國立交通大學

多媒體工程研究所

碩 士 論 文

以少量的動作感知器來驅動角色動作之研究

Character Animation Driven by Sparse Motion Sensors

研 究 生：劉峻豪

指導教授：林奕成 博士

中 華 民 國 九 十 九 年 七 月

以少量的動作感知器來驅動角色動作之研究

Character Animation Driven by Sparse Motion Sensors

研 究 生： 劉峻豪　　Student： Chun-Hao Liu

指導教授： 林奕成博士　　Advisor： Dr. I-Chen Lin

國 立 交 通 大 學

多 媒 體 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institutes of Multimedia Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年七月

以少量的動作感知器來驅動角色動作之研究

學生：劉峻豪　　指導教授：林奕成博士

國立交通大學

多媒體工程研究所

摘要

動作捕捉系統提供了一個高品質且準確的技術，來驅動虛擬角色的動作。然而，昂貴的價格與繁雜的後續處理過程，使得此一技術難以普遍運用在互動式的環境中。在本論文中，我們提出一個利用少量的感知器裝置，即可驅動角色動作的方法。首先，我們要求使用者配戴少量的感知器，並且跟隨範例動作舞動，以取得一動作主題於感知器之間的特性。其次，動作資料會切成數個片段，並且以動作圖的資料結構，連接各個平順的片段動作。然後，透過隱馬爾可夫模型來計算一動作片段至另一片段的機率，我們發現此動作重建的方法，可以更加真實及可靠。最後，為了增加動作的變化性，我們更進一步的混合類似使用者舞動的數個片段，而資料庫則無此一還原動作片段。因此，在即時的互動介面環境下，使用者可利用少量且廉價的動作感知器裝置，以驅動虛擬角色的動作。此外，該簡易之系統裝置，更可運用於各類高級互動之介面系統。

關鍵字：電腦動畫，互動介面，WII 控制器，感知器。

Character Animation Driven by Sparse Motion Sensors

Student: Chun-Hao Liu **Advisor: Dr. I-Chen Lin**

Institute of Multimedia Engineering

National Chiao Tung University

Abstract

Motion capture devices provide an accurate and high definition technique to generate avatar's motion from reality. Nevertheless, the expensive cost and tedious post-processing make it difficult to gain the popularity and to perform on interactive applications. In this thesis, we propose a data-driven approach to drive character animation by sparse motion sensors. To acquire the motion characteristics of a subject (player), we ask a user to follow exemplar motion and record the acceleration and angular velocity from sparse motion sensors. The motion data are then divided into several clips, and we construct the action graph to connect each smooth pair of clips. While applying hidden Markov Model to calculate the probability of transition from one clip to the other, it shows that the reconstructed motion is much more realistic and reliable. Finally, to extend the motion variety, we farther blend clips that are similar to user's action for playing motion unseen in the database. In real-time, the user is capable of driving avatar's motion by inexpensive sensors. In addition, such an easy-to-use system can also be applied to advanced interaction systems.

Keywords: Computer animation, interactive interfaces, Wii remotes, sensors.

Acknowledgement

光陰似箭，在研究所的兩年學習中，有歡笑、淚水和努力。於此，由衷要感謝的人就是指導教授，林奕成教授。無論在學習上遇到困難、在研究上遇到瓶頸，或是在人生旅途中遇到挫折時，都是林教授在旁細心指導與諄諄教誨，而得以克服萬難及增強自信與勇氣，以面對未來的挑戰。兩年來的成長，即是我進入研究所最大的收穫。

在背後默默關心的父母和親人，是我最大的避風港。每當遇到困頓或寂寞時，家人的鼓勵，就成為最重要的精神支柱，或許他們無法直接在專業領域上給予協助，但總能從關切中得到力量，而持續產生奮發向前的動力。

最後，也要感謝所有的指導師長，以及自始至終強力支持的朋友和同學。有了大家的相伴，才會有屬於今日的我。為了表達內心至高無上的謝意，畢業後將時時期許自己，以專業知能回饋於社會，而做為一個對社會和國家具有貢獻的交大人。

Content

List of Figures

List of Tables

1. Introduction

1.1 Motivation

In recent years, the computer generated characters can be seen in lots of interesting regions, such as cartoons, movies, games or virtual environments. Nevertheless, the most popular technique to drive motion of characters is through motion capture device, which is tedious to setup, high-cost and time-consuming. To solve this problem, there have been many researches that discuss about this topic, such as using camera to catch silhouette, or wearing Wii remotes to receive acceleration data and predict human motions.

However, most existing methods used motion recognition but regards less on reasonable motion transition. Taking a baseball player as an example, most of the batter will perform three actions on the field, which are stance, swinging bat, and stretchy. Most of methods by motion recognition will reconstruct this sequence of swinging bat well with data sensors. But if he suddenly detects that the coming ball is in the strike zone when the batter stops swinging suddenly, this immediate transition between swinging and stopping will not be considered in most of recognizing methods. Besides, existing methods mostly recognize and playback motion in database, and limit the variety of character motion.

In this thesis, we propose another a novel approach to capture human motion from sparse off-the-shelf motion sensors. In our system, we use Wii remotes with MotionPlus as our input, where accelerometer and the gyroscope inside provide us motion signals to reconstruct human motion in real-time. The proposed motion estimation system provides high performance and accuracy of reconstructing smooth human motion in real-time, so that it will be adequate to various interactive applications.

Figure 1.1: Wii remotes with MotionPlus. [Nintendo]

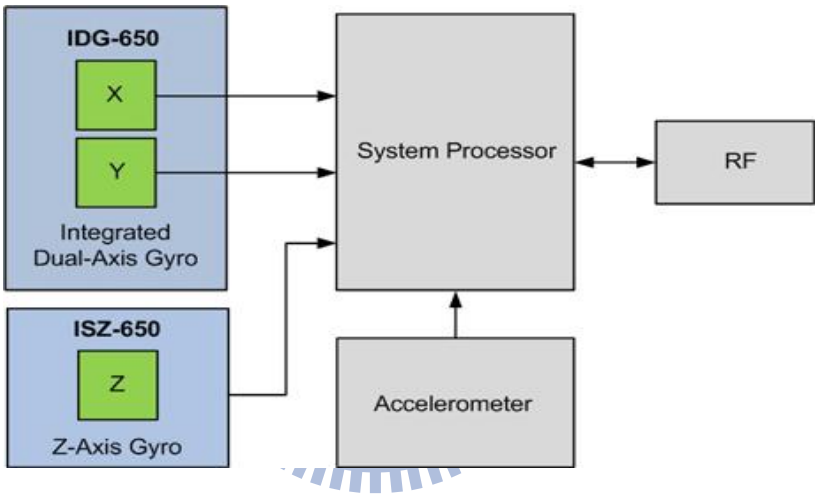

Figure 1.2: System diagram of motion sensors. [Invensense]

1.2 Background

A famous game corporation, Nintendo, has developed a new TV game console, Wii, in the late 2006. Meanwhile, they also introduced an innovative controller, Wii remote, also called Wiimote, which contains not only buttons but the IR sensor and three-axis accelerometer. The accelerometer captures net force acting on Wiimote in the range from -3*g* to 3*g*, where *g* is gravitational acceleration.

In June 2009, an expansion device for Wii remote, MotionPlus, was released for

capturing the complex motion with more degree of accuracy. With three-axis gyroscope, MotionPlus can receive the local angular velocity in pitch, roll and yaw rotations. In other words, the data of local rotation derived by Wii remote is much more precise.

Since the Wiimote and MotionPlus off-the-shelf are not expensive, in this research, we put one to five such devices on subject's limbs, and propose an efficient algorithm to reconstruct subject's motion from sensing data.

Two critical issues in MotionPlus are initialization and error accumulation. In order to fix initial error, we should put MotionPlus on a flat surface and the system will automatically detect the background noise of MotionPlus, and revise the new coming value every time. To fix accumulated the pitch and roll rotation can be calibrated by the relative direction of gravity. Since there is no relationship between the yaw rotation and gravity (yaw rotation is horizontal to the plane), in most Wii games, they used the sensor bar position in IR camera for yaw calibration. In other words, the yaw rotation is set to zero whenever Wii remotes pointing toward the sensor bar.

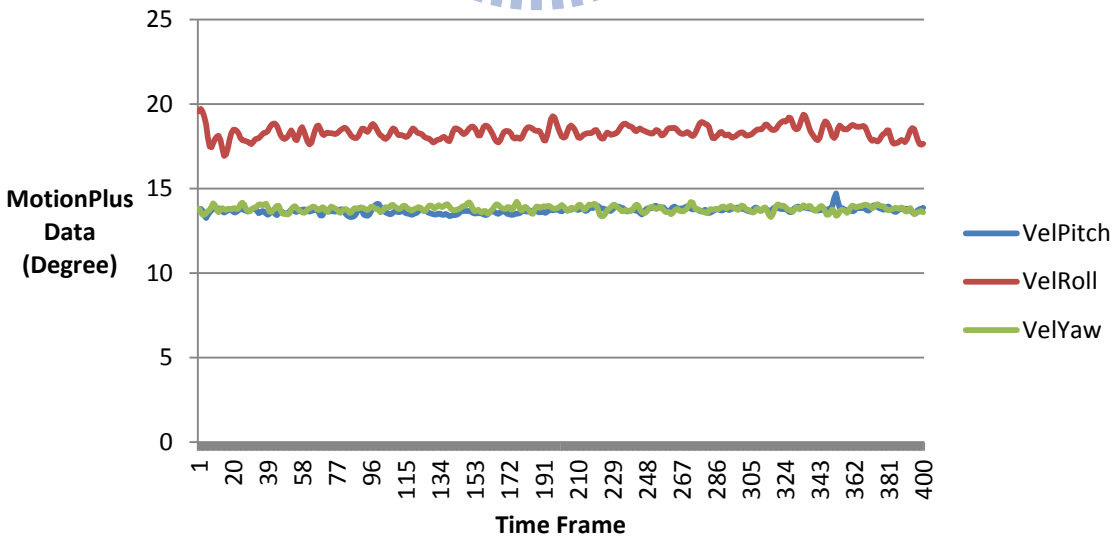

Figure 1.3: The velocity value of pitch, roll and yaw rotations before fixing the background noise of MotionPlus

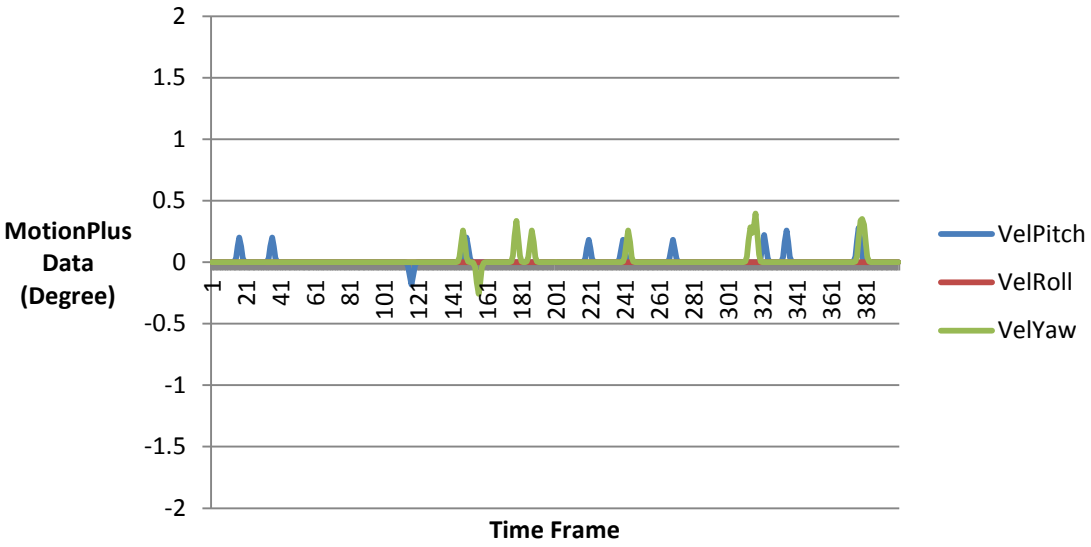

Figure 1.4: The velocity value of pitch, roll and yaw rotations after fixing the background noise of MotionPlus

1.3 Frameworks

In this thesis, we propose a data-driven method to reconstruct human motion in real-time. First of all, we use motion capture data from CMU MoCap Lab. To associate the data for motion sensors with actual motion we require, we acquire users to wear Wii remotes with MotionPlus and ask them to follow the motion playback on the screen. To avoid sensors' data are asynchronous with user's action, we divide training motion into several short segments, it makes user easier to follow up and make our motion alignment more accurate (Sec. 3.1). Besides, the training motion is divided into several training motion clips for later reconstruction (Sec. 3.2). Then, we use principal component analysis (PCA) to reduce the dimensionality of sensors' data (Sec. 3.3).

In the second stage, we perform the k-mean method to cluster training motion clips into several groups, relying on the reduced-dimensionality data by PCA (Sec 3.4). For the

performance issue, rather than calculating the difference between input data from Wii remotes to each clusters, we utilize implementing linear discriminant analysis (LDA) for dimensionality reduction before later classification (Sec. 3.5). Afterward, we construct an action graph to connect each smooth pair of training motion clips, depending on the total Euclidean distance between the positions of one's end frame to the position of other's start frame (Sec. 3.6). It is not necessary that consider the middle frames of clip since a clip is short.

In the third stage, we calculate the transition probability in action graph, and the probabilities from each node in graph to each cluster. With these pre-processed data, we can construct the hidden Markov Model (HMM) as a database for reconstruction (Sec. 3.7). The hidden states are nodes in action graph, and observations are clusters of acceleration and angular velocity. In order to get the maximum probability of the path in transitions, we implement a dynamic programming algorithm, Viterbi algorithm, for finding the most likely sequence of hidden states. This Viterbi path will result in a smooth and reasonable motion.

Finally, in order to produce the motion which is not in the database, we propose performing motion blending methods. Rather than finding a single Viterbi path with the maximum probability, we find more than one path. If their maximal probabilities are close enough, or larger than a user-define threshold, we blend the training motion clips from Viterbi paths (Sec. 3.8).

In real-time motion reconstruction, the user should wear Wii remotes with MotionPlus as the motion sensors (Sec. 3.9). Our system performs in multi-threading styles: one for collecting and processing the data by sensors, and other for rendering the reconstructed human motion.

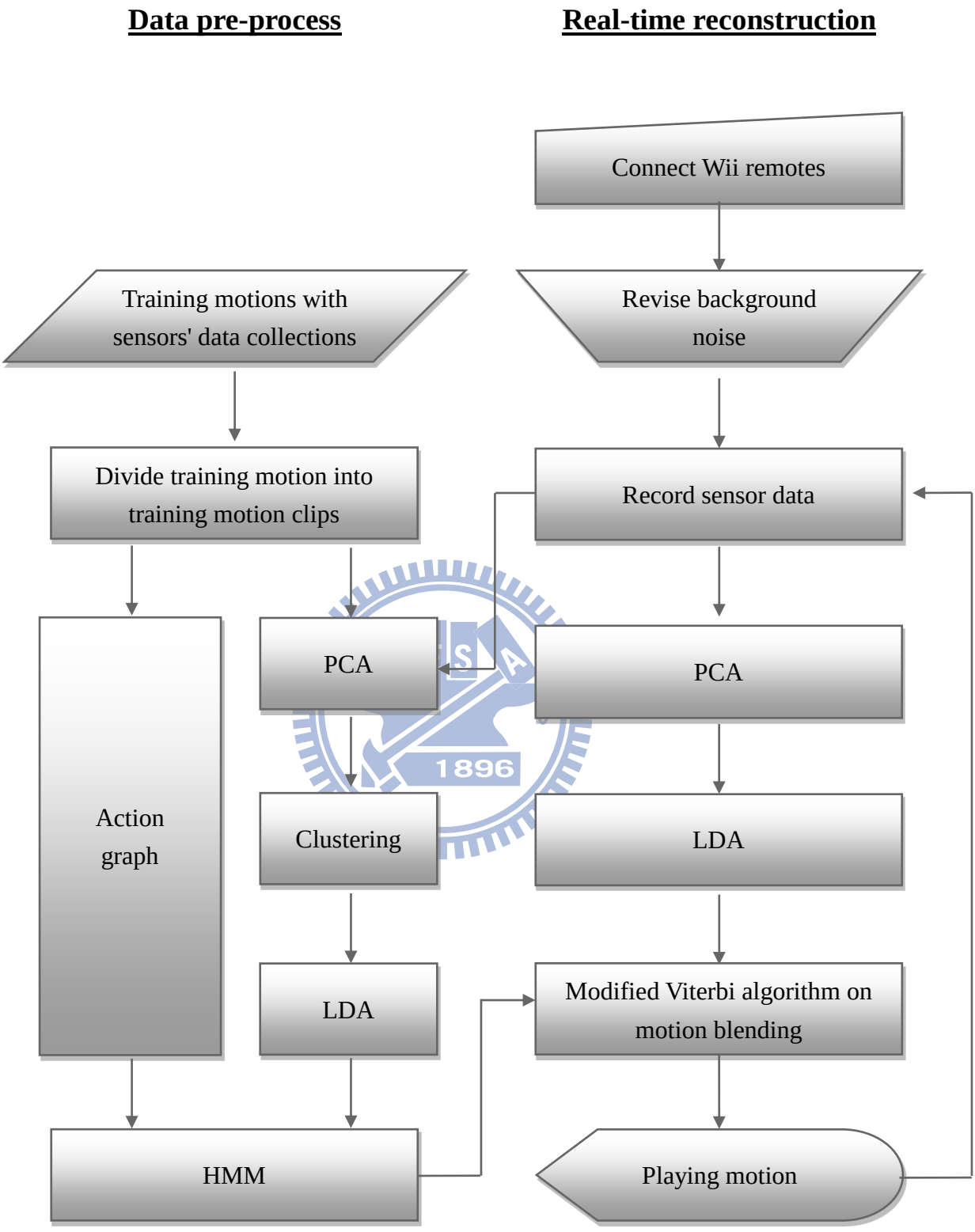

Figure 1.5: Flowchart of our system

2. Related Works

Since the topic of 3D animation becomes popular in modern days, more and more researchers and corporations pay attention to the significance of transforming human motion from reality into digital characters. There are two major technologies to capture human motions. One is employing optical systems, or the motion capture devices, and the other is non-optical systems, which often uses sensors as the substitution.

