Efficient Motion Capture Database Indexing on the Cell Processor

by

Bogdan-Cosmin Bucur, BSc

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Computer Science in Interactive Entertainment Technology

University of Dublin, Trinity College

September 2009

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Bogdan-Cosmin Bucur

August 31, 2009

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Bogdan-Cosmin Bucur

August 31, 2009

Acknowledgments

I would like to express my gratitude to my supervisors, Dr.Rozenn Dahyot and William John O'Kane, for their helpful advice and comments on the algorithms and techniques involved in the project. Many thanks also go to my wife, Cristina, for carefully proof-reading this work and pointing out numerous errors. For all remaining shortcomings, as customary, I assume full responsibility.

The data used in this project was obtained from mocap.cs.cmu.edu. The database was created with funding from NSF EIA-0196217.

BOGDAN-COSMIN BUCUR

University of Dublin, Trinity College
September 2009

Efficient Motion Capture Database Indexing on the Cell Processor

Bogdan-Cosmin Bucur
University of Dublin, Trinity College, 2009

Supervisors: Rozenn Dahyot & William John O'Kane

This project aims to implement the system and tools necessary for real-time motion segmentation and recognition. Motion data streams are usually high-dimensional and Principal Component Analysis is used to reduce them to a more manageable size. Singular Value Decomposition is employed on the lower dimensional representation of individual motions to compute feature vectors that will be used in training a hierarchical decision tree of Support Vector Representation and Discrimination Machines. A normalized variant of Edit Distance on Real sequences is employed on recognized motions to compute a classification confidence value and further improve the quality of motion stream segmentation. Finally, these algorithms are implemented in a real-time motion segmentation and recognition library allowing applications running on the Cell/B.E. to identify portions of interest in a potentially live motion data stream (i.e. generated by a Motion Capture (MoCap) device).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The trend in video games design during the past decades has been to present the human player with increasingly complex environments, striving for close to real-world graphics quality and compelling level of physical interaction. This has motivated the hardware manufacturers to invest in developing faster, more powerful processors and graphics cards and has led to the proliferation of gaming consoles, among which the latest arrivals (i.e. Microsoft's Xbox 360 and Sony PlayStation®3) are virtual "beasts" of processing power. During the last five years the returns from solely increasing the clock rate and polygon counts have started to diminish, and it soon became evident that in order to survive, the gaming industry needed to focus its attention towards a different, less-developed area.

The breakthrough came from Nintendo with the launch of the Wii™ gaming console on September 14, 2006; although it was meant to compete primarily with Microsoft's Xbox 360 and Sony's PlayStation®3, the focus was not on performance but on a new way of player interaction. The notable feature of the console stands in its wireless controller (Wii Remote) that comes equipped with an accelerometer capable of detecting 3D movement, allowing for a more natural way of player interaction with a game through the use of hand movements. The novel control scheme combined with reasonable quality graphics makes for a more compelling, immersive game-play and a much more enjoyable user experience. Its increasing popularity stimulated a growing interest in developing more advanced computing interaction interfaces, striving to remove the cumbersome "classic" controller mechanisms.

Computer Vision based user interaction was explored by Sony in several released titles. Among these it is worth mentioning the *Eye Toy*® [1] series, where the player interacts with the game through a USB camera; the player image is segmented from the background and its position on the screen is used to detect collisions with various hotspots, triggering specific game events. More advanced, *The Eye of Judgment*™ [2] uses the camera to locate fiducial markers in the real world and superimpose synthetic 3D characters over them, creating an augmented reality experience for the players. However, because the camera input is 2D, the amount and accuracy of vision based user control for the aforementioned games is limited to rather simple scenarios. Nevertheless, the recently announced ZCam, a low-cost, consumer video camera capable of capturing depth information, is a good candidate to revolutionizing once again the video game industry, availing 3D body tracking as a way of interacting with the virtual world [3].

1.1 Project Goal

The current project aims to implement a library capable of recognizing motions performed by a human subject in real-time, after proper off-line training from a motion database. The applications for such a tool are numerous, ranging from entertainment (e.g. a rhythm and dance game such as *Dance Dance Revolution*, where the player is no longer required to step on switches in the floor, instead his or her body pose is actively tracked by cameras and compared to *golden standard* dance step patterns) to allowing people with disabilities to better interact with a computer. In this implementation the user's body pose is assumed to be tracked in real-time by cameras and converted to joint angle values, similar to the output of a MoCap system; the conversion from camera images to angular joint parameters is ignored, the main focus being on segmenting and recognizing individual motions in the high dimensional data stream.

A natural prerequisite of motion recognition is to train classifiers using sample motion patterns. The patterns are normally taken from a (potentially large) database of pre-recorded motion clips [4] where each clip usually contains several logically distinct motions. In an attempt to simplify working with a large number of motion clips, this project also investigates the possibility of automatically determining similar motion patterns and discovering motion

classes in a collection of clips.

1.2 Document Outline

This document is structured as follows:

- Chapter 2 reviews the previous work in the areas of motion stream segmentation and motion recognition;
- Chapter 3 sets forth the theoretical background and algorithms used in this project;
- Chapter 4 gives a detailed description of the implementation of both off-line database management component running on Windows and the real-time motion classifier running on the Cell Broadband Engine (Cell/B.E.) [5];
- Chapter 5 presents the obtained quality / performance of the implemented motion segmentation and classification algorithms evaluated on a set of motion streams;
- Chapter 6 offers a set of conclusions about the project and also several directions to be explored in future work.

Chapter 2

Previous Work

Motion data is generally given as a topological description of a set of bones (skeleton) together with a set of poses. While the topology remains constant (for instance, the head will always be connected to the neck at an approximatively constant relative offset), the relative bones' orientation (e.g. the pose) changes with time. A typical MoCap device will sample the pose of its human subject 120 times per second; for instance, assuming that the employed (simplified) skeleton consists of only 30 joints each being encoded as a quaternion, the resulting motion data stream will be 120-dimensional with a bit rate of 450 kbit/s. Although at first glance the number of raw dimensions is high, the number of intrinsic dimensions is intuitively lower, due to the constraints of some joints. For instance, as opposed to a shoulder joint that has 3 degrees of freedom, the knees, elbows or wrists can be approximated as hinges having just 1 degree of freedom (DOF).

