rshoulder,lshoulder,lshoulder, lhand]);
  nbf(6,i) = flet_nplane ([lshoulder,rshoulder,rshoulder, rhand]);
  nbf(7,i) = flet_cosine ([lshoulder,lelbow,lhand]);
  nbf(8,i) = flet_cosine ([rshoulder,relbow,rhand]);
  nbf(9,i) = norm( d1(:,plhand,i));
  nbf(10,i) = norm( d1(:,prhand,i));
  nbf(11,i) = (lshoulder-rshoulder)' * (rhand-lhand) / norm(lshoulder-rshoulder) /
```

```
  nbf(12,i) = norm(lhand-rhand);

  % lower body
  legs = [lhip,lknee,lknee,lfoot,rhip,rknee,rknee,rfoot];
  nbf(13,i) = flet_segsp_distance( [legs,lhand]);
  nbf(14,i) = flet_segsp_distance( [legs,rhand]);
  nbf(15,i) = norm(lhand-neck);
  nbf(16,i) = norm(rhand-neck);
  nbf(17,i) = flet_segp_distance( [lhip,rhip,lhand]);
  nbf(18,i) = flet_segp_distance( [lhip,rhip,rhand]);
  mknee = (lknee+rknee)/2;
  nbf(19,i) = flet_cosine( [neck,root,mknee]);
  nbf(20,i) = norm( d1(:,proot,i));


  nbf(21,i) = flet_plane( [lhip,rhip,rfoot,lfoot]);
  nbf(22,i) = flet_plane( [lhip,rhip,lfoot,rfoot]);
  nbf(23,i) = flet_nplane( [mknee,root,rfoot,lfoot]);
  nbf(24,i) = flet_nplane( [mknee,root,lfoot,rfoot]);
  nbf(25,i) = norm( d1(:,plfoot,i));
  nbf(26,i) = norm( d1(:,prfoot,i));
  nbf(27,i) = flet_cosine( [lhip,lknee,lfoot]);
  nbf(28,i) = flet_cosine( [rhip,rknee,rfoot]);
  nbf(29,i) = flet_nplane( [rhip,lhip,lhip,lfoot]);
  nbf(30,i) = flet_nplane( [lhip,rhip,rhip,rfoot]);
  nbf(31,i) = (lhip-rhip)' * (rfoot-lfoot) / norm(lhip-rhip) / norm(lfoot-rfoot);
end
end
```

B.2.2 Converter From Non-binary Feature To Binary Feature

```
function [bf] = nbf2binf(nbf,hlshw),

n = size(nbf,2);
bf=[];

hl = hlshw(1,:);
sw = hlshw(2,:);
hw = hlshw(3,:);
cos120 = cos(pi*2/3);
cos120s = ones(1,n)*cos120;

thresh = [hl;hl;  hl*0.2;hl*0.2;  hl;hl;  cos120s;cos120s; ...
          hl*2.3;hl*2.3; zeros(1,n)];
bf(1:11,:) = nbf(1:11,:) > thresh;

thresh = [hl*0.3; hl*0.5;hl*0.5; hl*0.5;hl*0.5; hl*0.5;hl*0.5];
bf(12:18,:) = nbf(12:18,:) < thresh;

thresh = [cos120s; hl*2.1; hl*0.3;hl*0.3;  hl;hl;  hw*3;hw*3; ...
          cos120s;cos120s;  hw;hw; zeros(1,n)];
bf(19:31,:) = nbf(19:31,:) > thresh;

a = [];
for i=1:n,     % encode into a long int
  tmp = int32(0);
  for j=1:size(bf,1),
    tmp=tmp+bf(j,i)*2^(j-1);
```