2.1 Animations by Motion Capture Devices

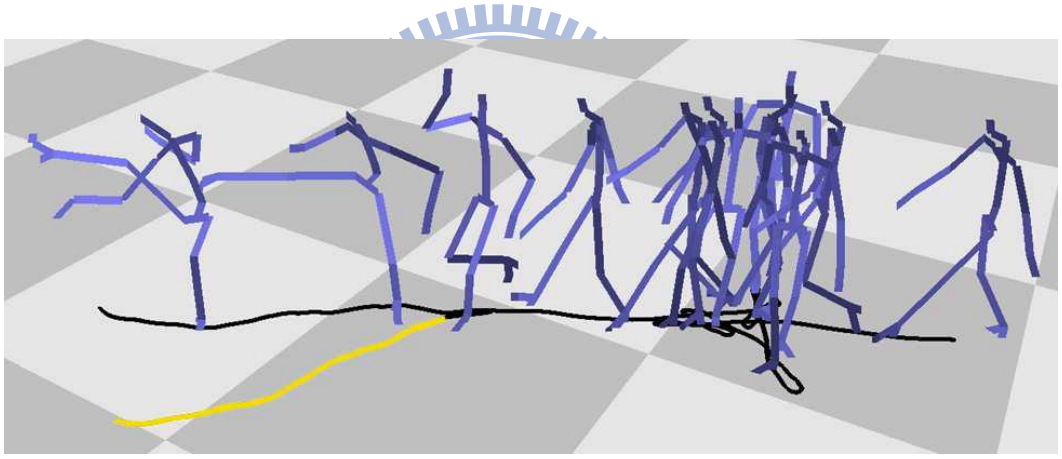

Figure 2.1: An example that character is driven by motion graph. [KGP02]

Kovar et al. [KGP02] proposed a method using the concept of graph structure to play continuous motion clips smoothly, where the closet frames were found to transit between clip pairs. By deliberating upon these supplementary transitions as edges of a directed graph, called motion graph, was automatically constructed by setting the threshold and blend two clips by applying spherical linear interpolation (Slerp). A nodes of graph was the sequence of frames, or what we called the motion clip. To generate continuous motions, the largest strongly connected component (SCC) of the motion graph was the only one can be used.

In this thesis, we take advantage of graph structure, building an action graph to address the problem in motion synthesis. Specifically, when a user drives the character in different categories, such as from running to boxing, our system traverses graph nodes and searches a proper path from running to boxing motion. This path guarantees that the motion clips and transitions are synthesized smoothly.

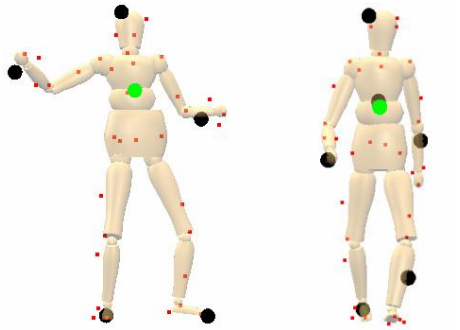

Figure 2.2 The principal markers are shown in black and the estimated markers are shown in red. [LZWM06]

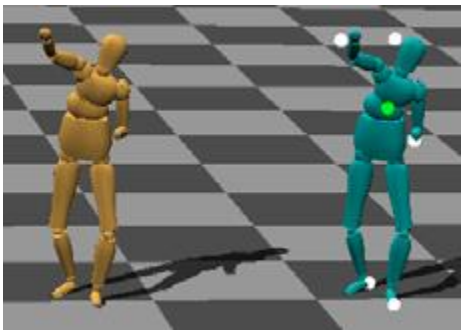

Figure 2.3: The golden model represents the actual pose data. The cyan model shows an estimated pose. [LZWM06]

Liu et al. [LZWM06] proposed that they used an adequate subset of original markers on motion capture were sufficient to predict the whole markers for subtle variations in human motions. The main concept was to build piecewise linear models and search principal markers automatically by off-line data training, and then to construct the three dimensional motion models.

By using position information with a small set of markers, this approach is able to derived the position for a large amount of non-principal markers. Besides, this method is based on a few markers with motion models, instead of considerable markers with motion database. Hence, the less requirements are appropriate to interactive applications.

However, if samples of training data were insufficient, it possibly results in an erroneous estimation given an unseen motion. Furthermore, because it only operates on

position data, which is difficult to be obtained for popularity by off the shelf low-cost devices like Wii remotes. Namely, when a user demands to control the position of a marker or a joint, it is uncontrollable due to none of the position information by reality. In our approach, our system can be applied position data, the acceleration or angular velocity data from Wii remotes.

Figure 2.4: The above pictures are that user wearing markers to control the full-body motion of avatar in from of two synchronized cameras. [CH05]

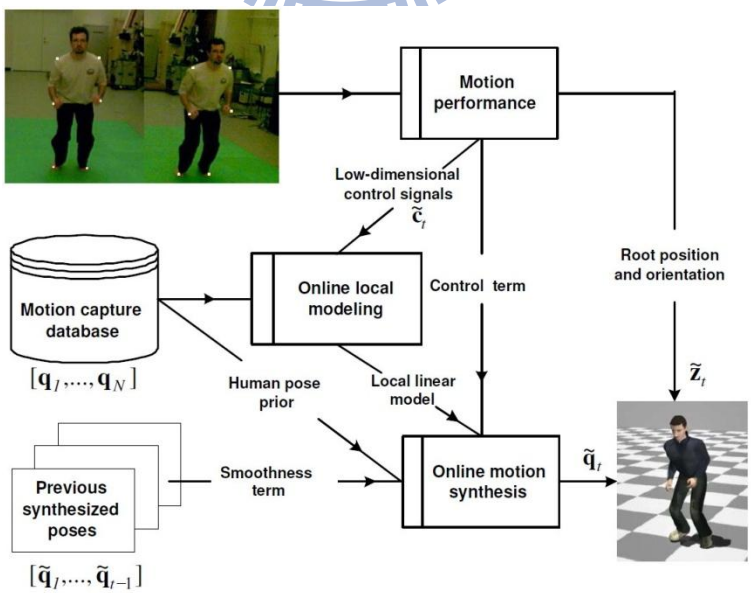

Figure 2.5: System overview. [CH05]

Chai et al. [CH05] introduced to employ two synchronized cameras and a few of retro-reflective markers to reconstruct full-body motion of avatar. The major difference to the [Liu et al. 2006] was that they take the location of the markers, by using video cameras as the input. This system consisted of three significant components, which were motion performance, online local modeling, and online motion synthesis. In other words, this system first extracted positions of markers, which specified the desired trajectories of certain points on the animated character. Then, it searched the closest to signals by neighbor graph in the database. Finally, they synthesized motions to smooth the reconstructed animation of avatar.

This approach performs six motion categories, walking, running, hopping, jumping, boxing, and Kendo (Japanese sword art) from low-dimensional signals in real-time. Besides, since only two cameras and a small number of markers are required, it is not as expensive and cumbersome as the original motion capture devices.

However, this system demands for retro-reflective markers and synchronized cameras to perform motion as input, the issue of occlusion and post-processing make it remained impractical for home usage in recent years. Besides, the performing area is strongly limited if it is used in interactive application.

2.2 Animation with Low-cost Sensors

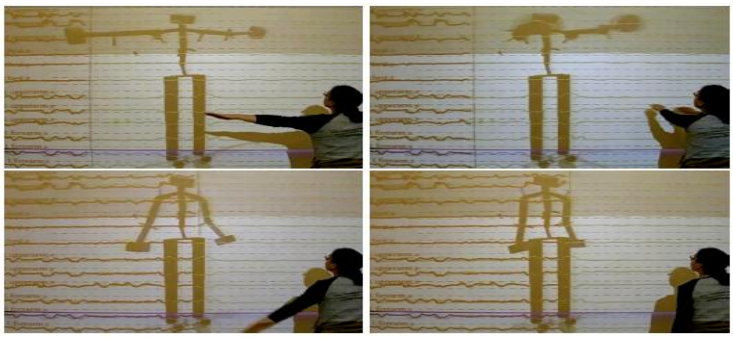

Figure 2.6: Snapshots of a user performing a calibration motion. [SH08b]

Slyper et al. [SH08b] proposed a method to capture human motion based on accelerometers. The accelerometer, ADXL 330, is the same MEMS as the Wii remotes. This approach was to install five ADXL 330 on a shirt, which were on each forearm, upper arm, and one on the chest. By measuring the acceleration value of human motion, they proceeded to compare clips of motion accelerations in the database. Finally, the system played back the corresponding motion clip. The strengths of this approach were simple to implement and often perform accurately in some motions.

Nevertheless, when a user performs a complicated motion, this approach may match the wrong clips unless the training database is sufficient enough. But if the database is too large, the system needs more searching time in the database. Moreover, if the chosen frames of a motion clip is long, the latency will also be long; on the other hand, the transition between two clips will be visually non-fluent. Besides, this method only recognizes the motion on upper body. Once the lower body is taking into account, the system will be spoilt by interferences from more accelerometers, and the accuracy may decrease substantially.

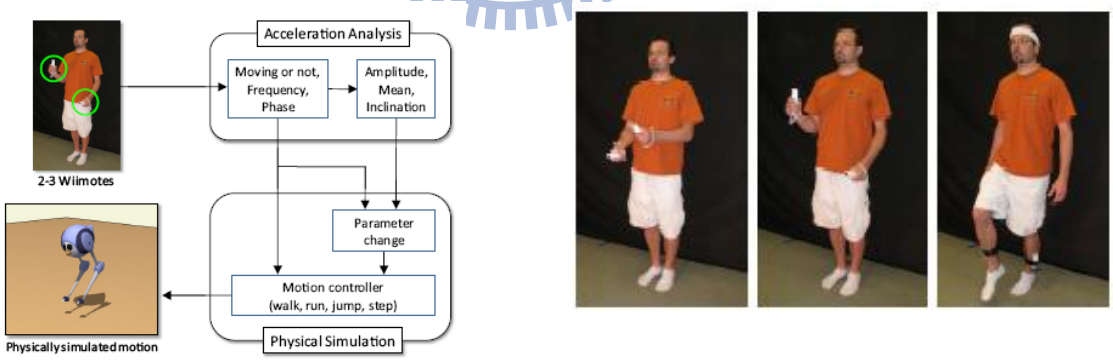

Figure 2.7: A user uses Wii remotes to control a bird character by physically simulation. [SH08a]

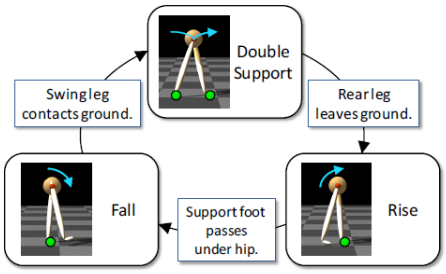

Figure 2.8: A state machine for walking. [SH08a]

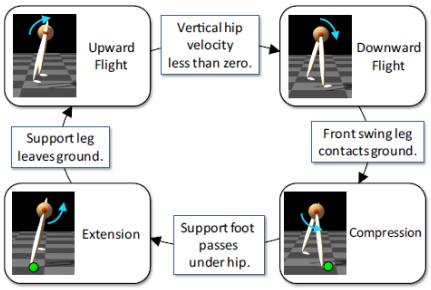

Figure 2.9: A state machine for running. [SH08a]

Shiratori et al. [SH08a] also used Wii remotes attached on user's arms, wrists, or legs to get the accelerations for the control of physically simulated character. In this approach, they formulated the data into features, such as frequency, mean, amplitude, etc. In the real-time, the physical simulation was processed by transforming these features into the corresponding motion. During the user study and analysis, they proved that three of the Wii remotes interfaces provide better control than the traditional joystick interface.

In this physically simulated system, a user controls a bird character by Wii remotes. There are four motions supported in this system, which are walk, run, jump and step. This character is able to arbitrarily change directions and movements (6 uncontrolled degrees of freedom, DOF), with two hips (3 DOFs) and two telescoping joints (1 DOF). Moreover, this system offers the potential for some natural responses to rough terrain and other disturbances, such as tumbling over or trembling.

But with the limitation on a few of acceleration information, there remains the ambiguity problem because of unobvious motions. Additionally, if there are more complex motions are desired except these four motions, there should be more complicated physically theories to support them.

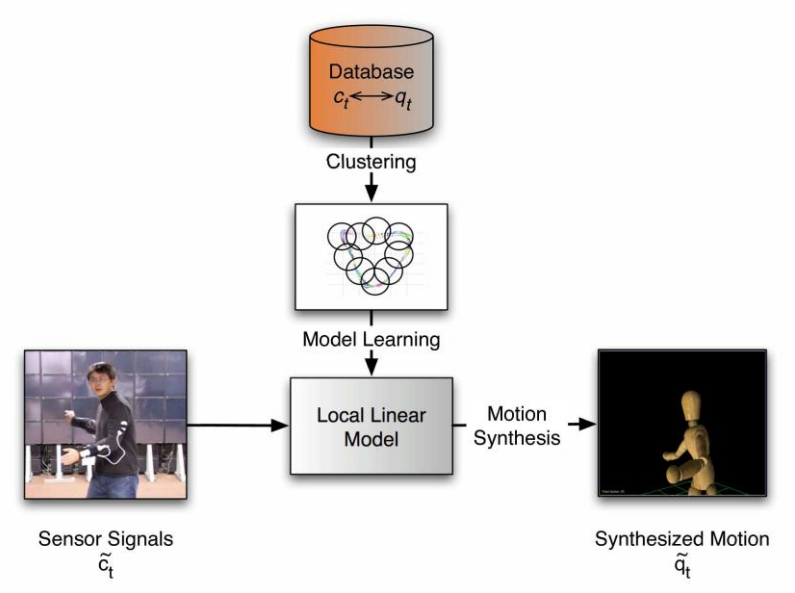

Figure 2.10: Motion query and synthesis. [XKCG*08]

Xie et al. [XKCG*08] used a data-driven approach that operated the optical motion capture while attached four Wii remotes to a performer's body, for obtaining the frame-to-frame acceleration mapping. They reduced the dimensionality of acceleration data by implementing principal component analysis (PCA), and built a local linear model by applying radial basis function (RBF). After the data pre-preprocessing, the system could drive the character by database from input sensor data.

This proposed system can be used for various applications. Besides, since the dimensionality is reduced, this design will be scalable and is capable of large database without degradation. By a sufficient number of training examples, it is able to synthesize a great deal of disparities on human motions.

However, because the framework of this approach only concentrates on motion estimation, it suffers from the smoothness problem due to the noise in the sensor data, or different motion clusters synthesize together, such that the visual discontinuity of motion may happen.

Figure 2.11: Calibration Setup. [SH09]

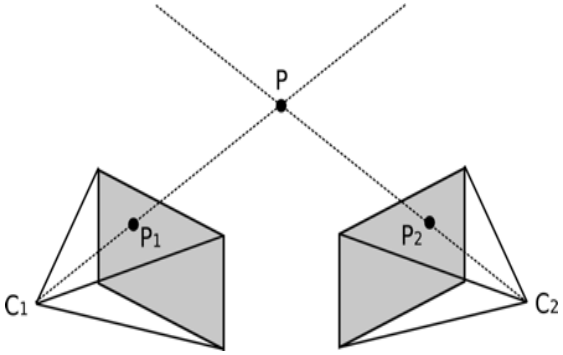

Figure 2.12: Triangulation. [SH09]

Scherfgen et al. [SH09] used two Wii remotes performing as cameras. Rather than applying accelerometers, they preferred to take advantages of IR sensors insides Wii remotes to do the tracking process. They calibrated Wii remotes' cameras by a calibration board with infrared reflectors and illuminated groups of them, following by applying triangulation, which was to track a 3D point position P by observing 2D point positions P1 and P2.

This approach allows an user to interact in a more intuitive way instead of just pressing buttons or applying the acceleration data into motion based on data-driven modeling. However, because of the triangulation dependent on targeting the ray at the same point in 3D space, when rays do not intersect due to the limited camera resolution, there will be numerical inaccuracy and the processing errors in the point detection.

3. Driving Character using Motion Sensors

3.1 Training Data Collection

At first, we use motion capture data as training motion from CMU MoCap Lab. To avoid device dependant bias of sensor data, we acquire users to wear Wii remotes with MotionPlus on two wrists and legs, and ask them to follow the motion playing on the screen in order to associate the training data for motion sensors with actual motion. However, the directly retrieved Wii data often falls behind the training motion, since we often perform the action after we have seen the motion on the screen. Namely, it is not synchronized.

We propose two methods to solve this problem without using motion capture device. One is to use a "pause" button on Wii remote, which is a function key stopping the motion on the screen and waiting for the user's response. This method can reduce most asynchronous problems on simple motions, but not the complex one. The other one is to divide training motion into several short segments, and we record the training sensor data with time warping using a finite state machine (FSM) controller. From our experimental result, the later one makes our motion alignment more accurate.