2.1 Dimensionality Reduction

Principal Component Analysis (PCA) [6, 7] is amongst the most popular dimensionality reduction techniques employed on motion data. Given a data-set, PCA returns a set of orthogonal directions such that the data-set has maximum variance along the first PCA direction, second-maximum variance along the second PCA direction, and so on. The amount of variance on each direction can be used to estimate the quantity of information preserved by

choosing only a subset of the original dimensions. The new number of dimensions is usually chosen such that the quantity of information preserved is above 90%-95%; the reduced dimensinality data-set is obtained by projecting the original data points along the chosen subset of directions returned by PCA [8, 9, 10, 11].

An important observation to be made about the PCA based dimensionality reduction technique is that the number of dimensions needed to preserve the same amount of information grows with the diversity of the motions in the database [12]. To keep the number of dimensions low, PCA can be preceded by a clusterization step that groups together similar motion segments. For each cluster, a different set of PCA directions is determined and used to compute a smaller number of relevant dimensions than it would be possible in the case of applying PCA to the entire database [13].

Although PCA is the default dimensionality reduction algorithm, in the case where the data-set distribution is not Gaussian its result is negatively influenced by the orthogonality requirement on the principal components (basis vectors). *AutoSplit* is a technique that determines a set of hidden variables and basis vectors (not necessarily orthogonal) such that the mutual independence among the hidden variables is maximized [14]. Informally, in a manner similar to Independent Component Analysis (ICA) [15, 16], the original data points are thought of as generated by a number of unknown signal sources (hidden variables), and the algorithm determines a set of directions and signal source parameters, such as the signal emitted by a source is dominant in its associated direction and mostly null in all other directions.

The above mentioned methods have one thing in common: they all discover a lower dimensional embedding in a linear space of the original high-dimensional space. However, the data-set may have non-linear structures invisible to PCA and in this case, a better / more natural set of dimensions can be obtained by applying techniques similar to *Isomap* [17]. The idea behind *Isomap* is to consider the data points on a non-linear manifold and the distance between them defined as the length of the geodesic as opposed to the line segment (Euclidean distance). The geodesic is estimated as the minimum cost path in a graph generated by associating the Euclidean distances as the costs of the edges between each data point and its k-nearest neighbors. *ST-Isomap*, an extension to *Isomap* for data with both spatial and temporal relationships [18], was used to derive behavior vocabularies from time-series data of human motion [19]. However, a downside to *Isomap* based methods is represented by the assumption that the data-set has many samples (in order to be able to estimate the geodesics

accurately), which does not always hold true in the case of human motion databases.

The poses in the original data-set can also be modeled with a Gaussian Process Latent Variable Model (GPLVM), another non-linear dimensionality reduction technique, related to PCA but allowing for a non-linear mapping between the embedded and the original space [20]. The method is better suited for small data-sets and is at the base of Scaled GPLVM, employed by Grochow et al. to model poses in a set used to train a style-based IK system [21].

2.2 Motion Segmentation

Another important problem in motion data stream analysis is how to split the potentially infinite stream into small, manageable chunks (segments), comprised of distinct motions. One of the simplest segmentation techniques is to detect zero crossings of angular velocities [22]; as a result, the obtained motion segments are normally shorter than a logical motion and correspond to motion primitives, such as raising a hand, tilting the head, etc.

Barbič et al. propose three segmentation techniques involving PCA, Probabilistic Principal Component Analysis (PPCA) [23] and respectively Gaussian Mixture Model (GMM) in order to detect cut-points between logically distinct motions (e.g. walking, punching, running) in a stream. In their experiments the PPCA based algorithm provided the closest solution to the ground truth; a possible explanation for this observation is that PPCA tracks changes in the distributions describing the motions rather than the change in dimensionality as in the case of PCA. Furthermore, GMM discards the temporal dependence between frames and also cannot easily run in an unsupervised setting because it needs an initial estimate of the number of Gaussians in the data-set. While there are algorithms to automatically estimate the GMM and choose a model that maximizes some criterion, the authors report sub-optimal number of Gaussians returned by auto-estimation using the Bayesian Information Criterion [12].

The segmentation techniques above are simple to understand and straightforward to implement; more importantly they do not need a training phase and can be applied to virtually any motion stream. There are however more involved algorithms that are first trained to

recognize classes of motions and then used to perform both stream segmentation and motion classification. Billon et al. propose an architecture where training, segmentation and classification can be done on the fly; an agent is trained to recognize only one motion segment and stream segmentation is achieved by allowing all agents to look for the pattern they are trained to recognize in the incoming data stream. The input stream is segmented only if an agent detects the start, middle and end of the motion it was trained for, in the proper order [24]. While this method allows the human operator to train and test the system on the fly, it also scales poorly with the number of motion samples and is therefore not suitable for a constrained computation resources environment such as a games console.

In the case where the input stream contains only brief transitions between known motions, Li et al. propose several techniques that track the output of a classifier to detect motion segments. In the first phase the classifier is trained to recognize motion classes; at runtime the motion stream is divided into increasingly longer segments and each segment is run through the classifier; local maxima in the classifier output represent recognized motions and thus valid segments [25, 26]. Due to the classifier design, the Support Vector Machine (SVM) architecture [26] provides more accurate results but also scales poorly with the number of classes. The method based on similarity search [25] has poorer recognition rate but scales better with the number of training samples through the use of an index [27].

2.3 Motion Recognition

Several approaches exist to the problem of motion recognition. Template matching algorithms evaluate a similarity/distance measure between an unknown data input and predefined, labeled motion sequences in a database. Vlachos et al. note that Euclidean distance is inappropriate to capture similarities in high dimensional data due to sensitivity to noise and small variations in the time axis, and offer as better alternatives the Dynamic Time Warping (DTW) and Longest Common Subsequence (LCSS), extended to handle an arbitrary number of dimensions [28]. Quian et al. use joint angles to represent the motion and use the *Mahalanobis* distance to measure similarity [29]. Li et al. define a non-metric similarity measure (kWAS) based on PCA analysis of both the unlabeled input and the database matrices of motion data, and employ it to perform both segmentation and recognition on the input

stream [25].

An alternative to template matching is to further reduce the number of data dimensions by clusterizing similar body poses / motion primitives and associate unique integer identifiers to each cluster. A Hidden Markov Model (HMM) can be trained for each class of motions so that its output matches the sequence of cluster identifiers of the training motions [30, 31]. Alternatively, since both the motion database and the input sequence are encoded as strings of cluster IDs, one can employ pattern matching algorithms to perform motion recognition [10, 11].