```
  end
  a(i)=tmp;
end

bf=a;
end
```

B.2.3 Generate Return Lists From Binary Features

```
function [rls] = bf2rl(bf,cl,masks),
% bf 2 retrieval lists
%   function [rls] = bf2rl(bf,cl,masks),
if ~exist('masks'),  % fuzzy retreival mask
  masks=[2^32-1];
end

clstart = [];       % clip start and end frame
for i=1:length(cl),
  clstart(i) = sum(cl(1:i-1))+1;
end
size(clstart)
clending = clstart+cl'-1;
rls = {};   % retrieval list to be returned

for k=1:length(masks),  % try every masks
  art = zeros(1,length(bf));
  mk = masks(k);
  for i=1:length(bf),
    art(i) = bitand(mk,bf(i));
  end
```

```
% find segment ending and starting, in terms of frame
segending = sort(unique( [find(diff(art)~=0) clending]));
segstart = sort(unique( [find(diff(art)~=0)+1 clstart]));
sl = segending-segstart+1;   % segment length

% find clip length in terms of segments
scl = [];
p = 1;
for i=1:length(cl),
  sp = p;
  ct = sl(p);
  while ct<cl(i),
    p=p+1;
    ct=ct+sl(p);
  end
  assert(ct==cl(i));

  scl = [scl p-sp+1];  % clip length in segs
  p=p+1;
end
assert(sum(scl)==length(sl));
arte = art(segstart);

% clip no. of each segment
clno = zeros(1,length(arte));
cs = 1;
for i=1:length(scl),
  clno(cs:cs+scl(i))=i;
  cs = cs + scl(i);
end
```

```
  size(clno);

  %
  rl = {};
  for e = 1:length(scl),   % try every motions as the query example
    ea = arte( sum(scl(1:e-1))+1:sum(scl(1:e)));
    lea = length(ea);

    caca = find(arte==ea(lea));   % intersecting inverted files
    for i=fliplr(1:lea-1),
      ce = caca-1;
      ce = ce(find(ce>=1));
      caca = ce( find(arte(ce)==ea(i)));
    end

    cls = unique(clno(caca));
    cls = cls(find(cls~=e));  % remove the example from returned list
    rl{e} = cls;  % return list for querying motion e
end

rls{k}=rl;  % return lists by searching using mask k
end
end
```

B.3 Motion Word-based Vector Space Model

Features in this system is taken from the Binary Feature-based String Matching system with modifications.

B.3.1 Computing Features

```
% select some realtion features and pad them with
% their 1st derivatives
function [dnd] = nbf2dnd(nbf,hlshw,frate),
% dnd: distance and derivatives features
% nbf: non-binary features
% hlshw: humerus length, shoulder and hip width,
%        for normalization
% frate: frame rate

  t =[1 2 3 4 5 6 12 13 14 15 16 17 18 21 22 23 24 29 30];
  % subset of features we use

  tx=[1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1  3  3];
  % which length should every feature be normalized:
  % 1-humerus length, 2-shoulder width, 3-hip width

  nbfp=[];
  for j=1:length(t),  % normalizing
    nbfp(j,:)= nbf(t(j),:) ./ hlshw(tx(j),:);
  end

  % approximating 1st order derivative
  d1=diff(nbfp,1,2)*frate;
  n=size(nbfp,2);
  d1(:,n)=d1(:,n-1);
  dnd=[nbfp;d1];
end
```

B.3.2 Clustering

We use the free "fastkmeans" software package came along with the paper: Elkan 2003, "Using the triangle inequality to accelerate kMeans". We downloaded the code from http://www-cse.ucsd.edu/ elkan/fastkmeans.html.

Wraper Of fastkmeans With Random Initial Seeds

```
% fastkemans with random seeds
function [idx,centers] = mykmeansfast(data,k,runs),
% data: feature vectors
% k   : number of clusters
% runs: how many times to run
% idx : index of nearest center
% centers: cluster centroids

mind=1e100;
n=size(data,1);
if k>n,
  disp('k>n in kmeans');
end

for i=1:runs,
  while true,
    tmp = randint(1,k,[1 n]);
    if length(unique(tmp))==k,  % found k unique initial seeds
      break;
    end
  end

  [c,ii] = fastkmeans(data,data(tmp,:),2);
```

```
  d = sumdist_clustering(data,ii,c);
  if d<mind, % select the best
    mind=d;
    centers =c;
    idx = ii;
  end
end
end
```

Hierarchical K-means Code