There are five states in the controller, including two pause states, a playing state, a recording state, and a time warping state. In the beginning, a user presses button *one* to waiting state, and playing the motion by pressing button *two*. Whenever the user plans to divide training motion into a segment, it goes to waiting state by pressing button *two* and it causes the motion stopping on the screen. Then by pressing button two again, the user should perform like the motion segment that he has seen on the screen; meanwhile, the acceleration and angular velocity from Wii controllers are recorded into buffer. When the user finishes his action, pressing button *two* to time warping state and the controller goes to waiting state

when it finished. In the time warping state, we simply perform linearly warping, because the motion segments are short and it is computationally efficient. The procedure is iteratively performed until the end frame of playback motion. Finally, the data will be stored in the database after we apply a low pass filter.

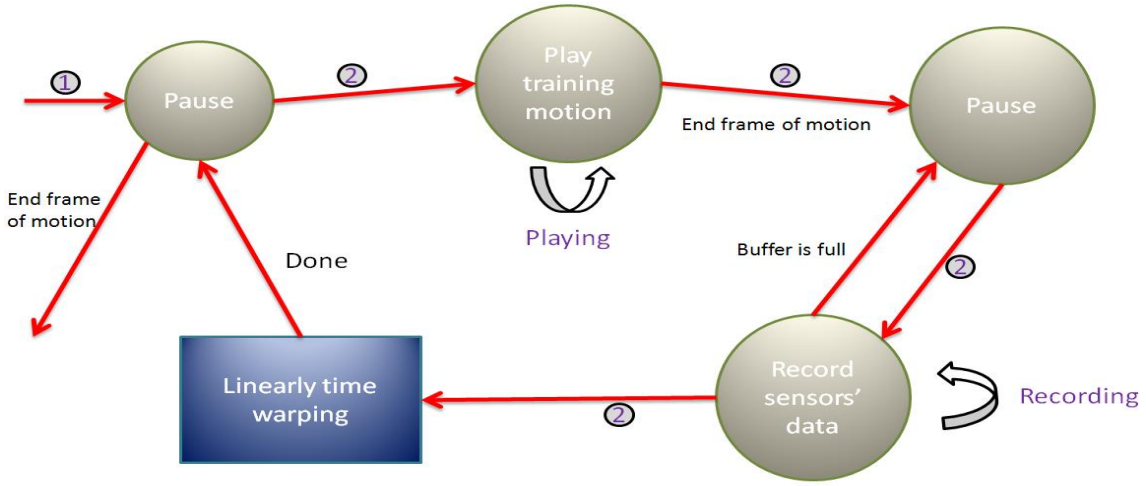

Figure 3.1: The finite state machine of motion playing and the Wii remotes data record.

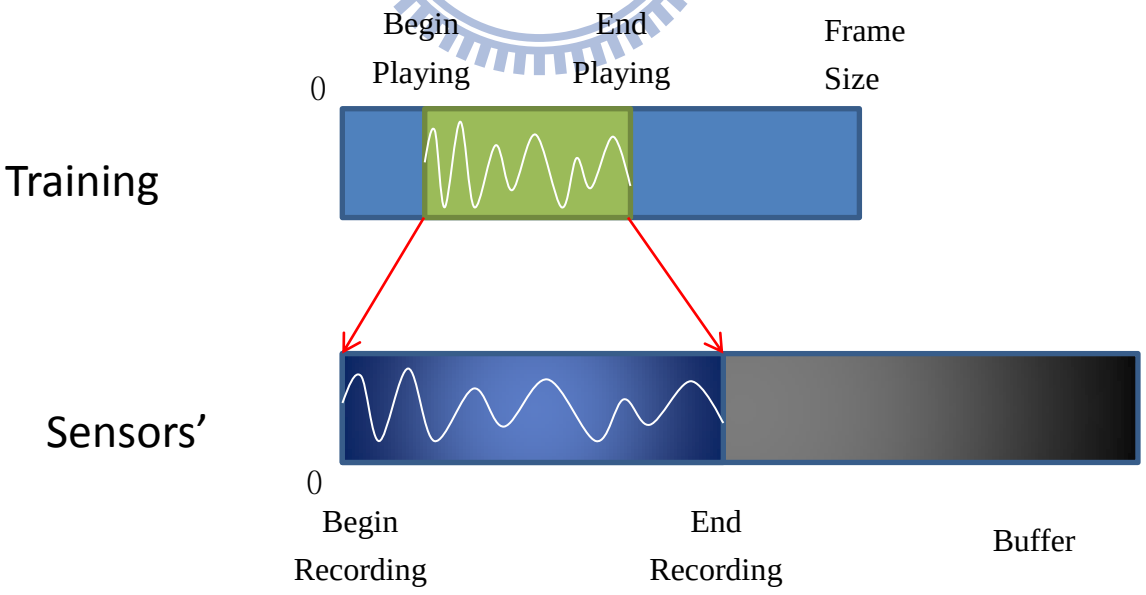

Figure 3.2: An example of linearly time warping.

3.2 Motion Clips

In this stage, we are going to divide the training motion into several parts, or so called training motion clips in this thesis. A training motion clip is formally a sequence of frames, and associates with information between the displayed avatar and Wii sensors, such as joints' angles, skeleton's position, and training sensor data. This information will be utilized to drive the avatar for novel motions. There are two major methods to divide motion into clips.

The first method is to simply divide the whole training motion data into *n* parts, where *n* is a user-defined number. But this method will not take the sequence between two middle of clips into consideration. In other words, when the system is trying to reconstruct human motion from middle of i_{th} clip to middle of $(i + 1)_{th}$ clip, it selects either one of them. However, whether what the result is, it is not expected.

The second method is rather than dividing into n parts, we take the overlay frames between two clips into account. Namely if there are *k* frames per clip, we divide the training motion starting from $(k / 2) \times i$, where $i = 1, 2, 3, \cdots$. Therefore, sequential frames between each pair can be considered and making the reconstructed motion acts more stable and flexible. In our approach, we prefer using the second method in the implementation.

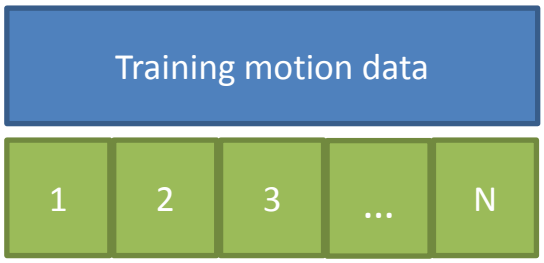

Figure 3.3: Motion clips without overlay frames.

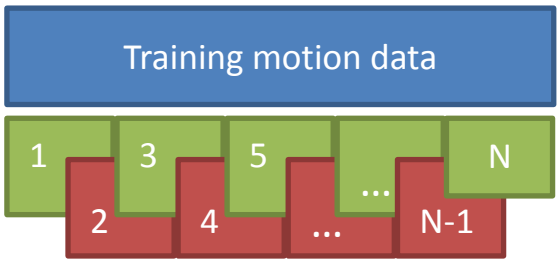

Figure 3.4: Motion clips with overlay frames.

3.3 Principal Component Analysis

After collecting the training or online received data from motion sensors, we need to maintain two matrices, an matrix of acceleration, and a matrix of angular velocity. The size of matrices are $(C \cdot W \cdot D \cdot 3) \times N$, where C is the number of data training collections, W is the number of motion sensors, e.g. accelerometers and gyroscopes, D is the number of frames in a motion clips, 3 is for three dimensional axes, and N is the number of clips in training motion. However, the dimension of matrix grows larger as the C, W or D increases. Consequently, the huge dimension results in poor performance on reconstructing motion.

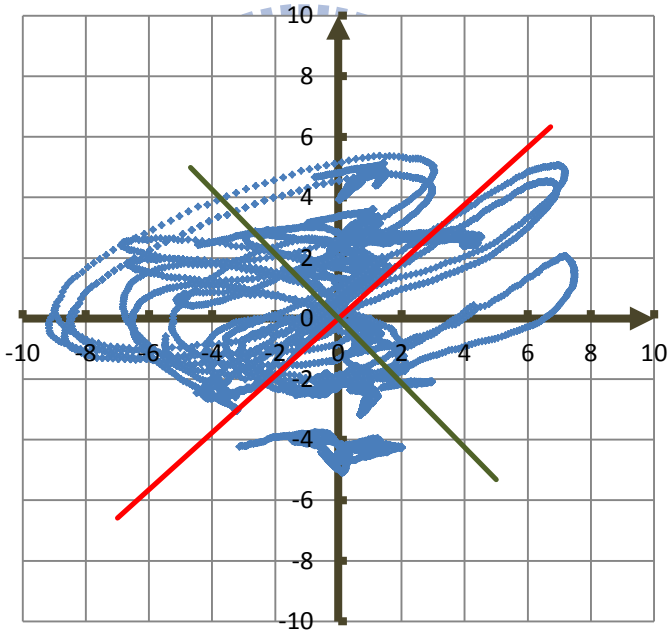

Figure 3.5: PCA of a two dimensional sample data.

In order to efficiently execute matrix operation in real-time, the process of dimensional reduction is required. Therefore, we propose using principal component analysis (PCA) in advance. PCA is a eigenvector-based multivariate analysis, which turns plenty of possibly correlated data into a few of uncorrelated, or what we called principal components. These

components should also be responsible for the remaining variability; thus the reduced-dimensionality data persists for the significant features from the given huge information.

The major concept of PCA is an orthogonal linear transformation, which projecting the given data into new coordinates system. The projection is dependent on the variance of data, such that the first component contains the greatest variance, second component contains the second variance, etc. Therefore, when the enough number of principle components is extracted, they are able to stand for most of features in the given data. In our case, we reduce the dimensionality from 90 to 7. since it captures 99% of the motion variance.

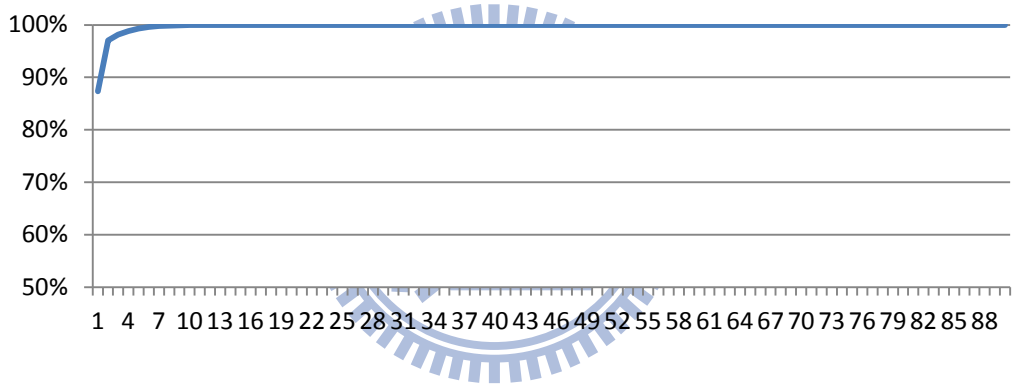

Figure 3.6: Percentage of variance explained by the principal components of angular velocity data set.

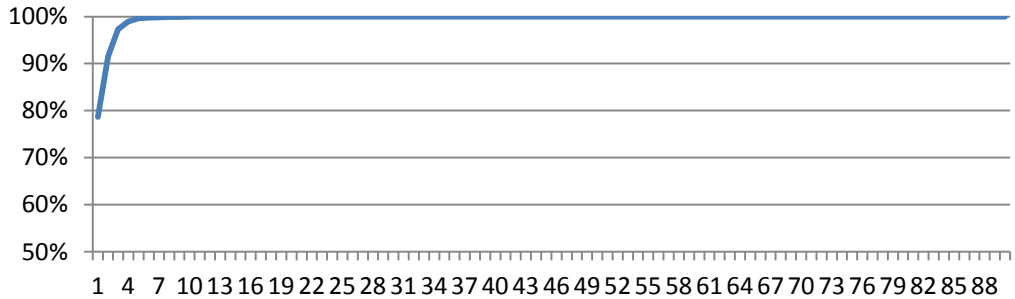

Figure 3.7: Percentage of variance explained by the principal components of acceleration data set.

To apply PCA in our sensors' data, we prepare two matrices for the training or online received data on acceleration and angular velocity by motion sensors. Since the procedures of PCA on angular velocity and acceleration are equivalent, we explain only the procedures of PCA on acceleration and denoting the data matrix as X of dimensions $M \times N$, where $M = (C \cdot W \cdot D \cdot 3)$ and N is the number of clips in training motion, C is the number of data training collections, W is the number of motion sensors, D is the number of frames in a clips, 3 is for three dimensional axes. We calculate the mean vector on each column w of dimension $M \times 1$.

$$w[m] = \frac{\sum_{n=1}^{N} X[m, n]}{N}, \qquad m = 1, 2, 3, \dots, M \tag{1}$$

Then, we produce a row vector t of dimension $1 \times N$, whose elements are all 1's. We subtract mean vector w from X to derive the new matrix X' of dimension M $\times$ N, whose mean is zero.

$$X' = X - wt \tag{2}$$

We calculate the covariance matrix C of dimension $M \times M$ from the outer product of matrix X' with itself. The E is the expected value operator.

$$C = E[X' \times X'] \tag{3}$$

Finally, we find the eigenvectors and eigenvalues of the covariance matrix C. The matrix of eigenvectors is denoted as V, and the D is the diagonal matrix that $D[m, m]$ is the m_{th} eigenvalue of C, where $1 \leq m \leq M$.

$$D = V^{-1}CV \tag{4}$$

When applying PCA, we can choose components and forming feature vectors by eigenvectors with highest eigenvalues. The result of reduced-dimensional data r_i^j is produced by the following equation:

$$r_i^j = (v_i^j - \overline{v_j})A_j^{-1} \tag{5}$$

where $\overline{v_j}$ is the mean value of the clip j, A_j^{-1} is the inverse matrix built from the eigenvectors V corresponding to the largest eigenvalues from C.

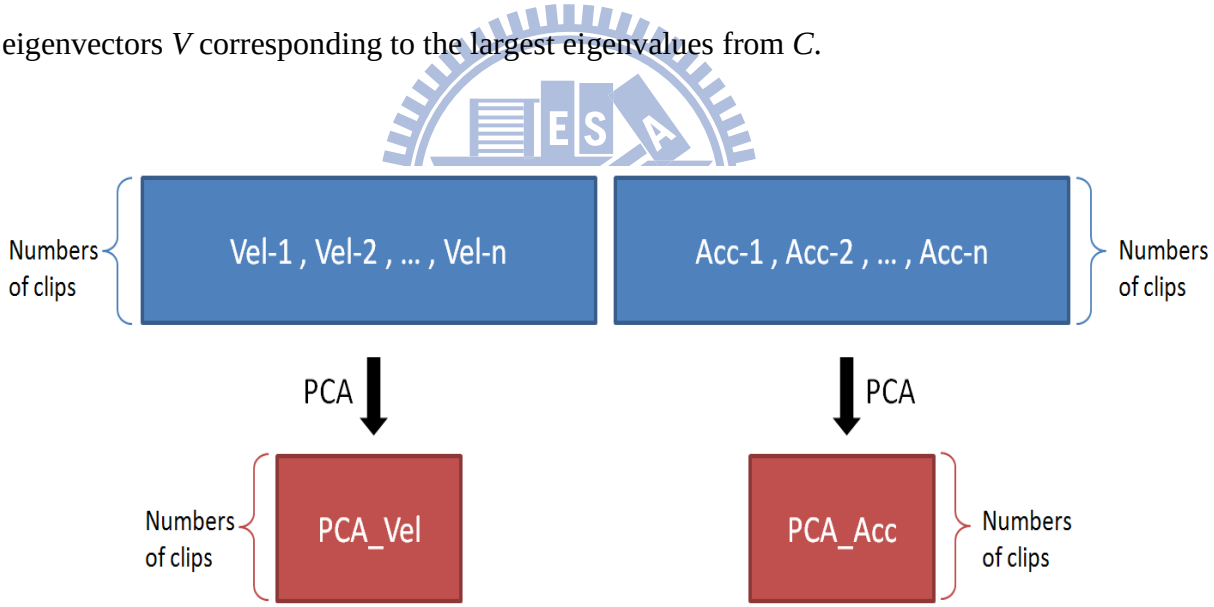

Figure 3.8: We reduce the dimensionality of angular velocity matrix and acceleration matrix derived by motion sensor.

3.4 Clustering

When we derived the new incoming training data from motion sensor, computing all of training motion clips in the database to find the mostly correlated one is very inefficient, since the searching time increases as the database grows. Such approach results in poor performance and is difficult to be employed in interactive applications. We prefer to partition the reduced-dimensionality training sensors' data into a number of clusters in pre-processing. Besides, these clusters will be used in building a statistic model. Hence, this approach makes the reconstructed motion with the data-driven model efficiently by searching the closely cluster during the runtime.

Given a large set of reduced-dimensionality training data of acceleration and angular velocity, we assign each of them with different user-defined weight. We have to change weights relying on different motion categories. Since the gyroscope is sensitive in rotations, we should set a larger weight on angular velocity when motion categories are with small details, such as forehand driving or backhand slicing in tennis. On the contrary, we set a larger weight on acceleration when categories are differing with obvious differences, such as squatting or running, where the accelerometer is sufficient to recognize and the sensitive gyroscope may be the noise.

Finally, these two data are combined according to assigned weights. Assuming that there are *M* clusters that should be partitioned from the merged data, we apply the k-means method to derive the cluster of each data. These partitions are iteratively calculated by their Euclidean distance from data to the means value of the cluster, until each element belongs to the cluster with the nearest mean.