SVM classifiers have also been employed successfully in motion recognition, generally achieving a higher classification rate than HMMs. Li et al. use Singular Value Decomposition (SVD) on the motion matrices to compute the two most representative eigenvectors and concatenate them to compute feature vectors for individual motion classes. The feature vectors are used in training one-versus-one SVMs to discern between each pair of classes in the database and their output is combined to both segment and determine the most probable motion candidate in the input stream [32, 26].

HMM or SVM classifiers alone do not always produce the smooth outputs required by a specific application. Several hybrid approaches were taken in order to tackle this problem. For instance, Sukthankar and Sycara connect the SVM classifier outputs to a hand-coded HMM in order to reduce state transitions caused by false detections [9]. Alternatively, Wan et al. use HMMs as the first stage of classification, further processed by a SVM classifier in order to reduce the ambiguities in the output [33].

Wang et al. employ Gaussian Process Dynamical Models (GPDMs) [34] to determine a mapping from the high dimensional input space to a low dimensional latent space with associated dynamics. The result is a nonparametric model for the dynamic system (in this case the human performing a motion) that also accounts for uncertainty in the model. Although the technique achieves good classification results, the authors report impractical training times (i.e. 9 hours for 289 frames of data) [35].

Chapter 3

Theoretical Background

This chapter sets forth the theoretical foundations needed by the implementation phase of this project. The exposition closely follows the path taken by the motion data, starting from the motion capture device, passing through pre-processing and dimensionality reduction stages, and finally being used in either off-line classifier training or real-time segmentation and classification.

3.1 Motion Capture

Motion Capture (MoCap) is a technique of recording and digitizing the motion of a (generally) human subject. It is widely used in the entertainment industry, medical research or military applications and it is preferred over manual animation of a virtual model in the case of long / complex motions. Existing MoCap devices typically assume the human subject wears a special suit with markers placed at the joints and the motion to be captured takes place in an area surrounded by receivers (e.g. cameras). The MoCap device samples the position of the markers several times per second with respect to the receivers; given the fact that the configuration of the receivers is known and will not change during the recording session, a triangulation algorithm determines the 3D position of each marker. Applying prior knowledge about the relative positions of the markers on the human skeleton, and assuming the latter is non-deformable (e.g. the distance between certain markers is constant, because

they are placed on the same non-deformable bone), the MoCap system can further infer the relative orientation of each bone with respect to its parent.

In what follows, the MoCap device is abstracted as a black-box that can provide a topological description of the subject (i.e. the hierarchy, length and relative positioning of bones), a frame rate at which the motion data is sampled, and finally, for each sampled frame, a vector of relative orientations for each bone in the skeleton plus a translation for the whole model. Relying entirely on MoCap data generated on the fly is impractical, time-consuming and potentially expensive; given a motion category one can usually find several versions of it already recorded and stored in the CMU motion database freely available on-line [4]. At the time of writing, the CMU database provides 2605 recorded sessions belonging to over 140 human subjects as either asf/amc or c3d files. However, for easier visualization and debugging this project employs a MotionBuilder-friendly version of the CMU dataset converted to BVH [36] file format.

3.2 The BVH File Format

This project uses a subset of BVH encoded motion clips in the CMU database for both the training and the performance evaluation phase. BVH files are commonly used in professional animation packages such as MotionBuilder or 3D Studio Max, and consist of two parts: a preamble that describes the topology and proportions of the animated skeleton, and the actual motion data, sampled at fixed time intervals. A sample file is given in Listing 3.1.

The file begins with the keyword **HIERARCHY**, followed by the description of the animated skeleton. A skeleton can have an arbitrary number of root joints (i.e. with no parents), but typically there is only one, defined by the **ROOT** keyword. Once a root joint has been defined, its child joints are defined inside accolades by the **JOINT** keyword. Further children of child joints are defined recursively in a similar manner, inside enclosing accolades.

The relative position of a joint with respect to its parent is given by the parameters of the **OFFSET** keyword. The animated channels of a joint are specified by the **CHANNELS** keyword: the first parameter sets the number of variables and the rest (one of *Xposition*, *Yposition*, *Zposition*, *Xrotation*, *Yrotation*, *Zrotation*) specify the meaning of each variable. Bones are not explicitly defined, but they can be generated between connected joints. A

special non-animated joint is artificially added at the end of each chain of joints with the **End Site** keyword; it cannot have children joints on its own and is only useful in inferring the length of the terminal bones.

Several observations must be made about the animated parameters. First of all, they are relative to the space of the parent joint and in order to compute the orientations of bones in world space at a given time, one must start from the root and propagate the computation towards the leaf joints. Furthermore, the transform matrix of a joint relative to its parent is computed as follows:

$$M_{relative} = R_a R_b R_c + T, \quad \text{with } a, b, c \in \{X, Y, Z\} \tag{3.1}$$

where R_X, R_Y, R_Z are rotation matrices around the X, Y, and respectively the Z axis, $\{a, b, c\}$ is a permutation of the set $\{X, Y, Z\}$ that indicates the order the multiplication must take place and must be identical to the order of rotations in the **CHANNELS** parameters, and T is simply a matrix containing only the translation component initialized to the relative joint offset.

The start of the motion data is marked by the **MOTION** keyword. Following it, the number of frames in the motion is given as the parameter of the **Frames** keyword, and the sample rate as the parameter of the **Frame rate** keyword. In the remainder of the file, all the animated parameters mentioned in the hierarchy are listed in the exact order as they appear in the skeleton description, one frame per line of text.

```
HIERARCHY
ROOT Hips
{
  OFFSET 0.00000 0.00000 0.00000
  CHANNELS 6 Xposition Yposition Zposition Zrotation Yrotation Xrotation
  JOINT LHipJoint
  {
    OFFSET 0 0 0
    CHANNELS 3 Zrotation Yrotation Xrotation
    JOINT LeftUpLeg
    {
      OFFSET 0.17314 -0.47571 2.01064
      CHANNELS 3 Zrotation Yrotation Xrotation
```

```
      End Site
      {
        OFFSET 0.00000 -0.00000 1.04147
      }
    }
  }
  JOINT RHipJoint
  {
    OFFSET 0 0 0
    CHANNELS 3 Zrotation Yrotation Xrotation
    End Site
    {
      OFFSET -0.00000 -0.00000 1.10597
    }
  }
}
MOTION
Frames: 495
Frame Time: .0083333
2.2287 17.6234 -24.8011 0 0 0 0 0 0 -17 0 0 0 0 0
//.............................................
```

Listing 3.1: Sample BVH file

3.3 Smoothing & Filtering BVH File Data

A BVH file stores information about the skeleton (bone names and topology, position of joints relative to the bones, etc.) and also the relative orientations of the bones with respect to their parents at each sampled time-step, encoded as Euler angles. This representation poses a problem when interpolating between frames (this happens when files sampled at different rates are to be combined into the same motion database and the animations within need to be re-sampled at a common frame rate) and is addressed in this project by converting the BVH file to a proprietary format, where the animation is encoded using quaternions. Also, up to cubic Bézier curves are fitted on the data to reduce the joint orientation noise / jitter observed in BVH animations and make them sample rate independent [37].