```
function [tr,nwordnum] = hkm(a,k,depth,nextwordno),
% A recursive function for hierarhical clustering
%  a: feature vectors
%  k: branching factor
%  deapth: current tree depth
%  nextwordno: the counted id of next word added
%  tr: vocabulary tree
%  nwordnum  : the counted id of next word added

if depth==0   % leaf
  tr = struct('wordno',nextwordno);
  nwordnum = 1;
  return;
end

% centers is K-by-P matrix, row major
[idx,centers] = mykmeansfast(a',k,10);

tmp=[]; for i=1:k, tmp(i)=length(find(idx==i)); end;
```

```
% cluster the children
cs=cell(1,k);
nwordnum = 0;
for i=1:k;
  ac=a(:,find(idx==i));
  [cs{i},tmp] = hkm( ac,k,depth-1,nextwordno+nwordnum);
  nwordnum = nwordnum+tmp;
end

tr = struct('cs',{cs},'cetrs',centers');
end
```

B.3.3 Converting Motions Into Words

```
function [art] = nbf2art(nbf,voc,
% nbf: features vectors
% voc: motion vocabulary
% art: motion word articles

n=size(nbf,2);
art=zeros(1,n);
for i=1:n,
    % convert a(:,i) into words   ; a int32
    sample=nbf(:,i);

    cnode = voc;
    while isfield(cnode,'cs')==1,
    % search for the leaf that contains the feature vector ''sample''
dists=[];
for j=1:size(cnode.cetrs, 2),
dists = [dists norm(sample-cnode.cetrs(:,j))];
```

```
end
cnode = cnode.cs{ find(dists==min(dists)) };
    end

    wordno = cnode.wordno;
    art(i)=wordno;
end
end
```

B.3.4 Using Vector Space Model

Obtaining Term-frequency Matrix

```
% make TF matrix
% art: article of words, nfx1
% cl:  clip lens, nx1 mat
% return tf matrix:  tf(i,j) word j in document i;
function [tf] = mktfmatrix_gen (wn, art, cl),

tf = zeros(size(cl,1), wn);
size (tf);

% make tf matrix
p = 1;
for i=1:size(cl,1),   % every clip
for j=1:cl(i),
tf(i, art(p)) = tf(i, art(p))+1;
% article i has word art(p), art(p) is a word in article i
p=p+1;
end
end
```

```
for i=1:size(cl,1),
tf(i,:) = tf(i,:) / cl(i);
end

end
```

Obtaining Inverse-document-frequency Vector

```
% make idf vector
function [idf] = mkidf(tf),

size(tf);
D = size(tf,1);

nW = size(tf,2);
idf=zeros(nW,1);

for i=1:nW,
tmp = size(find(tf(:,i)~=0));
tmp = prod(tmp);
if tmp==0,
idf(i) = 0;    % useless word
continue;
end

idf(i) = log(D/tmp);
end

end
```

Obtaining TF-IDF Weights Document Vectors

```
% tv to tfidf weighted tv
% function [titv] = t2t (tf,idf),
%  titv(i,j):  doc i word j
%   NO TV anymore
function [titv] = t2t (tf,idf),
wn=size(tf,2);
n=size(tf,1);
assert(wn==length(idf));
titv=zeros(n,wn);

for i=1:size(tf,1),
  titv(i,1:wn) = tf(i,:) .* idf';
  % normalize it
  titv(i,1:wn) = titv(i,1:wn) / norm(titv(i,1:wn));
end
end
```

From TF-IDF Vectors To Similarity Matrix

```
% tv to similarity matrix by L2 norm
% function [smcs] = sm (tv),
%
function [smcs] = sm (tv),
n=size(tv,1);
smcs=[];
for i=1:n,
for j=1:n,
  smcs(i,j)= tv(i,:)*tv(j,:)' /norm(tv(i,:)) /norm(tv(j,:));
end
```