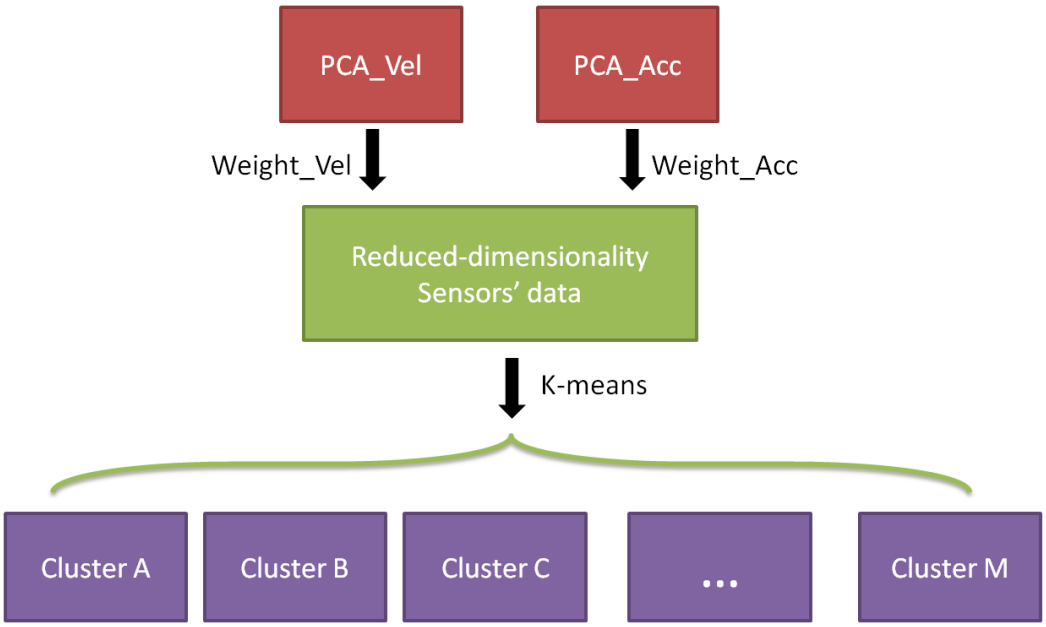

Figure 3.9: We partition the data into several clusters by K-means method.

3.5 Linear Discriminant Analysis

After associating the training motion clips with their corresponding clusters, we apply linear discriminant analysis (LDA) for better discrimination. The LDA method analyzes separating each cluster by subspace projection, and it preserves as much of the cluster discriminatory information as much as possible. This method is widely utilized in many applications, such as machine learning, prediction, pattern recognition, etc.

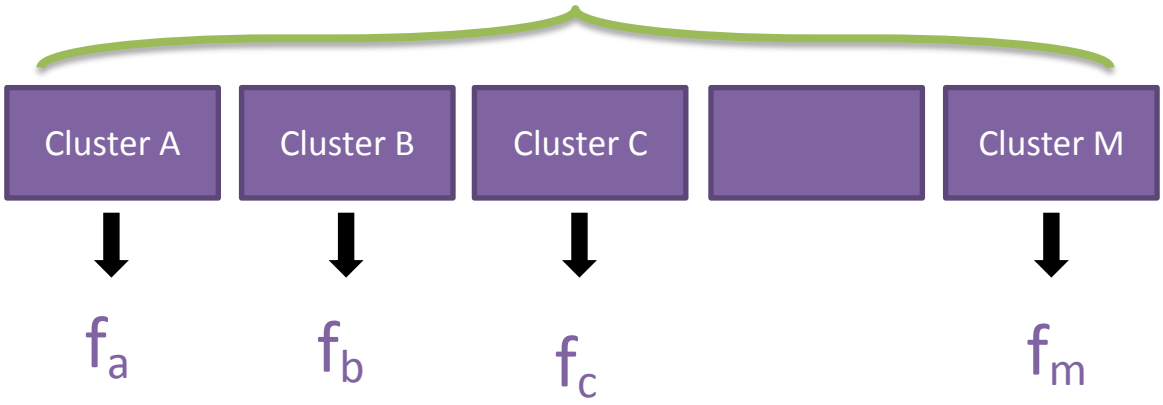

Figure 3.10: Each cluster with its discriminant function.

First of all, we denote training or online received data matrix as *x*, which each row is the frame number of the training motion, and each column is the angular velocity or acceleration data. The goal of LDA is to compute a matrix $W = [w_1|w_2|\cdots|w_{C-1}]$, and the data of new coordinate y_i is derived by projecting data *x* onto the w_i, where $1 \leq i \leq C-1$ and *C* is the total number of clusters.

$$y_i = w_i^T x \quad \rightarrow \quad y = W^t x \tag{6}$$

To compute the optimal projection *w**, we define a measure of the scatter in multivariate feature space *x*, or so called within-class scatter matrix. We denote it as S_W in this thesis.

$$S_w = \sum_{i=1}^{c} \left(\sum_{x \epsilon D_i} (x - m_i)(x - m_i)^t \right) \quad , \quad m_i = \frac{\sum_{x \epsilon D_i} x}{n} \tag{7}$$

Then, we compute the difference between means, or so called between-class scatter. We denote it as S_B in this thesis. The rank of S_B is as most (*C* - 1) since it is the sum of *C* matrices of rank one or less.

$$S_B = \sum_{i=1}^{c} n_i(m_i - m)(m_i - m)^t \quad , \quad m = \frac{\sum_{\forall x} x}{n} \tag{8}$$

Finally, the linear discriminant is defined as the linear function $w^t x$ that maximizes the ratio of between-class to with-in class scatter. The determinant of the scatter matrices is used to derive a scalar objective function.

$$J(W) = \frac{|W^t S_B W|}{|W^t S_w W|} \tag{9}$$

To obtain the optimal projection matrix W^* that maximizes the ratio, we derive it by transforming into the following generalized eigenvalue problem. Namely, the columns of matrix are the eigenvectors corresponding to the largest eigenvalues. This projections with maximum separability information are the eigenvectors corresponding to the largest eigenvalues of $S_w^{-1} S_B$.

$$W^* = [w_1^* | w_2^* | \cdots | w_{c-1}^*] = arg\ max \left\{ \frac{|W^t S_B W|}{|W^t S_w W|} \right\} \rightarrow \ S_B w_i \ = \lambda S_w w_i \tag{10}$$

In runtime, we derive the projection data y' by projecting the matrix data x' from motion sensors, and we obtain its corresponding cluster by applying discriminant function f. The data x' is in cluster j if the function f_j generates the minimum discriminant value.

$$cluster\ (x') = \underset{1 \le j \le C-1}{arg\ min}\ f_j = \underset{1 \le j \le C-1}{arg\ min} \left(y_j - y' \right)^2 \tag{11}$$

3.6 Action Graph

Without the concept of graph structure, two similar motion clips belonging to different motion categories may mislead into one or another when reconstructing human motion. This causes a visually discontinuous motion, and the artificial result is hardly accepted in any interactive application. Although the discontinuous motion can be blended with convoluted filtering algorithms, such as low-pass filtering, it is not an ideal solution since it only alleviates the discontinuity. A direct method to solve it is increasing the numbers of frames in a clip, and thus it minimizes problems in synthesizing the discontinuous motion, which is also able to perform in real-time application.

However, the critical side effect of using this method is latency time. Namely the numbers of frames in a training motion clip increase, the accumulative time from online received sensors' data should also be longer. Besides, the searching time for a training motion clip also increases. Finally, the problem of synthesis still exists, and merely the frequency to combine two motion clips decreases. Therefore, in order to prevent the problem of smoothness from synthesizing two discontinuous motion clips due to ambiguity of sensor' data, we propose constructing an action graph to avoid it. The idea of action graph is similar to motion graph in [KGP02]. The graph node represents a training motion clip from the training motion, and the graph edge is automatically generated transitions.

3.6.1 Detecting candidate transitions

When we have training motion clips from training motion database, we calculate the total Euclidean distance for each pair of clips, that is the difference from the positions of one's end frame to the position of other's start frame. It is not necessary that consider the middle frames of clip since a clip is short. After we have the distance between each pair of

training motion clips, we normalize the difference for finding the candidate transition. Then we construct a distance grid plot, whose element contains the normalized difference from one training motion clip to the other. It is beneficial to find candidate transitions and for probabilistic computation in hidden Markov Model.

In order to automatically search for the candidate transition, the user has to set $G_{threshold}$, a threshold value between zero and one. Then the difference that is under the $G_{threshold}$ will form the candidate transition, or what will be the edge of the graph. About the threshold setting, there is no definite value for all of motions. It depends on what motion categories that user wants to perform. If the motion is common in everyday life, such as walking, jumping or running, it is suggested that threshold should be set lower. The lower threshold provides a smoother transition of clips. Because people are sensitive to those common motions and can be easily aware if the transition is unusual. However, if the motion is a specific type like ballet or yoga, it is recommended that threshold is ought to be higher. Setting the higher threshold provides a higher connective graph. Therefore, the stretching or spinning move with huge differences is able to have flexible candidate transitions from those motion types.

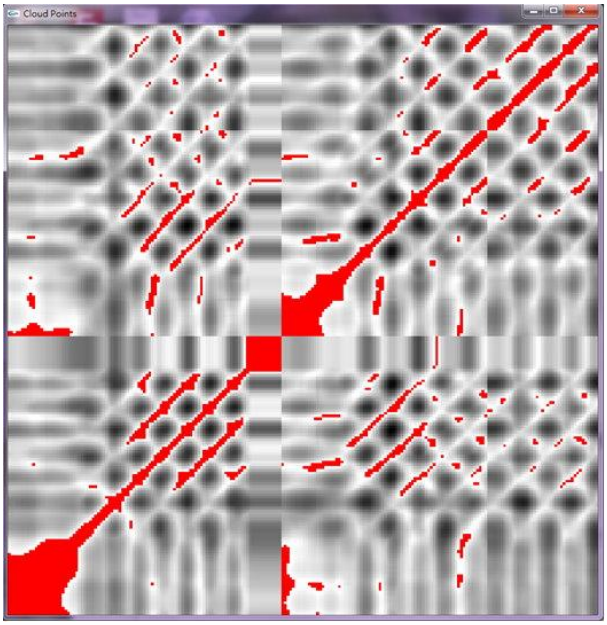

Figure 3.11: Taking 3 motions as an example. The distance grids plot shows that the distance between each pair of clips. (White point represents shorter distance. Red point represents distance under threshold. Black point represents longer distance.)

3.6.2 Constructing action graph

The action graph can be built after we have training motion clips and the candidate transitions in the database. A training motion clip contains a set of frames with all information of the character, such as the position of the root joint, the orientation of each joint, the reduced-dimensionality sensors' training data, and the corresponding number of cluster. The training motion clip represents the node of the graph. The length of the node, or the number of frames in a clip, should not set too long due to latency time. The longer size a clip is, the longer accumulating time (latency) we have for collecting sensors' online received data. We set ten to twenty frames per clip in our experiment, and the frame per second (FPS) is about sixty in the system. A candidate transition contains the incoming node and outgoing node, and the auto-generated a small set of frame containing with the information of the position of the root joint and the orientation of each joint. A candidate transition represents

the edge of the graph.

To construct action graph, we place all of training motion clips as nodes, and connecting two nodes with an edge if these two nodes are a candidate transition. However, the sink nodes or dead end nodes may exist in the graph, and they make the following motion infeasible. A dead end node occurs if there is no outgoing edge from this node, and a sink node happened if this node does not connect at least two other nodes. Consequently, the process of driving human motion by sensors will be halted if the system entered these nodes. To fix this problem, we eliminate sink and dead end nodes by traversing all nodes and edges.

When the system is traversing the graph for reconstructing motion of avatar in run time, it is able to synthesize the smooth motion from nodes and edges without spending extra time on calculating the transit frames.

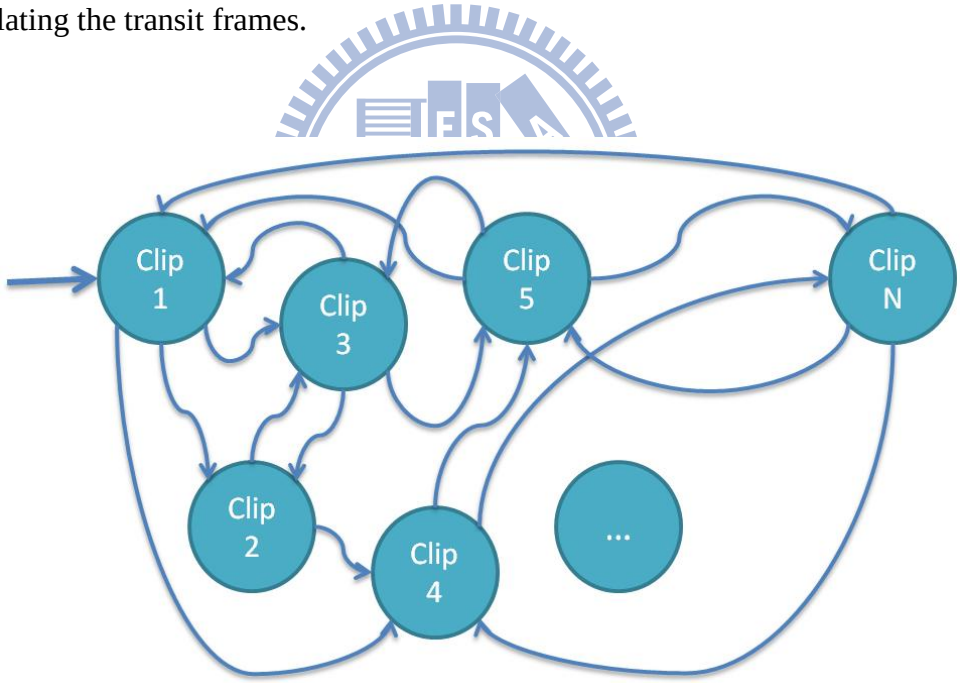

Figure 3.12: An example of action graph.

3.7 Hidden Markov Model

Signals received by motion sensors are continuous, time-varying, and easily interfere with noises. Therefore, for generality reasonable and visually pleasant motion, we characterize the statistical property of the signal data, and build a statistical model that is assumed to be a Markov process with unobserved states. This model is capable of providing the basis for a theoretical description of a signal processing system, and also can be used to provide us a desired output sequence of clips for reconstructing human motion. More specifically, instead of traversing the node of graph by calculating the nearest neighbor, we make use of a probabilistic model, hidden Markov Model, to search the reasonable sequence of training motion clips efficiently and accurately.

The hidden Markov Model (HMM) is a finite state machine that can be considered as the simplest dynamic Bayesian network. Since the sequence of states in HMM is hidden, they can only be conjectured by the given the sequence of observations. There are a number of researches using HMM to learn the various signals such as speech recognition, gene prediction, alignment of bio-sequence, etc. In our approach, the HMM is to be utilized in human motion recognition.

A standard hidden Markov Model contains the following elements: states, possible observations, state transition probability, observation (or emit) probabilities, and initial state distribution.

- States: We denote the individual state as $S = \{ s_1, s_2, s_3, \ldots , s_n \}$, where n is the numbers of state, and the state at time t as q_t. Though the state is hidden, but the physical significance is attached to sets of states of the model. Hence, a training motion clip represents as single state in our approach, and each state dependently decides the next state.
- Possible observations: We denote the individual possible observations as $O = \{ o_1, o_2, o_3, \ldots , o_m \}$, where m is the numbers of distinct possible observation per state. It is

correspondent with the output of the system being modeled. A data cluster obtained by motion sensor represents the possible observation in our approach.

- State transition probability: We denote the state transition probability distribution as $A = \{ \alpha_{ij} \}$, where

$$\alpha_{ij} = P\left[q_{t+1} = s_j \mid q_t = s_i\right], \qquad 1 \le i, j \le n \tag{12}$$

- Observation probability: We denote the observation probability distribution as $B = \{ \beta_i(k) \}$, where

$$\beta_i(k) = P[o_k at\ t \mid q_t = s_i], \qquad 1 \le i \le n,\ 1 \le k \le m \tag{13}$$

- Initial state distribution: We denote the initial state distribution as $\Gamma = \{ \gamma_i \}$, where

$$\gamma_i = P[q_1 = s_i], \qquad 1 \le i \le n \tag{14}$$

We denote the parameter of the model as $\pi = (A, B, \Gamma)$ for convenience. To derive the A, B, Γ in our approach, we will discuss it in the following section. Besides, there are a few of important constrains on the above three probability distributions, which are as the following:

$$\begin{cases} 0 \le \alpha_{ij}\ ,\ \beta_i(k)\ ,\ \gamma_i \le 1, & 1 \le i, j \le n \\ \sum_{j=1}^{n} \alpha_{ij} = 1, & 1 \le i \le n \\ \sum_{k=1}^{m} \beta_i(k) = 1, & 1 \le i \le n \\ \sum_{i=1}^{n} \gamma_i = 1 \end{cases} \tag{15}$$

Given a HMM, there are three basic problems:

- Given the model π and observation sequence O, how do we efficiently calculate the probability $P(O \mid \pi)$?
- Given the model π and observation sequence O, how do we generate the state sequence S that is most likely the optimal one?
- Given the model $\pi = (A, B, \Gamma)$, how do we adjust the parameters to maximize the probability $P(O \mid \pi)$?

There are standard solutions to solve these problems. For the first one, instead of computing all of the possible state sequences, it is better to efficiently perform forward algorithm by dynamic programming. For the second one, a formal solution to find a state sequence is also a dynamic programming algorithm, which is called Viterbi algorithm. This algorithm is closely related to forward algorithm for the computation of maximum probability, and thus the state sequence is derived by backtracking method. Besides, it is the most related to our approach. Namely, the problem of our system is how we generate the optimal sequence of motion clips based on the clusters obtained by motion sensor. For the final, the Baum-Welch method is the formal solution to adjust parameters to maximize the probability.

Therefore, our system applies the dynamic programming algorithm to derive the clip sequence, which is a method modified by the solution of Viterbi algorithm. Instead of finding a single Viterbi path, we compute more than one paths to blend them into one. To do so, we first compute the transition probability distribution A and the initial probability distribution Γ in action graph, and the observation probability distribution B from each node in graph to each cluster. With the probability distribution, we are able to build the hidden Markov Model $\pi = (A, B, \Gamma)$ for the later usage.