3.3.1 Bézier Curves

The Bézier curve is a parametric curve $f_n(t)$ that is a polynomial function of the parameter t [38]. The degree n of the polynomial $f_n(t)$ is exactly one less the number of points defining the curve, also called control points. A Bézier curve has the desirable property that it passes exactly through the start and end control points, but generally does not contain the interior control points. Instead, the first segment of the control points polygon is tangent to the start of the curve and in a similar manner, the end of the curve is tangent to the last segment of the same polygon. Formally, given control points P_0, P_1,..., $P_n \in \mathbb{R}$, the Bézier curve associated with the set of control points can be defined as:

$$f_n(t) = \sum_{i=0}^{n} P_i B_i^n(t) \in \mathbb{R}, \quad t \in [0,1] \tag{3.2}$$

where $B_i^n(t)$ are Bernstein polynomials of degree n, defined as:

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i \tag{3.3}$$

A sample cubic Bézier curve is shown in figure 3.1. Directly from the definition the following relations hold true:

$$\begin{aligned}
f_1(t) &= (1-t)P_0 + tP_1 \\
&= (P_1 - P_0)t + P_0 \\
f_2(t) &= (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2 \\
&= (P_0 - 2P_1 + P_2)t^2 - 2(P_0 - P_1)t + P_0 \\
f_3(t) &= (1-t)^3 P_0 + 3(1-t)^2 tP_1 + 3(1-t)t^2 P_2 + t^3 P_3 \\
&= (3(P_1 - P_2) + P_3 - P_0)t^3 + 3(P_0 - 2P1 + P_2)t^2 - 3(P_0 - P_1)t + P_0
\end{aligned} \tag{3.4}$$

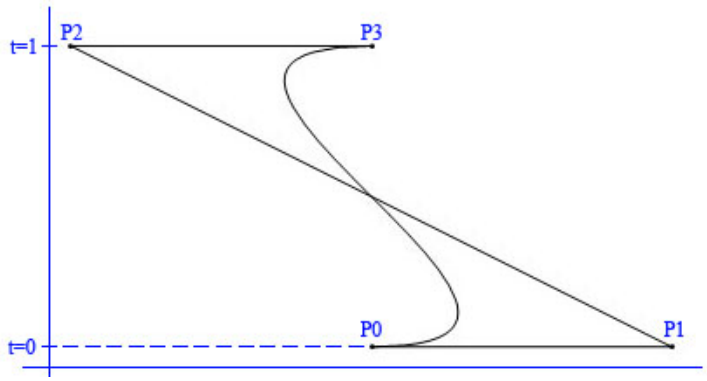

Figure 3.1: Sample cubic Bézier curve defined by four control points.

3.3.2 Curve Fitting

The problem of fitting a Bézier curve to a set of data points is straightforward: given value pairs (t_0, p_0), (t_1, p_1),...,(t_n, p_n), the sought after polynomial $f(t)$ must fulfill the following conditions:

- $f(t_0) = p_0$
- $f(t_n) = p_n$
- minimize $\sum_{i=0}^{n}(p_i - f(t_i))^2$

Since higher degree Bézier curves exhibit too much unwanted oscillations, this presentation will only focus on fitting a cubic polynomial to a data-set. The first two conditions above imply the first and last control points are already known, i.e. $P_0 = p_0$ and respectively $P_3 = p_n$. The remaining problem is to discover the remaining control points P_1, P_2, such that the approximation error is minimized, and it is equivalent to solving the over-determined system of equations:

$$\begin{cases} (1-t_1)^3P_0 + 3(1-t_1)^2t_1P_1 + 3(1-t_1)t_1^2P_2 + t_1^3P_3 & = p_1 \\ (1-t_2)^3P_0 + 3(1-t_2)^2t_2P_1 + 3(1-t_2)t_2^2P_2 + t_2^3P_3 & = p_2 \\ \dots & \\ (1-t_{n-1})^3P_0 + 3(1-t_{n-1})^2t_{n-1}P_1 + 3(1-t_{n-1})t_{n-1}^2P_2 + t_{n-1}^3P_3 & = p_{n-1} \end{cases} \tag{3.5}$$

In matrix form, the system can be written as $A \times P = B$, where:

$$
\begin{aligned}
A &= \begin{bmatrix} 3(1-t_1)^2 t_1 & 3(1-t_1)t_1^2 \\ 3(1-t_2)^2 t_2 & 3(1-t_2)t_2^2 \\ \vdots & \vdots \\ 3(1-t_{n-1})^2 t_{n-1} & 3(1-t_{n-1})t_{n-1}^2 \end{bmatrix} \\
P &= \begin{bmatrix} P_1 \\ P_2 \end{bmatrix} \\
B &= \begin{bmatrix} p_1 - (1-t_1)^3 P_0 - t_1^3 P_3 \\ p_2 - (1-t_2)^3 P_0 - t_2^3 P_3 \\ \vdots \\ p_{n-1} - (1-t_{n-1})^3 P_0 - t_{n-1}^3 P_3 \end{bmatrix}
\end{aligned}
\tag{3.6}
$$

As mentioned before, the system of equations is generally overdetermined and a vector P that minimizes the residual of the solution $\|A \times P - B\|$ can be easily determined as $P = A^+ \times B$, where A^+ represents the pseudo-inverse of A. A^+ can be computed from the Singular Value Decomposition (SVD) of matrix A and suitable algorithms are thoroughly discussed in numerical methods literature [39].