```
end

end
```

From Similarity Matrix to Return Lists

```
% similarity matrix to retrieval matrix
function [rm] = f(sm),
rm={};
n=size(sm,1);
for i=1:n,
  ws = sm(i,:);
  ws = sortrows([ws' (1:n)'],[-1]);
  rm{i} = ws(:,2)';
  rm{i} = rm{i} (find(rm{i}~=i));
end
end
```

Bibliography

[1] Okan Arikan and David A. Forsyth. Interactive motion generation from examples. *ACM Transactions on Graphics*, 21(3):483–490, July 2002.

[2] Okan Arikan, David A. Forsyth, and James F. O'Brien. Motion synthesis from annotations. *ACM Transactions on Graphics*, 22(3):402–408, July 2003.

[3] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval.* ACM Press / Addison-Wesley, 1999.

[4] Stefan Carlsson. Combinatorial geometry for shape representation and indexing. In *Object Representation in Computer Vision*, pages 53–78, 1996.

[5] Chih-Yi Chiu, Shih-Pin Chao, Ming-Yang Wu, Shi-Nine Yang, and Hsin-Chih Lin. Content-based retrieval for human motion data. *J. Visual Communication and Image Representation*, 15(3):446–466, 2004.

[6] K. Forbes and E. Fiume. 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, 2005. ACM.

[7] W. T. Freeman and E. H. Adelson. The design and use of steerable filters. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(9):891–906, 1991.

[8] Kristen Grauman and Trevor Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision*, pages 1458–1465, Washington, DC, USA, 2005. IEEE Computer Society.

[9] Kristen Grauman and Trevor Darrell. Approximate correspondences in high dimensions. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 505–512. MIT Press, Cambridge, MA, 2007.

[10] C. Harris and M. Stephens. A combined corner and edge detection. In *Proceedings of The Fourth Alvey Vision Conference*, pages 147–151, 1988.

[11] Eugene Hsu, Sommer Gentry, and Jovan Popović. Example-based control of human motion. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 69–77, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[12] Eamonn Keogh, Themistoklis Palpanas, Victor B. Zordan, Dimitrios Gunopulos, and Marc Cardle. Indexing large human-motion databases. In *vldb'2004: Proceedings of the Thirtieth international conference on Very large data bases*, pages 780–791. VLDB Endowment, 2004.

[13] Lucas Kovar and Michael Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics*, 23(3):559–568, August 2004. In Press.

[14] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Transactions on Graphics*, 21(3):473–482, July 2002.

[15] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[16] CMU Graphics Lab. Cmu graphics lab motion capture database.

[17] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. *ACM Trans. Graph.*, 21(3):491–500, 2002.

[18] Yi Lin. Efficient human motion retrieval in large databases. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 31–37, New York, NY, USA, 2006. ACM.

[19] Feng Liu, Yueting Zhuang, Fei Wu, and Yunhe Pan. 3d motion retrieval with motion index tree. *Comput. Vis. Image Underst.*, 92(2-3):265–284, 2003.

[20] Guodong Liu, Jingdan Zhang, Wei Wang, and Leonard McMillan. A system for analyzing and indexing human-motion databases. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 924–926, New York, NY, USA, 2005. ACM.

[21] David G. Lowe. Object recognition from local scale-invariant features. pages 1150–1157, 1999.

[22] Meinard Müller, Tido Röder, and Michael Clausen. Efficient content-based retrieval of motion capture data. *ACM Transactions on Graphics*, 24(3):677–685, August 2005.

[23] Juan Niebles, Hongcheng Wang, and Li Fei-Fei. Unsupervised learning of human action categories using spatial-temporal words. *International Journal of Computer Vision*.

[24] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. pages II: 2161–2168, 2006.

[25] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[26] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of speech recognition.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[27] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

[28] Gerard Salton, Edward A. Fox, and Harry Wu. Extended boolean information retrieval. *Commun. ACM*, 26(11):1022–1036, 1983.

[29] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 1470, Washington, DC, USA, 2003. IEEE Computer Society.