3.7.1 Generate states and observations

Before we construct the model π, we first generate two sets. They are states $S = \{ s_1, s_2, s_3, \ldots, s_n \}$, and observations $O = \{ o_1, o_2, o_3, \ldots, o_m \}$, where n is the numbers of states and m is the numbers of distinct possible observation per state.

As we mention, the training motion is divided into several clips, and action graph connects each smoothly pair of clips. Assume that the directed action graph is $G = (V , E)$. The set of vertices is denoted by $V = \{ v_1, v_2, v_3, \ldots, v_n \}$, and the set of edges is denoted by $E = \{ e_1, e_2, e_3, \ldots, e_w \}$, where w is the numbers of edge. The amount of nodes is the same as the amount of motion clips. Then we are now utilizing nodes of the action graph as the states of HMM. More specifically, we assign each vertex v_i to the state s_i, for all i is from 1 to n.

The online received data obtained by motion sensors for driving avatar are set into a group c_t using LDA method in each duration time t. When the $t = m$ occurs, we form the sequence of clusters $C = \{ c_1, c_2, c_3, \ldots, c_m \}$. Then we assign each cluster c_j to the observation o_j, for all j is from 1 to m. Given the model π and states S, this procedure is iteratively running in real-time in order to uncover the hidden states for reconstructing human motion.

3.7.2 Probability distributions

Now we are going to generate three probability distributions, the state transition probability distribution $A = \{ \alpha_{ij} \}$, the observation probability distribution $B = \{ \beta_i(k) \}$, and the initial state distribution as $\Gamma = \{ \gamma_i \}$, where $1 \leq i, j \leq n$, n is the numbers of states, and $1 \leq k \leq m$, m is the numbers of cluster. Therefore, our system is able to build the model $\pi = (A, B, \Gamma)$.

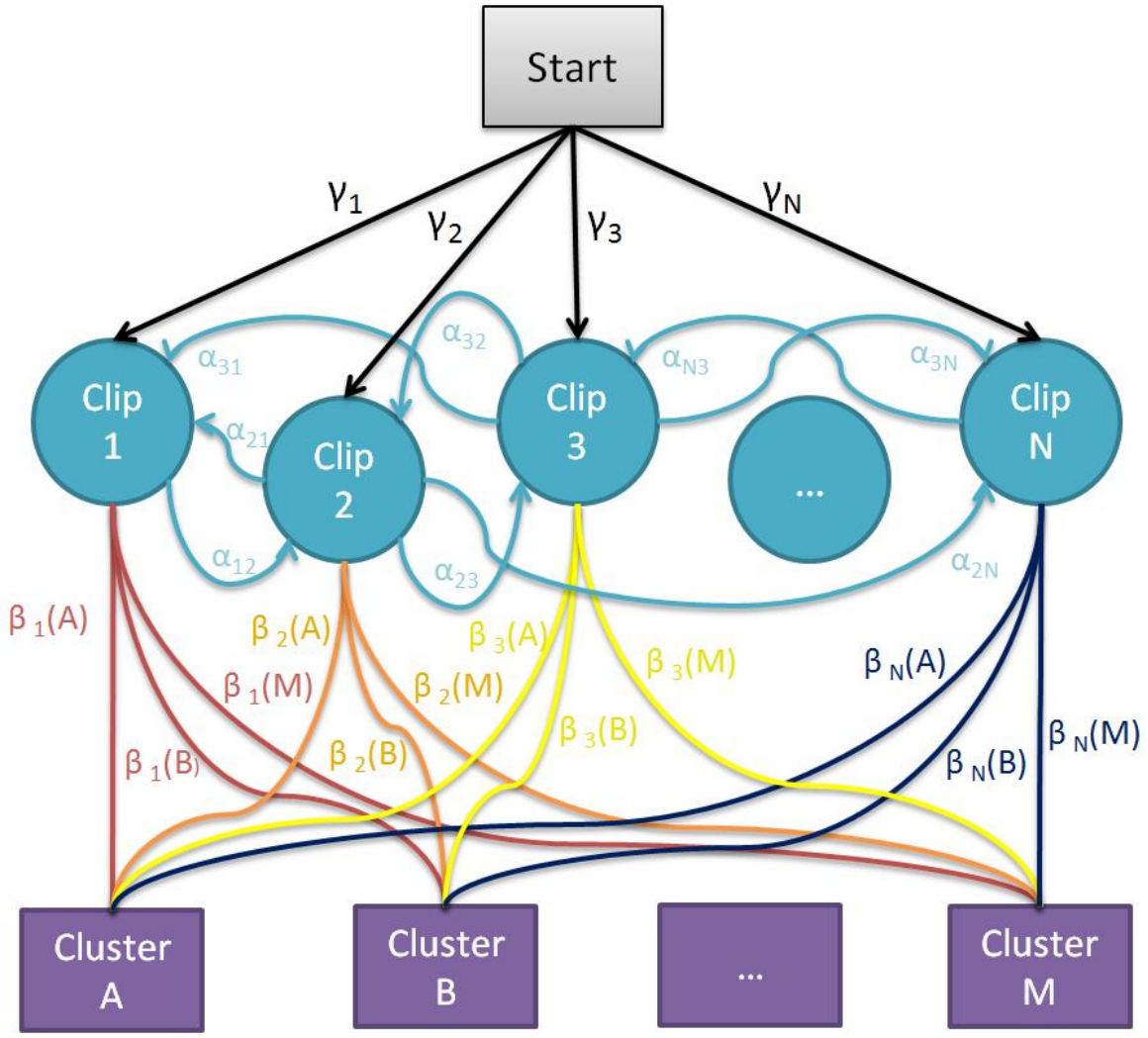

Figure 3.13: An example of hidden Markov Model that combines the action graph and the clusters with probability distribution. (The state transition probability distribution is denoted as $A = \{ \alpha_{ij} \}$. The observation probability distribution is denoted as $B = \{ \beta_i(k) \}$. The initial state distribution is denoted as $\Gamma = \{ \gamma_i \}$.)

Given the directed action graph $G = (V , E)$, $V = \{ v_1, v_2, v_3, \ldots , v_n \}$, and $E = \{ e_1, e_2, e_3, \ldots , e_w \}$, where w is the numbers of edge. In order to generate the state transition probability, we first simply assign the state probability $\alpha_{ij} = 0$ if the edge e_x connecting from node i to node j does not exist, namely the edge $e_x \notin E$, where $1 \leq i , j \leq n$ and $1 \leq x \leq w$. If the edge $e_x \in E$, we compute the probability as follows. Considering our action graph is constructed dependent on normalized Euclidean distance, and the intuition is

that human usually tend to act the similar motion rather than the dissimilar. Hence, we calculate the probability by that the longer distance having with the lower probability and the shorter distance having with the higher probability.

We denote the normalized distance from node i to node j as $Dist_{i,j}$, and the $G_{threhold}$ represents the user-define threshold for automatically candidates generation. The state probability distribution is computed by equation (16).

$$\alpha_{ij} = \frac{G_{threshold} - Dist_{i,j}}{\sum_{\forall(v_i,v_j)\in E}\left(G_{threshold} - Dist_{i,j}\right)}, \qquad \forall v_i \ , 1 \le i \ , j \le n \tag{16}$$

About observation probability, concerning that the clusters are partitioned by applying LDA on the online received data by sensors. Therefore, we calculate the discriminant value for each cluster and normalize these values to be the probability distribution from one state to each cluster. Besides, we subtract normalized discriminant value from one since the smaller value implies the larger probability.

We denote the normalized discriminant value from node i to cluster k as $Disc_i(k)$. The observation probability distribution is computed by equation (17).

$$\beta_i(k) = \frac{1.0 - Disc_i(k)}{\sum_{k'=1}^{m}(1.0 - Disc_i(k'))}, \quad 1 \le i \le n, \ 1 \le k \le m \tag{17}$$

Every time the system starts to reconstruct motion, we ask user to stand the same pose as the avatar in the first training motion clip. Therefore, we generate the initial probability distribution by equation (18).

$$\begin{cases} \gamma_1 = 1.0 \\ \gamma_i = 0.0, \qquad 2 \le i \le n \end{cases} \tag{18}$$

3.7.3 Viterbi algorithm

In order to get the maximum probability of the path in states, we adapt a dynamic programming algorithm, Viterbi algorithm, to derive the most likely sequence of hidden states. This path is sometimes called Viterbi path, and it represents the optimal sequence of hidden state corresponding to given observations.

There are a few of assumptions satisfied in a first-order hidden Markov Model before we apply Viterbi algorithm. The state sequence S and observation sequence O must be aligned by time point t with the same amount. Besides, to uncover the hidden state s_t, it can only be computed by the dependence on the observation o_t and the optimal sequence at point $(t-1)$.

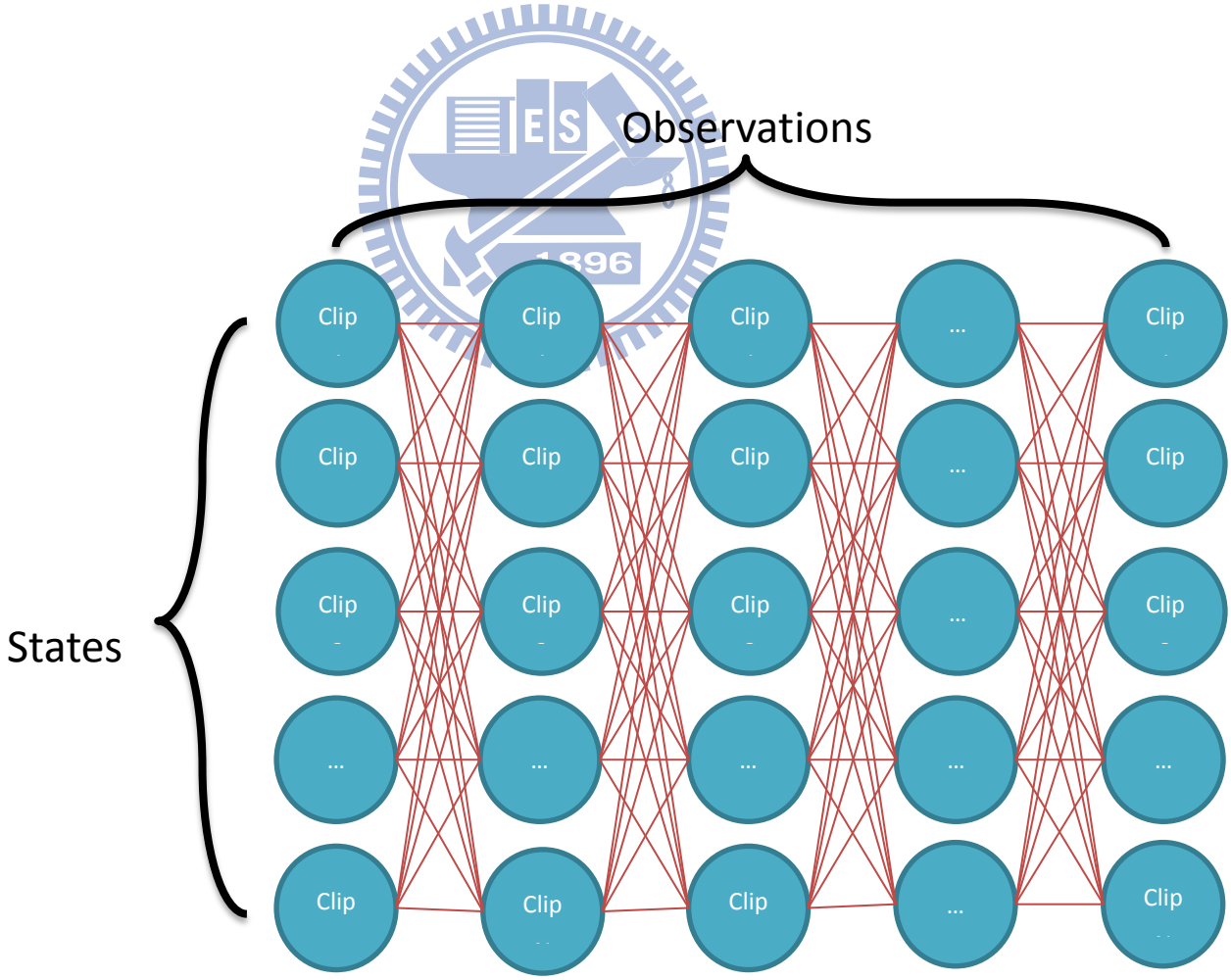

Figure 3.14: An example of Viterbi algorithm by given states and observations.

We are going to apply the Viterbi algorithm with the given hidden Markov Model π = (A, B, Γ) and states S = { s_1, s_2, s_3, ... , s_n }. Say we observe online received data O = { o_1, o_2, o_3, ... , o_m } from motion sensors, and the state sequence X = { x_1, x_2, x_3, ... , x_m } most likely be produced by the recurrence relations.

We denote the probability of optimal sequence as $Z_k(i)$, that is in charge of the first k observations on state i.

$$\begin{cases} Z_1(i) = \beta_i(o_1) \cdot \gamma_i & , \quad k = 1 \quad , \quad 1 \le i \le n \\ Z_k(i) = \beta_i(o_k) \cdot max_{x \in S}\left(\alpha_{xi} \cdot Z_{k-1}(i)\right), & 2 \le k \le m, \quad 1 \le i \le n \end{cases} \tag{19}$$

Since the Viterbi algorithm is based on dynamic programming method, the optimal sequence, or Viterbi path, can be retrieved by saving back pointers, $U_k(i)$, for the first k observations on state i.

$$\begin{cases} U_1(i) = 0 & , \quad k = 1 \quad , \quad 1 \le i \le n \\ U_k(i) = arg\, max_{x \in S}\left(\alpha_{xi} \cdot Z_{k-1}(i)\right), & 2 \le k \le m, \quad 1 \le i \le n \end{cases} \tag{20}$$

Then, we need a temporary state sequence Y = { y_1, y_2, y_3, ... , y_m }.

$$y_k^* = \underset{1 \le i \le n}{arg\, max}\left(Z_k(i)\right), \quad 1 \le k \le m \tag{21}$$

Finally, the Viterbi Path can be derived by the back tracking by the following recursively equation.

$$x_k^* = U_{k+1}(y_{k+1}^*), \quad k = m-1, m-2, m-3, \dots, 1 \tag{22}$$

3.8 Motion Blending

In order to produce the motion which is not in the database, we propose performing motion blending methods. Rather than finding a single Viterbi path with the maximum probability, we find more than one path. If their maximal probabilities are close enough, or larger than user-define threshold, we blend the motion clips from paths.

A optimal sequence of hidden states is obtained by the given model π, states S and observations O using canonical Viterbi algorithm, and this sequence possess the maximum probability comparing with other sequences. This optimal sequence, Viterbi path, is formed the sequence of motion clips for reconstructing human motion. But there is a limitation. For example, we have training motion that character swings a bat upward and downward. If a user tries to swing forward, the system derives the Virterbi path either swinging upward or swinging downward, depending on whose probability is larger. However, this motion might be blended using exist training clips. Therefore, we obtain the first v maximum probability $D = \{ d_1, d_2, d_3, \dots , d_v \}$ and the corresponding Virterbi paths $H = \{ h_1, h_2, h_3, \dots , h_v \}$, where v is the user-defined threshold of path amount. Probability d_i is assigned by the i_{th} maximal value of $Z_m(j)$ and h_i is derived by the recurrence equation, $1 \leq i \leq v,\ 1 \leq j \leq n$.

Then, we collect the probability d_i such that $p_{threhold} \leq d_i, 2 \leq i \leq v$, where $p_{threshold}$ is a user-defined threshold from zero to one. This threshold constrains the smallest probability that the corresponding state sequence can be blended. More specifically, we blend the sequence h_i into h_1 with setting the ratio of weights if the d_i meets the threshold; otherwise we drop the sequence h_i. The ratio of blending weight is denoted by $W = \{ w_1, w_2, w_3, \dots , w_l \}$, where l is the last sequence which d_l meets the constraint if $l = v$ or $p_{threshold} > d_{l+1}$. The elemental value of W is computed by the following equation.

$$w_i = \frac{d_i}{\sum_{j=1}^{l} d_j}, \qquad 1 \leq i \leq l \tag{23}$$

3.9 Real-time Motion Reconstruction

The user should wear a few of Wii remotes with MotionPlus as the motion sensors. In the beginning, we put all sensors on a flat surface, and the system automatically detects the background noise to revise the coming sensors' data every time. Then, the online received data are collected into data pool for duration, and the system reduce the dimension of data by applying PCA as section 3.3. By user-defined weights, the significance between acceleration and angular velocity can be evaluated by the user. The final online received data is partitioned by performing LDA as section 3.5, and its group is represented as an observation in HMM. These procedures are iterative operated until the number of observations reaches to the expected number.

When the number of observations is sufficient, the HMM is integrated with action graph and clusters. This model provides the final motion more reliable and of theorization. To reconstruct motion, we apply modified Viterbi algorithm on multiple paths as section 3.8. Namely, we blend closet clips and transitions for unseen motions beyond the database. The blended clips and transitions are synthesized into a series of frames displaying on the screen, and the above procedures are also running for reconstructing next motion in the meanwhile.

Besides, since reconstructing the combination of different motion categories is more complex than a specific motion, we should ask the user to wear more sensors to provide sufficient information. That is, our system calculates the variation of angular rate on two wrist and two legs from the training motion database and those variations are normalized into percentage. For example, the percentages on two wrists are higher than two legs on sword playing motion. On the contrary, the percentages on two wrists are lower on walking motion.