3.3.3 Smoothing & Filtering

Smoothing and filtering of the motion data is obtained by a curve simplification algorithm applied on each of its dimensions such that the difference between the original data and the approximation curves is less than a given (small) threshold. Closely following the exposition of Önder et al. a pseudo-code implementation of the curve simplification technique is given in Algorithm 1.

Input: Set of motion channel values $P_0...P_n$ equally spaced in time
Output: Set of approximating Bézier curves

begin
  while *Enough data points* **do**
    Fit a curve on the data set passing through the first and last points
    Compute $lineLength$ = $\|P_0\bar{P}_n\|$
    Compute $maxError$ = the maximum distance between the curve and $P_0\bar{P}_n$
    Locate the point P_i in the data-set corresponding to $maxError$
    if $\frac{maxError}{lineLength} \leq Threshold$ **then**
      Add curve to the set of approximating Bézier curves
      Stop subdivision
    end
    else
      Repeat algorithm on data-set $P_0...P_i$
      Repeat algorithm on data-set $P_i...P_n$
    end
  end
end

Algorithm 1: Curve simplification algorithm for filtering the BVH files motion data.

3.4 Dimensionality Reduction

The skeleton in the BVH files has 31 pivots (bones) and in the case where each pivot is represented by a quaternion, a motion frame will be encoded as a line vector containing 124 floating point components. As mentioned before, not all dimensions are equally important and a large proportion of pose information can be maintained with only a subset of the original components. To determine the intrinsic dimensionality of the motion data, this project submits the entire motion database to Principal Component Analysis (PCA).

3.4.1 Principal Component Analysis

PCA is a technique that transforms data to a new coordinate system where the new coordinates are ordered decreasingly by the variance of data along them [40]. Mathematically, given a data matrix A with m rows, where each row contains a sample (in this case a motion frame) of a n-dimensional stochastic variable $X \in \mathbb{R}^n$ with zero empirical mean, PCA

computes the decomposition:

$$A = U\Sigma V^T \tag{3.7}$$

such that $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$, $\Sigma = Diag(\sigma_1, \sigma_2, \cdots, \sigma_{min(m,n)})$, U and V are orthogonal, and $\sigma_1 \geq \sigma_2 \geq .... \geq \sigma_{min(m,n)} \geq 0$ are the eigenvalues of the matrix A. The first column of V is the direction in the space where variable X lives, along which the samples in the data matrix A have the greatest variance, and therefore is deemed the first principal component. The second column of V represents the direction in $\mathbb{R}^n$ corresponding to the second largest variance of the data and therefore is called the second principal component, and so on.

The largest variance of the input data is concentrated in the first r directions (preferably $r \ll n$) obtained after PCA and one can consider as relevant only the first r dimensions. The fraction of information preserved by choosing r as the number of relevant dimensions can be estimated by [12]:

$$f_{Quality} = \frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_{i=1}^{n} \sigma_i^2} \tag{3.8}$$

The number r of most relevant dimensions is determined by requiring that $1 \geq f_{Quality} \geq \tau$ where τ is a user selected parameter (i.e. 0.95). The raw $A \in \mathbb{R}^{m\times n}$ data matrix is then transformed to its reduced representation $A_{red} \in \mathbb{R}^{m\times r}$ by the following formula:

$$A_{red} = A \cdot [v_1|v_2|...|v_r] \tag{3.9}$$

where $v_1, \cdots, v_r$ are the first r column vectors of matrix V. To inspect the quality of the dimensionality reduction, one may need to compute the approximate raw data matrix $A_{approx.raw}$ from A_{red}. Due to the fact that v_i are orthogonal vectors, the formula is simply:

$$A_{approx.raw} = A_{red} \cdot [v_1|v_2|...|v_r]^T \tag{3.10}$$

A few remarks must be made about the time complexity of PCA. The decomposition of

data matrix $A = U\Sigma V^T$ is computed by SVD and the algorithms that perform it have a complexity of $O(mn^2)$ [39]. Generally $m \gg n$ - the number of frames is much larger than the number of dimensions - and the computation time can quickly become prohibitive. Fortunately, by defining the matrix M as $M = A^T A$, $M \in \mathbb{R}^{n \times n}$ and computing SVD for it, the computation time can be drastically reduced. The matrix M as defined above is a square matrix of size n, that is also symmetric. The following relation holds true, obtained by replacing A with its SVD decomposition in the definition of M and observing the fact that U is orthogonal:

$$\begin{aligned} M &= A^T A \\ &= (U\Sigma V^T)^T U\Sigma V^T \\ &= V\Sigma U^T U\Sigma V^T \\ &= V\Sigma^2 V^T \end{aligned} \tag{3.11}$$

Knowing that M is symmetric and SVD for symmetric matrices is unique, it is easy to observe that the right eigenvectors of A are the same as the eigenvectors of M and the latter's eigenvalues are the squares of the eigenvalues of A. Therefore, all the data needed by the dimensionality reduction (i.e. right eigenvectors and the squared eigenvalues) can be computed from the SVD of M, having a much smaller complexity of $O(n^3)$.

3.5 Motion Segmentation

Due to the fact that the BVH files usually contain more than one motion, it is necessary to allow for specifying portions of interest inside a motion clip. In order to tackle larger amounts of data, this project implements an automatic Probabilistic PCA [23] based segmentation of motion clips [12] into distinct behaviors (i.e. walking, cartwheel, etc.), while also allowing for manual editing of the cut points.

3.5.1 Probabilistic Principal Component Analysis

PPCA is an extension to PCA where the dimensions higher than r are no longer dropped but modeled with noise. Formally, closely following the exposition of Tipping and Bishop [23], given an observed variable $y \in \mathbb{R}^n$ (in this case a frame of motion data), PPCA relates it to a set of latent variables $x \in \mathbb{R}^r, r < n$, such that:

$$y = Wx + \mu + \varepsilon \tag{3.12}$$

where $W \in \mathbb{R}^{n \times r}$ is a matrix relating x and y, μ is the empirical mean of the observed variable y, and $\varepsilon \sim N(0, \lambda I)$ represents isotropic Gaussian noise. Therefore, PPCA models the observed variable y as having a Gaussian distribution with mean μ and covariance matrix $C = WW^T + \lambda I$. Maximum likelihood estimators for both W and λ are available in closed form, as follows:

$$\begin{aligned} \lambda &= \frac{1}{n-r} \sum_{i=r+1}^{n} \lambda_i \\ W &= V_r (\Lambda_r - \lambda I)^{\frac{1}{2}} R \end{aligned} \tag{3.13}$$

where λ_i are the eigenvalues of the sample covariance matrix S ordered decreasingly by magnitude, $V_r \in \mathbb{R}^{n \times r}$ is the matrix having the column vectors v_i equal to the corresponding eigenvectors of matrix S, $\Lambda_r = Diag(\lambda_1, \cdots, \lambda_r)$ and R is an arbitrary $r \times r$ orthogonal rotation matrix.

Following the notations in the previous section, the sample covariance matrix S of a motion matrix $A \in \mathbb{R}^{m \times n}$ containing observations of the variable y (motion frame) as line vectors is given by:

$$S = \frac{1}{m-1} A^T A = \frac{1}{m-1} M = \frac{1}{m-1} V \Sigma V^T \tag{3.14}$$

where $V \Sigma V^T$ is the SVD of M. Let the eigenvalues of matrix $M = A^T A$ be σ_i^2. By

virtue of uniqueness of SVD for the case of symmetric matrices, the eigenvalues of S will be $\lambda_i = \frac{1}{m-1}\sigma_i^2$ and the eigenvectors of S will be identical to those of M. Substituting the formula for λ_i and Equation 3.13 in the definition of covariance matrix, the following holds true:

$$\begin{aligned}
C &= WW^T + \lambda I \\
&= V_r(\Lambda_r - \lambda I)^{\frac{1}{2}} R R^T (\Lambda_r - \lambda I)^{\frac{1}{2}} V_r^T + \lambda I \\
&= V_r(\Lambda_r - \lambda I) V_r^T + \lambda I \\
&= \sum_{i=1}^{r} v_i v_i^T (\lambda_i - \lambda) + \sum_{i=1}^{n} v_i v_i^T \lambda \\
&= \sum_{i=1}^{r} v_i v_i^T \lambda_i + \sum_{i=r+1}^{n} v_i v_i^T \lambda \\
&= V Diag(\lambda_1, \cdots, \lambda_r, \lambda, \cdots, \lambda) V^T \\
&= \frac{1}{m-1} V \Sigma^* V^T
\end{aligned} \tag{3.15}$$

where $\Sigma^* = Diag(\sigma_1^2, \cdots, \sigma_r^2, \sigma^2, \cdots, \sigma^2)$ and $\sigma^2 = \frac{1}{n-r}\sum_{i=r+1}^{n}\sigma_i^2$.

3.5.2 Detecting Logically Distinct Motion Segments

The idea behind PPCA based motion segmentation is to create a probabilistic model for a number of motion frames (i.e. a segment) and detect how well the following frames are predicted by the model. If the model behaves well, the frames are added to the current segment and the model parameters are re-estimated. In the opposite case, the frames are considered as the start of a new, logically distinct motion segment, a cut-point is generated at the current position in the motion stream, and the whole process repeats starting from the cut-point [12]. Formally, assuming that PPCA was used to model the first K frames in a motion stream and compute a covariance matrix C as in Equation 3.15, the likelihood of frames $K+1$ to $K+T$ belonging to the model is estimated by computing an average Mahalanobis distance H:

$$H = \frac{1}{T} \sum_{i=K+1}^{K+T} (x_i - \bar{x})^T C (x_i - \bar{x}) \tag{3.16}$$

where x_i is the motion vector at frame i and $\bar{x}$ is the mean of the motion vector in the first K frames. Segments are determined by tracking the peaks and valleys of H along the motion stream and inserting cut-points whenever a valley is followed by a peak. A sample pseudo-code implementation is given in Algorithm 2.

Input: Motion stream, T = number of look-ahead frames, D = frame increment
Output: Set of cut-points

begin
  $K = T$
  while *Motion stream not empty* **do**
    $\bar{x} = \frac{1}{K} \sum_{i=1}^{K} x_i$
    Compute C using Equation 3.15
    Compute H using Equation 3.16
    if H *is at peak and last valley is far enough* **then**
      Append cut-point to returned set
      Consume K frames from the motion stream
      $K = T$
    end
    $K = K + T$
  end
end

Algorithm 2: PPCA based approach to motion stream segmentation.

3.6 Clusterizing Similar Motions

After identifying portions of interest in the database motion clips, logically similar motion samples must be grouped together into motion classes so that classifiers can be trained to recognize them. Again, in the spirit of minimizing user intervention, this project implements an auto-clusterization technique based on the $kWAS$ similarity measure [25]. Given two motion matrices $M_a = A_a^T A_a$, $M_b = A_b^T A_b$, the $kWAS$ similarity measure between M_a and

M_b is defined as:

$$kWAS(M_a, M_b) = \frac{1}{2} \sum_{i=1}^{k} \left(\left(\frac{\sigma_i}{\sum_{j=1}^{n} \sigma_j} + \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j} \right) |u_i \cdot v_i| \right) \tag{3.17}$$

where σ_i, λ_i are the eigenvalues of the matrices M_a and respectively M_b, u_i, v_i are the eigenvectors of the same motion matrices corresponding to the eigenvalues σ_i, and respectively λ_i, and k is an integer that controls how many (eigenvalue, eigenvector) pairs are considered, $1 < k < n$. The insight for this measure is that similar motions will have similar principal directions / variations along these principal directions. Using the same reasoning as in the case of dimensionality reduction, not all principal directions are equally important, and we can safely limit the number entering the similarity measure to only the most representative k directions corresponding to the k largest eigenvalues.

In order to discover similar motion samples, each pair of motion segments has its $kWAS$ measure computed and pairs with a similarity score larger than a threshold τ (i.e. τ = 0.95) are grouped together in the same motion class. A lower value for τ generates a smaller number of classes, where more logically distinct motions are merged together, while a higher value causes the opposite behavior, with more than one class for the same logical motion class.

3.6.1 Rand Index

The performance of the clusterization based on the $kWAS$ measure is evaluated against the ground truth using the *Rand index*, a normalized measure of the number of correct clusterization decisions [41]. Each pair of motion samples can be either in the same cluster or in distinct clusters. Based on whether the pair naturally belongs to the same cluster, 4 situations may occur:

- True Positive (TP): the pair is in the same cluster and naturally belongs to the same cluster;
- False Positive (FP): the pair is in the same cluster and naturally belongs to distinct clusters;

		Ground Truth	
		Positive	**Negative**
Observed Outcome	**Positive**	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

Table 3.1: Sample confusion matrix.