Finally, our system also performs on various motions with training data and sensors' percentage information of angular rate. To implement it, we give a user-defined bonus on probability $Z_k(i)$ from equation (19) if percentages on two wrists and legs are matched.

In our multi-threading system, reconstructed motions can be run at a rate of 0.016 seconds/frame, which are around 60 frames per second (FPS) animation. Therefore, this system provides the user to drive avatar's motion by inexpensive sensors in real-time performance. Furthermore, this easy-to-use system can also be applied to advanced interaction systems, such as games, computer animation, or virtual environment.

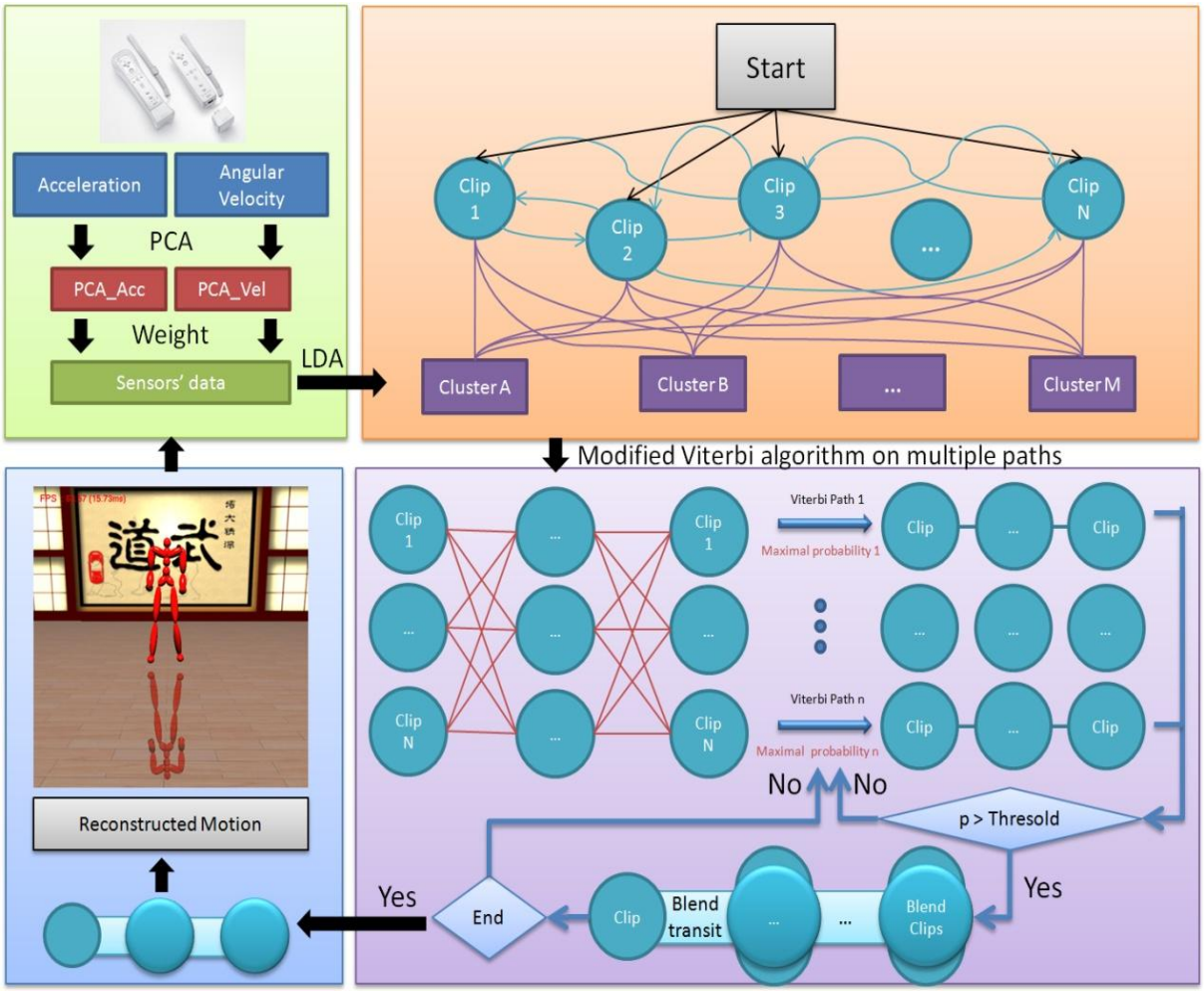

Figure 3.15: The framework to reconstruct human motion by sensors.

4. Implementation of Our System

Our system is implemented by using C++ language of object-oriented programming and built based on .NET Framework 3.5 with Visual Studio 2008. Several external library are utilized as well, such as OpenGL, MATLAB and WiiYourself. OpenGL library renders the scene and avatar, MATLAB calculates the singular value decomposition for polynomial function, and WiiYourself connects the Wii Remotes with MotionPlus via Bluetooth technology. The graphic user interface (GUI) is developed through windows form. Besides, Our system performs in multi-threading styles: one for collecting and processing the online received data by sensors, and another for rendering the reconstructed human motion.

The procedure of this system includes loading motion, training motion, and connecting motion sensors for driving character animation in real-time. Furthermore, we also implement four other methods to reconstruct motion in comparison with our approach, which are k-nearest neighbor (KNN), principle component analysis (PCA), linear discriminant analysis (LDA), and polynomial function.

- KNN: We have motion sensors' data associated with training motion clips. During runtime, we search for *k* clips that are the closest to the incoming sensing data in the database, and blending them into one for playing back human motion.
- PCA: We have reduced-dimensionality motion sensors' data associated with training motion clips. During runtime, we search for the closet training motion clips from incoming reduced-dimensionality sensors' data in the database, and playing back human motion.
- LDA: We have motion sensors' data associated with training motion clips, and we compute the linear discriminant function f_i for each training motion clip i. When new input data comes, we play back human motion clip j if f_j returns the minimum discriminant value from all of functions.

- Polynomial: We assume that positions of skeleton joints create a vector, sensors' data are variables, and degree of polynomial is dynamically selected by the user. Then coefficients are calculated by applying singular value decomposition (SVD), which is the canonical solution of linear least squares, and they are stored in the database. Finally, when new input data comes, we are able to reconstruct human motions by polynomial function.

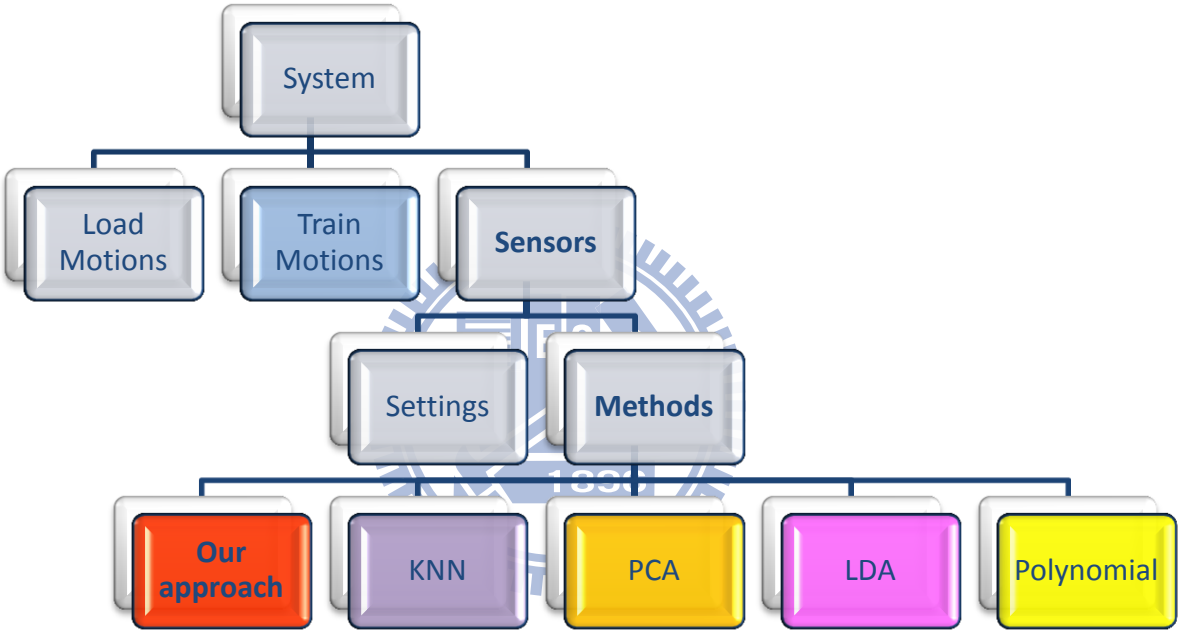

Figure 4.1: A simple tree of system framework is showed. Five methods to drive character's animation for comparison will be displayed by different colors.

This system begins with loading motion capture data from CMU MoCap library as the exemplar training motion, and the data contains the information of the position of root joint and the local rotations of other joints on each frame. Then, the global positions of the skeleton are computed for rendering skeleton, and we apply the training process with these information. Finally, the training motions are stored in the database for later driving avatar's

motion in real-time. Since we present four other methods in our implementation, those avatars are separated with different colors in order to distinguish the reconstructed motion from different methods. The red avatar is our approach, the purple one is KNN method, the orange one is PCA method, the pink one is LDA method, and the yellow one is polynomial method. Besides, the blue avatar represents the training motion from motion capture device.

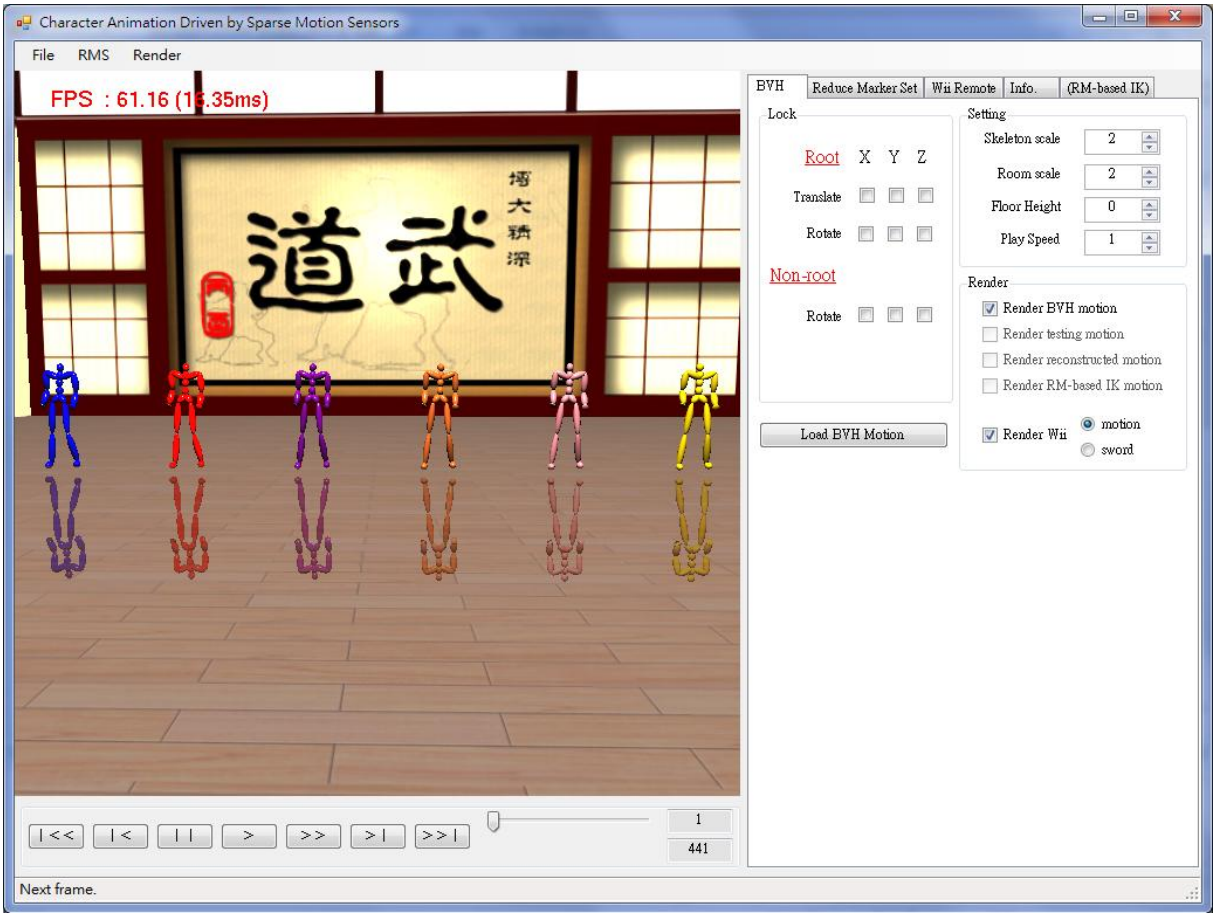

Figure 4.2: A screenshot of our system. The blue avatar represents the motion from training database. The red avatar represents the motion reconstructed by our approach. The purple avatar represents the motion reconstructed by KNN. The orange avatar represents the motion reconstructed by PCA. The pink avatar represents the motion reconstructed by LDA. The yellow avatar represents the motion reconstructed by polynomial.

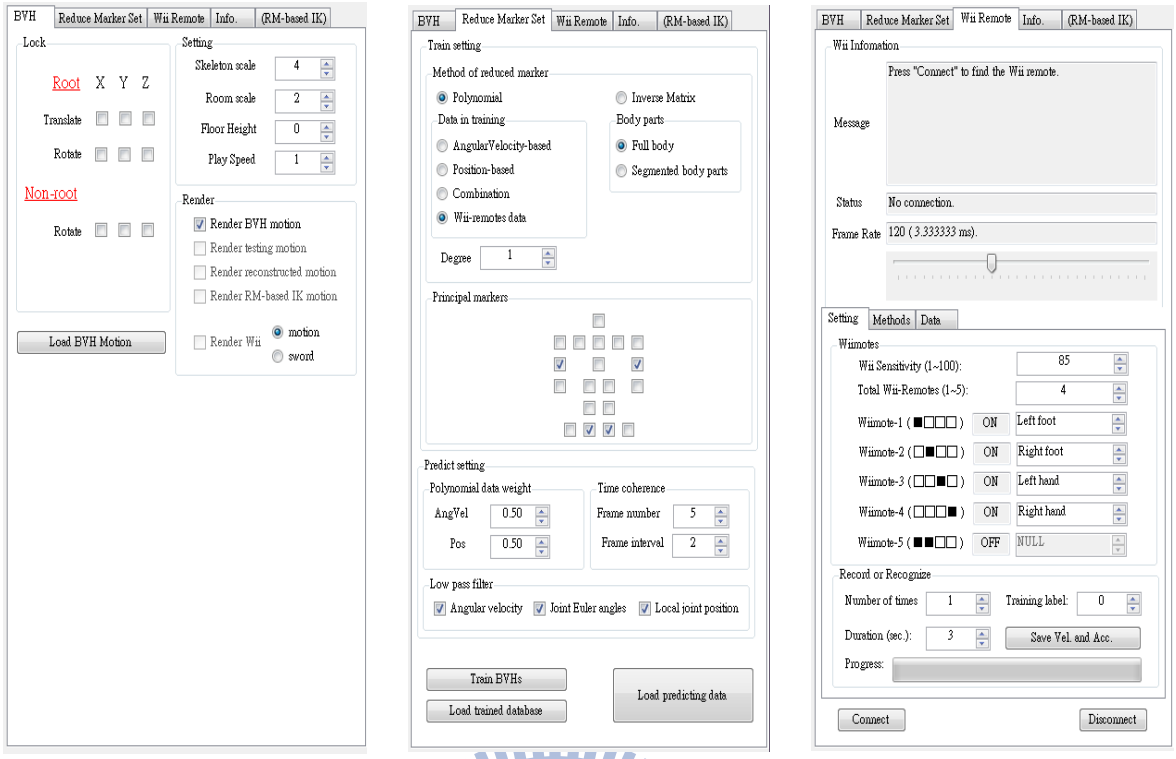

Figure 4.3: Tab pages to all functions.

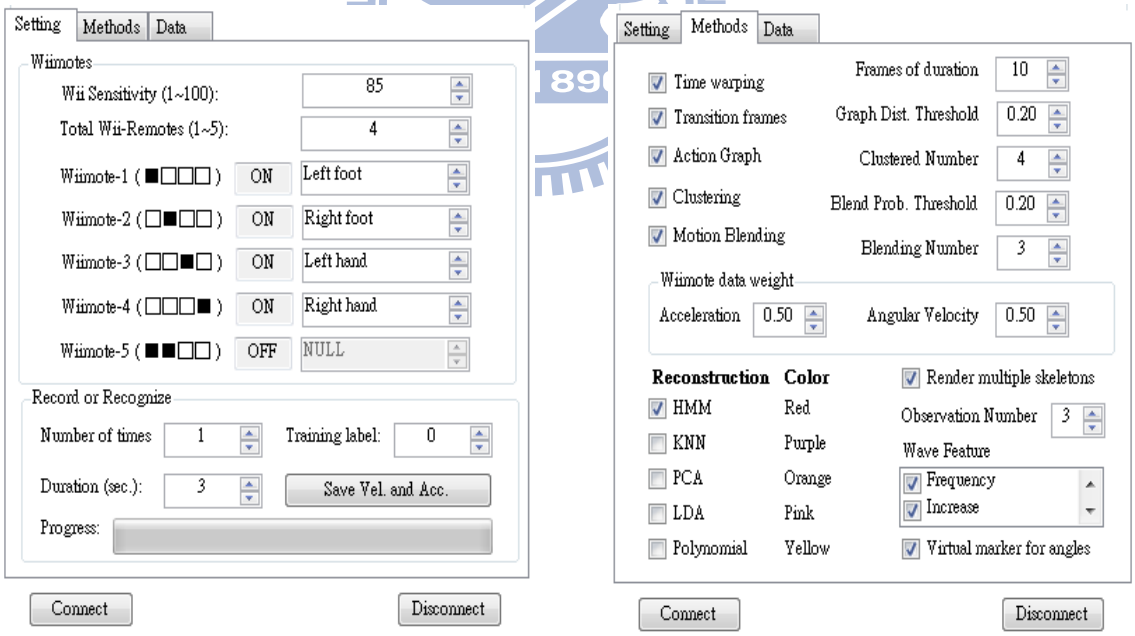

Figure 4.4: Tab pages in "Wii Remote."

5. Experiments and Results

Our experiments perform on a desktop with Intel® Core™ i7-930 Processor, 6GB main memory, and ATI Radeon HD 5770 graphics card. In our experiments, there are five motions in our training database, which are walking, running, sword playing, basketball free throw, and baseball pitch.

To drive a specific motion, our approach is able to provide a smooth and high quality of reconstructed motion by wearing one or two motion sensors. However, few sensors are unfeasible to be applied to combination of motions, due to insufficient information for distinguish from one motion to the other.

To drive a series of motions, such as running, sword playing, and then walking, we need to ask the user wears three or four sensors. Since the ratios of variations of orientations on two wrists and two legs are different for each motion category, and they can be calculated from training database in pre-processing, those attached sensors can provide the reliable information on variations of orientation. Therefore, when the user is trying to drive a character from one motion to the other, such as from walking to sword playing, we can evaluate that the variation of orientations on legs decreases and on wrists increases.

The variation of orientations θ is defined as below.

$$\theta = \sqrt{\frac{\sum_{f=2}^{n}\left(\tau_{f,i} - \tau_{f-1,i}\right)^2}{n}} \tag{24}$$

where n is the total frame number, $\tau_{f,i}$ represents in frame *f* and joint *i*, the summation of orientation on *x*, *y*, *z* rotations in training motion data, or pitch, roll, yaw rotations in sensors' data.

Table 5.1: Average ratio of orientation on five motions.

Motions	Data	Average ratio of orientation			
		Right hand	Left hand	Right foot	Left foot
Walking	Sensors data	0.183	0.208	0.330	0.279
	Training database	0.149	0.180	0.319	0.352
Running	Sensors data	0.179	0.181	0.324	0.316
	Training database	0.130	0.123	0.366	0.381
Sword playing	Sensors data	0.558	0.117	0.171	0.154
	Training database	0.415	0.153	0.216	0.216
Basketball free throw	Sensors data	0.517	0.255	0.120	0.108
	Training database	0.365	0.479	0.076	0.080
Baseball pitch	Sensors data	0.470	0.280	0.123	0.127
	Training database	0.353	0.208	0.188	0.251

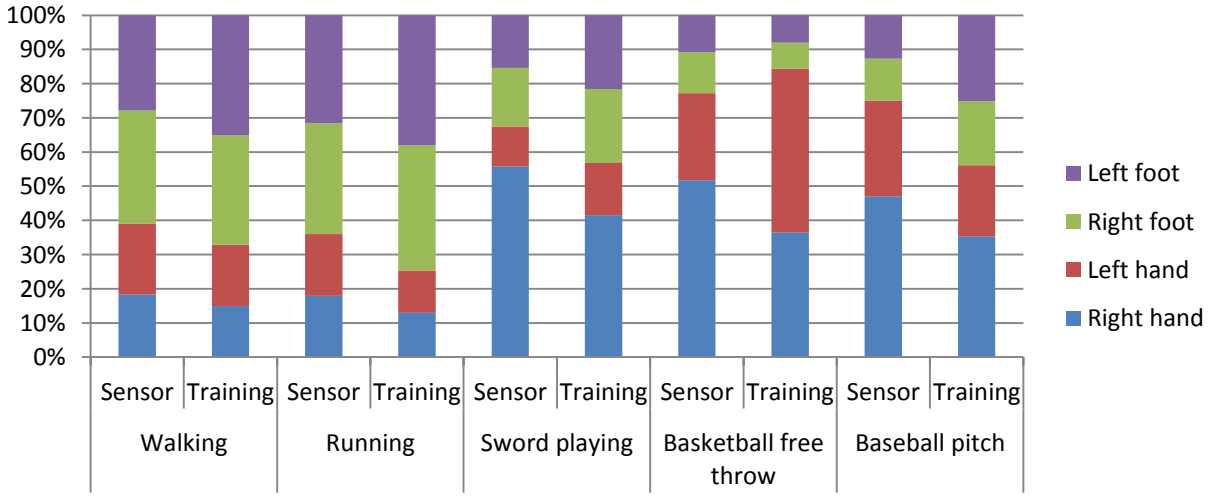

Figure 5.1: Average ratio of orientation on five motions.

5.1 Experimental Design

In our experiment, we use four motion sensors attached on user's wrists and legs to derive acceleration and angular velocity with the same data weights in 60Hz frame rate (16.6667ms). All parameters of our approach are described as follows. We set 10 frames for a training motion clip, 10 clusters for training data, 3 observations length for the hidden Markov Model. Besides, for the action graph, we set the distance threshold $G_{threshold}$ to 0.2 for a specific motion and 0.35 for the combination of different motions. The blending probability threshold $p_{threshold}$ is set to 0.2 for the modified Viterbi algorithm.

The motion capture data from CMU Mocap library is used as the ground truth data. They are walking, running, sword playing, basketball free throw, and baseball pitch. To measure the accuracy of our approach, we perform the evaluation by applying Root Mean Square (RMS). Besides, the accuracy of four other methods are compared as well. The ground motion and reconstructed motions are converted from joint angle data to joint local position data for error calculation, and the error value ε of RMS is computed by Euclidean distance by the equation defined as below.

$$\varepsilon = RMS(\omega_f, \tilde{\omega}_f) = \sqrt{\frac{\sum_{i=1}^{n}(\omega_{f,i} - \tilde{\omega}_{f,i})^2}{n}} \quad (25)$$

where ω_f is the ground truth motion data, $\tilde{\omega}_f$ is the reconstructed motion, f is the frame index number, n is the total dimension, and $\omega_{f,i}$ is the i^{th} dimension of ω_f.

For comparison, we also compute RMS for other reconstructing methods, and those are k-nearest neighbor (KNN), principle component analysis (PCA), linear discriminant analysis (LDA), and polynomial function.

5.2 Experimental Result

Table 5.2: Average RMS on five motions.

Motions	Processing time (sec.)	Frame numbers	Average RMS				
			Our approach	KNN	PCA	LDA	Polynomial
Walking	8.71	529	2.927	3.857	3.480	3.479	3.793
Running	3.53	202	2.082	4.185	4.723	4.790	3.597
Sword playing	12.15	749	6.542	7.540	6.708	7.893	6.526
Basketball free throw	6.24	362	6.809	7.180	7.262	6.842	7.9695
Baseball pitch	5.26	320	5.242	7.936	7.981	6.999	7.108

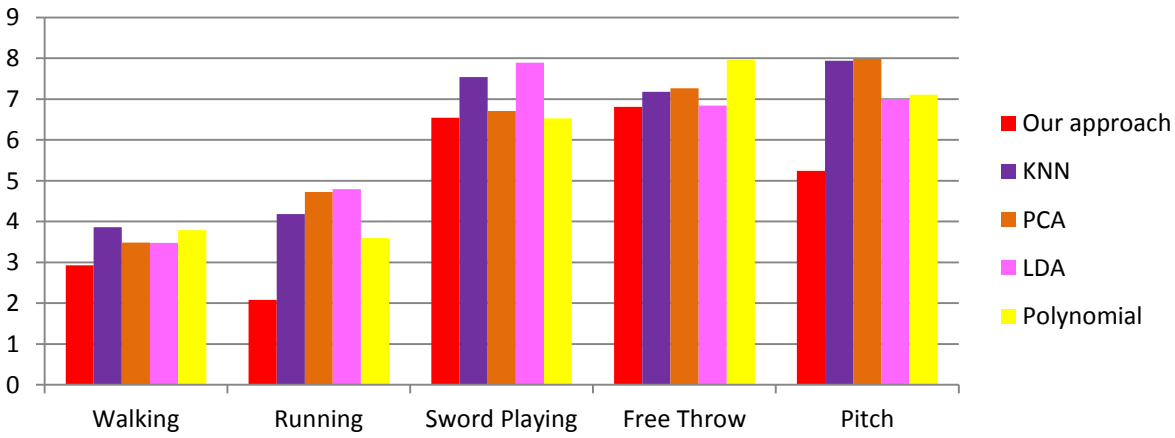

Figure 5.2: Average RMS on five motions.

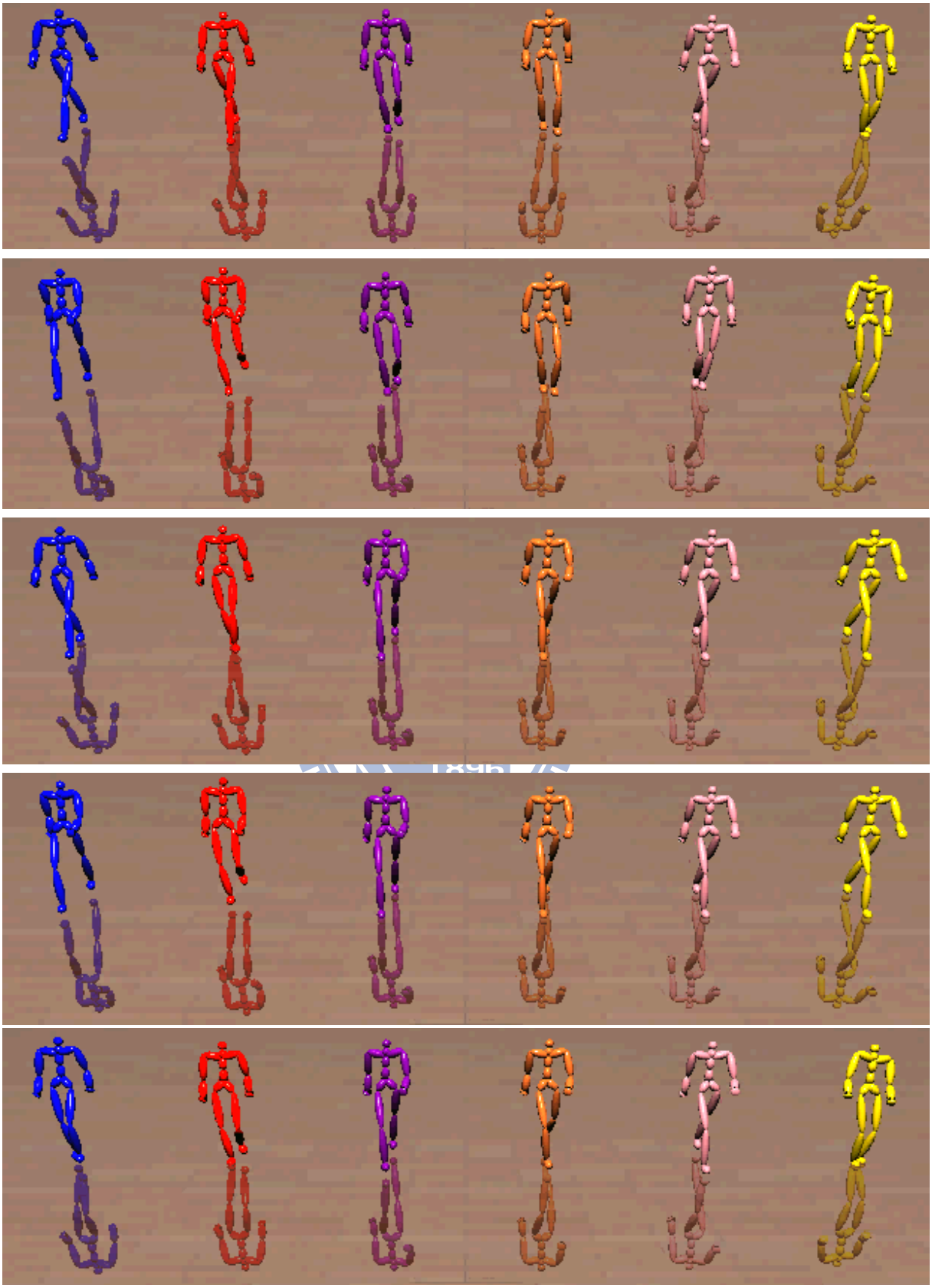

Figure 5.3: Screen shots on reconstructing walking motion. The avatars from left to right are ground truth, our approach, KNN, PCA, LDA, and Polynomial functions.

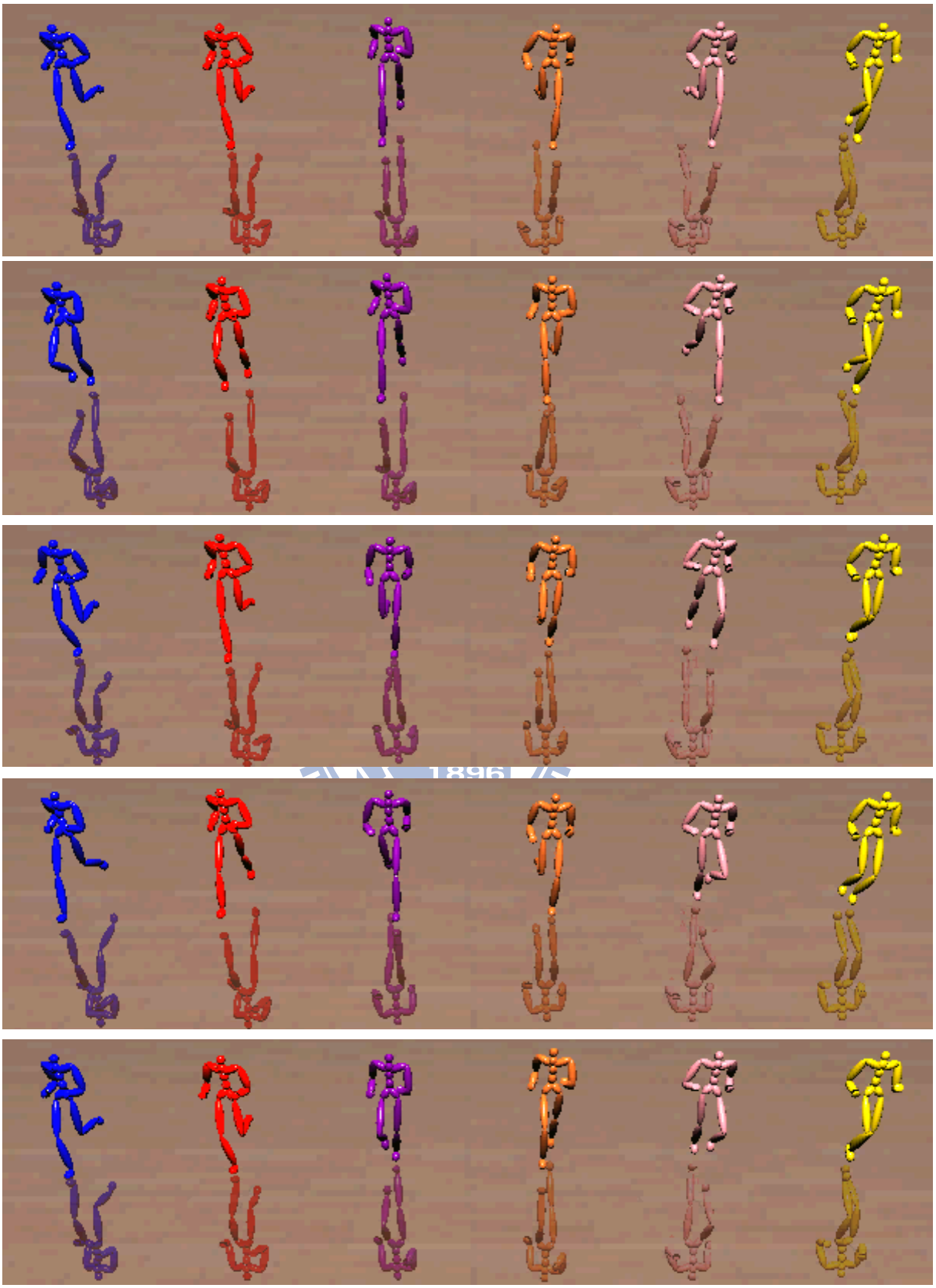

Figure 5.4: Screen shots on reconstructing running motion. The avatars from left to right are ground truth, our approach, KNN, PCA, LDA, and Polynomial functions.