- True Negative (TN): the pair is in distinct clusters and naturally belongs to distinct clusters;
- False Negative (FN): the pair is in distinct clusters and naturally belongs to the same cluster;

The four values are commonly organized into a confusion matrix, as shown in Table 3.1. Given the above notations, the *Rand index* is formally defined as:

$$RI = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.18}$$

It is easy to see that the *Rand index* takes values between 0 and 1. A high value (close to 1) indicates a clusterization solution close to the ground truth, while a value close to 0 indicates almost random random behavior of the clusterization algorithm.

3.7 Motion Classification

SVM are a powerful approach to supervised learning problems such as classification [42, 43] and were successfully applied to motion data. The theory behind SVMs is based on the idea of finding an optimal separation hyperplane between object classes, in the sense of maximizing the in-between-class margin.

3.7.1 Support Vector Machines

Formally, given m data samples $x_1, x_2, ..., x_m \in \mathbb{R}^n$ and a set of class labels $y_1, y_2, ..., y_m$ such that sample x_i is associated with the class label y_i and $y_i \in \{\pm 1\}$, the problem of

classification is to create a function $f(x) \rightarrow \{\pm 1\}$ such that f best predicts the class label of $x \in \mathbb{R}^n$.

The SVM approach to classification is to consider the training samples $x_1, x_2, ..., x_m$ linearly separable in a higher, possibly infinite dimensional space, and search for f as the equation of the separating hyperplane that maximizes the distance (margin) between the two classes. Given the (usually unknown) mapping $\phi(x) \rightarrow S$ that converts the data points $x \in \mathbb{R}^n$ to the higher dimensional space S where the classes are linearly separable, the decision function f as defined by the SVM formulation is:

$$f(x) = w^T \cdot \phi(x) + b \tag{3.19}$$

such that $f(x) \geq 1$ for $\forall x_i$ that have label $y_i = 1$ and $f(x) \leq -1$ for $\forall x_i$ that have label $y_i = -1$. It is easy to notice that f is the equation of a (separating) hyperplane in S, w is the hyperplane normal, and b is the hyperplane offset (scalar). As emphasized before, f is determined by maximization of the margin between classes that is inverse proportional to the norm of w. Therefore, the following problem must be solved (*maximum margin SVM*) in order to determine the SVM decision function:

$$\begin{aligned} minimize &: \frac{1}{2} w^T w \\ s.t. &: y_i(w^T \phi(x_i) + b) \geq 1, i \in \{1, ..., m\} \end{aligned} \tag{3.20}$$

Of course, the assumption that a separating hyperplane exists between classes can be unrealistic and it can be relaxed by incorporating outliers (*1-norm soft-margin SVM*):

$$\begin{aligned} minimize &: \frac{1}{2} w^T w + C \sum_{i=1}^{m} \varepsilon_i \\ s.t. &: y_i(w^T \phi(x_i) + b) \geq 1 - \varepsilon_i \\ &: \varepsilon_i \geq 0, i \in \{1, ..., m\} \end{aligned} \tag{3.21}$$

The first step in solving the SVM minimization problem is to form the Lagrangian. For the 1-norm soft-margin SVM, the Lagrangian is:

$$L = \frac{1}{2}w^T w + C\sum_{i=1}^{m}\varepsilon_i - \sum_{i=1}^{m}\alpha_i(y_i(w^T\phi(x_i)+b) - 1 + \varepsilon_i) - \sum_{i=1}^{m}\beta_i\varepsilon_i \tag{3.22}$$

The extremum point is found by setting partial derivatives with respect to w, ε_i and b to zero:

$$\begin{aligned}
\frac{\partial L}{\partial \varepsilon_i} &= C - \alpha_i - \beta_i = 0 \\
\frac{\partial L}{\partial b} &= \sum_{i=1}^{m}\alpha_i y_i = 0 \\
\frac{\partial L}{\partial w} &= w - \sum_{i=1}^{m}\alpha_i y_i \phi(x_i) = 0
\end{aligned} \tag{3.23}$$

Replacing the above condition in the Lagrangian, the initial SVM problem is transformed into its dual:

$$\begin{aligned}
maximize : & \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j \phi(x_i)\phi(x_j) - \sum_{i=1}^{m}\alpha_i \\
s.t. : & \ 0 \le \alpha_i \le C \\
: & \sum_{i=1}^{m}\alpha_i y_i = 0
\end{aligned} \tag{3.24}$$

As mentioned before, the mapping ϕ to space S is not usually known, but since in the dual formulation the vectors in the unknown space S are only involved in dot products, the problem can be solved by the kernel trick. The trick consists in replacing the dot products $\phi(x_i)\phi(x_j)$ with a kernel function $K(x_i, x_j)$ that has the mathematical properties of inner product in the higher dimensional space S. A common kernel function is the Gaussian RBF, defined as:

$$K(x_i, x_j) = exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \tag{3.25}$$

The normal vector w to the maximum margin separating hyperplane is a linear combination of training sample mappings in S. From the Karush-Kuhn-Tucker conditions, if:

- $\alpha_i = 0$, $\phi(x_i)$ is on the correct side of the plane;
- $0 < \alpha_i < C$, $\phi(x_i)$ is on the plane boundary and is called a *support vector*;
- $\alpha_i = C$, $\phi(x_i)$ is an outlier.

As soon as the α_i factors are determined, the plane offset b can be estimated as [39]:

$$b = \frac{\sum_{i=1}^{m} \alpha_i (C - \alpha_i)(y_i - w^T \phi(x_i))}{\sum_{i=1}^{m} \alpha_i (C - \alpha_i)} \tag{3.26}$$

3.7.2 Support Vector Representation and Discrimination Machines

SVMs have been designed to solve binary classification problems, an area where they exhibit very good accuracy. However, in real-world situations, it is usually necessary to discern between class and non-class samples and reject the latter as non-classifiable. In this respect, the original SVM formulation does not produce accurate results and alternative definitions were created to tackle this problem. The current implementation uses SVRDM to achieve binary classification with good rejection of non-class samples [44]. The kernel function is fixed to Gaussian RBF, the offset term b is dropped since for the particular choice of kernel it does not have a major influence on the solution, and the SVRDM for a two class classification problem seeks to find functions $f_1(x) = h_1^T \phi(x)$, $f_2(x) = h_2^T \phi(x)$, such that:

$$
\begin{aligned}
minimize : & \frac{\|h_1\|^2}{2} + C\sum_{i=1}^{m}\varepsilon_i \\
s.t. : & \varepsilon_i \geq 0 \\
& : y_i h_1^T \phi(x_i) \geq +T - \varepsilon_i, \forall y_i = +1 \\
& : y_i h_1^T \phi(x_i) \geq -t - \varepsilon_i, \forall y_i = -1
\end{aligned} \tag{3.27}
$$

and in the same time

$$
\begin{aligned}
minimize : & \frac{\|h_2\|^2}{2} + C\sum_{i=1}^{m}\xi_i \\
s.t. : & \xi_i \geq 0 \\
& : -y_i h_2^T \phi(x_i) \geq +T - \xi_i, \forall y_i = -1 \\
& : -y_i h_2^T \phi(x_i) \geq -t - \xi_i, \forall y_i = +1
\end{aligned} \tag{3.28}
$$