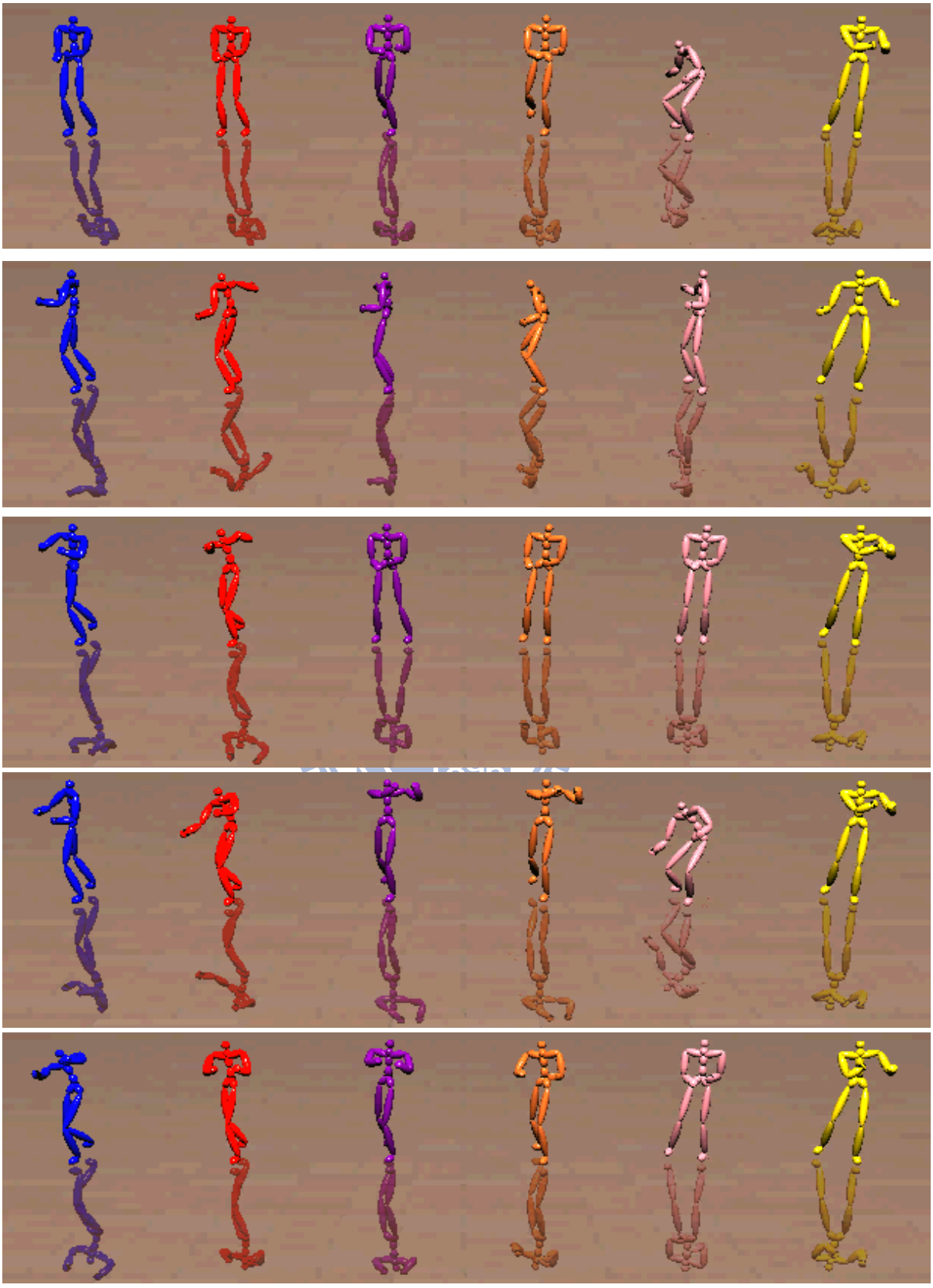

Figure 5.5: Screen shots on reconstructing sword playing motion. The avatars from left to right are ground truth, our approach, KNN, PCA, LDA, and Polynomial functions.

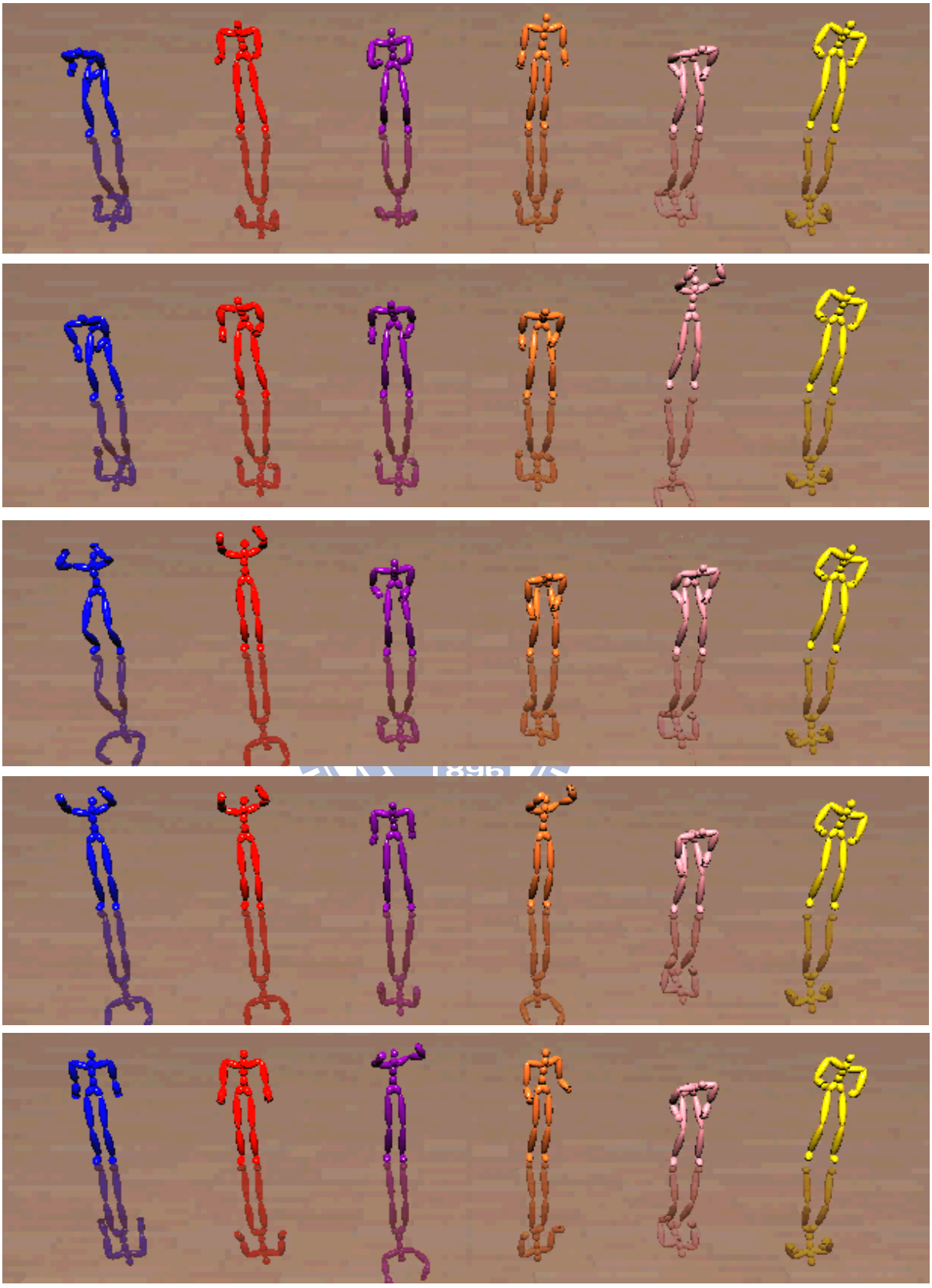

Figure 5.6: Screen shots on reconstructing basketball free throw motion. The avatars from left to right are ground truth, our approach, KNN, PCA, LDA, and Polynomial functions.

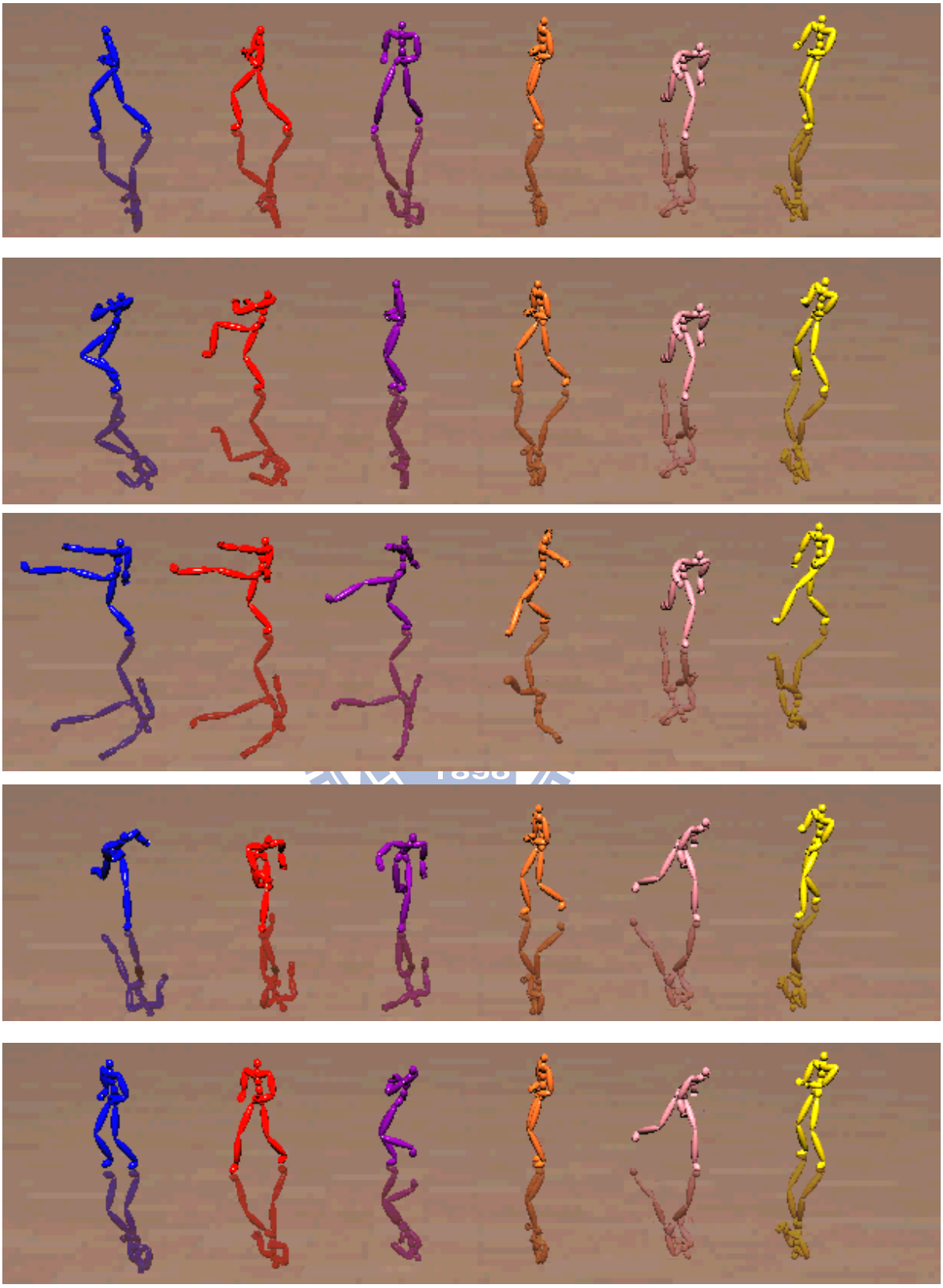

Figure 5.7: Screen shots on reconstructing baseball pitch motion. The avatars from left to right are ground truth, our approach, KNN, PCA, LDA, and Polynomial functions.

5.3 Discussion

In experiments, we demonstrate that our approach possesses the better results than other four methods, according to the lower RMS error. Though, we perform motion blending method, which makes the reconstructed motion with beyond to the training motion but similar to user's action. This RMS value is also accounted.

Our approach is also applicable if the user suddenly stops an action or changes one motion to the other. Namely, if we observe that variations of orientations from all sensors suddenly decrease substantially, our model is able to stop reconstructing motion with keeping the same pose as the user. If the significant variation of orientations alters, such as from hands to legs or from legs to hands, the probability on Viterbi algorithm can be tuned automatically by system in the meanwhile. Then, it computes the maximum possibility on motion sequence to follow up the user's action.

Furthermore, we solve the problem of synthesizing motions by applying graph structure, and thus the final reconstructed motion is much more continuous than others without using graph structure. Although this problem can also be solved by setting considerable frames in a training motion clip, the longer latency time to accumulate online sensors' data is a critical side effect.

Finally, in our approach, the PCA method reduces the dimension of sensors' data and it preserves the 99% of the motion variance. The LDA method analyzes separating each cluster by subspace projection, and it also preserves as much of the cluster discriminatory information as much as possible. Therefore, our system is able to run in real-time performance, which is around 60 frames per second (60 FPS) rate.

6. Conclusions

In this thesis, we propose a novel method to drive avatar's motion by utilizing sparse motion sensors. We ask a user to follow training motion and record the data of acceleration and angular velocity to retrieve features associating with training motion. Then, the training motion is divided into several small overlaying clips, and these clips are partitioned based on Euclidean distance by applying K-means method. Besides, we solve motion synthesis problem by constructing an action graph to connect each smooth pair of clips. While applying hidden Markov Model and performing the modified Viterbi algorithm on multiple paths, the reconstructed motion is much more reasonable and is capable of playing motion unseen in the database. Finally, a user can drive character animation by motion sensors.

The Wii remotes with MotionPlus are the only requirements for capturing motion features, and the automatic pre-processing and post-processing can be prevent users from setting a few of threshold constraints. Therefore, our approach is able to provide low-cost but high quality full-body motion in interactive applications, and it performs in environment without motion capture device.

References

[AT04] Agarwal, A. and Triggs, B. 2004. 3D Human Pose from Silhouettes by Relevance Vector Regression. cvpr, vol. 2, pp.882-888, 2004. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'04) - Volume 2.

[CH05] Chai, J. and Hodgins, J. K. 2005. Performance animation from low-dimensional control signals. ACM Trans. Graph. 24, 3 (Jul. 2005), pp. 686-696.

[CHP07] Cooper, S., Hertzmann, A., and Popović, Z. 2007. Active learning for real-time motion controllers. In ACM SIGGRAPH 2007 Papers (San Diego, California, August 05 - 09, 2007). SIGGRAPH '07. ACM, New York, NY, 5.

[CJM03] Chu, C.W. , Jenkins, O.C., and Mataric, M. J. 2003. Markerless Kinematic Model and Motion Capture from Volume Sequences, cvpr, vol. 2, pp.475, 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '03) - Volume 2.

[FBRA07] Farella, E., Benini, L., Riccò, E., and Acquaviva, A. 2007. MOCA: A Low-Power, Low-Cost Motion Capture System Based on Integrated Accelerometers. In Advances in Multimedia, vol. 2007, Article ID 82638, 11 pages.

[IWZL09] Ishigaki, S., White, T., Zordan, V. B., and Liu, C. K. 2009. Performance-based control interface for character animation. ACM Trans. Graph. 28, 3 (Jul. 2009), pp. 1-8.

[J02] Jolliffe, I.T.. Principal Component Analysis, 2nd, Springer, 2002.

[KGP02] Kovar, L., Gleicher, M., and Pighin, F. 2002. Motion graphs. ACM Trans. Graph. 21, 3 (Jul. 2002), pp. 473-482.

[KG05] Kwon, D. Y. and Gross, M. 2005. Combining body sensors and visual sensors for motion training. In Proceedings of the 2005 ACM SIGCHI international Conference on Advances in Computer Entertainment Technology (Valencia, Spain, June 15 - 17, 2005). ACE '05, vol. 265. ACM, New York, NY, pp. 94-101.

[LH99] Lee, J. and Ha, I. 1999. Sensor fusion and calibration for motion captures using accelerometers. Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on , vol.3, pp.1954-1959.

[LZWM06] Liu, G., Zhang, J., Wang, W., and McMillan, L. 2006. Human motion estimation from a reduced marker set. In Proceedings of the 2006 Symposium on interactive 3D Graphics and Games (Redwood City, California, March 14 - 17, 2006). I3D '06. ACM, New York, NY, pp. 35-42.

[MK01] Martinez, A.M. and Kak, A.C. , 2001. PCA versus LDA. Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol.23, no.2, pp.228-233.

[MRWS*99] Mika, S., Ratsch, G., Weston, J., Scholkopf, B., Mullers, K.R. 1999. Fisher discriminant analysis with kernels. Neural Networks for Signal Processing IX, 1999. Proceedings of the 1999 IEEE Signal Processing Society Workshop, pp. 41-48.

[R90] Rabiner, L. R. 1990. A tutorial on hidden Markov models and selected applications in speech recognition. In Readings in Speech Recognition, A. Waibel and K. Lee, Eds. Morgan Kaufmann Publishers, San Francisco, CA, pp. 267-296.

[SH08a] Shiratori, T. and Hodgins, J. K. 2008. Accelerometer-based user interfaces for the control of a physically simulated character. ACM Trans. Graph. 27, 5 (Dec. 2008), pp. 1-9.

[SH08b] Slyper, R. and Hodgins, J. K. 2008. Action capture with accelerometers. In Proceedings of the 2008 ACM Siggraph/Eurographics Symposium on Computer Animation (Dublin, Ireland, July 07 - 09, 2008). Symposium on Computer Animation. Eurographics Association, Aire-la-Ville, Switzerland, pp. 193-199.

[SH09] Scherfgen, D. and Herpers, R. 2009. 3D tracking using multiple Nintendo Wii Remotes: a simple consumer hardware tracking approach. In Proceedings of the 2009 Conference on Future Play on @ GDC Canada (Vancouver, British Columbia, Canada, May 12 - 13, 2009). Future Play '09. ACM, New York, NY, pp. 31-32.

[TL06] Tiesel, J. P. and Loviscach, J. 2006. A Mobile Low-Cost Motion Capture System Based on Accelerometers. In Advances in Visual Computing.

[XKCG*08] Xie, L., Kumar, M., Cao, Y., Gracanin, D., and Quek, F. 2008. Data-driven motion estimation with low-cost sensors. Visual Information Engineering, 2008. VIE 2008. 5th International Conference on , vol., no., pp.600-605.

[CMUMocap] CMU Graphics Lab Motion Capture Database. http://mocap.cs.cmu.edu/

[InvenSense] MEMS Gyro | Gyroscope | Motion Plus | Processing - InvenSense Home. http://invensense.com/index.html

[Nintendo] Nintendo of America Inc. Headquarters are in Redmond, Washington. http://www.nintendo.com/wii

[WiiYourself] WiiYourself – gl.tter's native C++ Wiimote library. http://wiiyourself.gl.tter.org/