It is easy to see that a SVRDM is basically composed of two SVMs, where the ± 1 thresholds were replaced by T and t respectively. If $T = 1, t = -1$, the SVRDM decision functions will be identical, $h_1 = h_2$ and the classification behavior is that of a regular SVM. However, if t is increased towards T, each decision function will tighten around the samples of the class it represents; therefore, combining the decision functions in the SVRDM has the desirable effect of improving the rejection capability over a conventional SVM. It is worth noting that solving the above formulation for a SVRDM implies solving two identical quadratic programming problems with box constraints. The first Quadratic Programming Problem (QPP) solves for h_1 that best represents class A with respect to class B, and the second QPP solves for h_2 that best represents class B w.r.t. class A. Since the training samples are the same, the problems differ only by the threshold values, the latter QPP simply replacing T with t.

Because the term b found in regular SVMs was dropped from the SVRDM definition, the dual problem for a SVRDM is easier to solve. Indeed, the Lagrangian for the first half of a

SVRDM is:

$$L = \frac{1}{2}h_1^T h_1 + C\sum_{i=1}^{m}\varepsilon_i - \sum_{i=1}^{m}\alpha_i(y_i h_1^T \phi(x_i) - \tau_i + \varepsilon_i) - \sum_{i=1}^{m}\beta_i\varepsilon_i \tag{3.29}$$

where $\tau_i = T$, if $y_i = 1$ and $-t$ otherwise. By setting again the partial derivatives of the Lagrangian w.r.t. h_1 and ε_i to zero,

$$\begin{aligned} \frac{\partial L}{\partial \varepsilon_i} &= C - \alpha_i - \beta_i = 0 \\ \frac{\partial L}{\partial h_1} &= h_1 - \sum_{i=1}^{m}\alpha_i y_i \phi(x_i) = 0 \end{aligned} \tag{3.30}$$

the dual of the SVRDM problem becomes:

$$\begin{aligned} maximize : \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j K_{ij} - \sum_{i=1}^{m}\alpha_i\tau_i \\ s.t. : 0 \leq \alpha_i \leq C \end{aligned} \tag{3.31}$$

As mentioned before, this is a QPP problem with box constraints, which can be solved by successive over-relaxation (SOR). This project implements a SOR algorithm based on the code from Numerical Recipes [39] with a modified relaxation replacement [45]:

$$\alpha_i^{k+1} = \alpha_i^k - \frac{1}{K_{ii}}(\sum_{j=1}^{i-1}K_{ij}\alpha_j^{k+1} + \sum_{j=i}^{m}K_{ij}\alpha_j^k - \tau_i) \tag{3.32}$$

A sample pseudo-code implementation of the SOR technique that solves the SVRDM problem is given in Algorithm 3. Note that the procedure must be called twice, once for each half of the SVRDM.

Input: Set of training samples x_1,...,x_m
Output: Vector of weights α_1,...,α_m

begin
  Set $\alpha_i \leftarrow 0, \forall i \in \{1, \cdots, m\}$
  while $\exists i$ *s.t.* $|\alpha_i^k - \alpha_i^{k-1}| > threshold$ **do**
    for $i \leftarrow 1$ **to** m **do**
      Compute α_i^{k+1} from Equation 3.32
      if $\alpha_i^{k+1} < 0$ **then** $\alpha_i^{k+1} \leftarrow 0$
      if $\alpha_i^{k+1} > C$ **then** $\alpha_i^{k+1} \leftarrow C$
    end
  end
end

Algorithm 3: SOR algorithm for training a SVRDM

3.7.3 Estimation of SVRDM Parameters

An important problem in proper training of SVRDMs (and more generally, SVMs) to discern between two classes of objects, is represented by the choice of parameters. In the case where the kernel is the Gaussian RBF function as defined by Equation 3.25, there are only two parameters that need be considered: C, the outlier penalty, and σ, the kernel width. Keerthi and Lin provide a good analysis on the asymptotic behavior of SVMs with Gaussian kernel, pointing out the cases where the solution manifests undesired traits such as underfitting (the SVM recognizes all space as belonging to the trained class) or overfitting (the SVM recognizes the training samples perfectly but virtually nothing else) [46].

Traditionally, both C and σ are estimated by testing the generalization ability of various SVM (one for each combination of (C, σ)) by cross-validation and choosing the model with the best accuracy. However, parameter estimation by cross-validation is very time-consuming, and in this project an automatic method for estimation of σ is employed [44]. The penalty incurred by incorporating outliers has a fixed value of 10, following the observation in the original paper also confirmed by personal tests, that its choice influences the classifier performance only to a very small amount.

The technique for auto-estimation of σ is based on another concept introduced by Yuan

and Casasent, Support Vector Representation Machine (SVRM), briefly discussed in what follows.

Support Vector Representation Machines

A SVRM is basically a variant of a one-class SVM with no bias term and a Gaussian kernel; its purpose is to best learn to distinguish samples belonging to a class with respect to the rest of the space, in other words to create a representation of a class. The difference from a common SVM is that the training process requires samples only from the class that needs to be learned, samples from the non-class are assumed to be not available. Formally, a SVRM strives to determine a decision function f such that $f(x) \geq T$ for each x belonging to the represented class and $f < T$ otherwise. Given the training samples x_1,...,x_m, the SVRM decision function is determined by solving the following minimization problem:

$$\begin{aligned} minimize : & \frac{\|h\|^2}{2} + C\sum_{i=1}^{m}\varepsilon_i \\ s.t. : & \ \varepsilon_i \geq 0 \\ & : h^T\phi(x_i) \geq +T - \varepsilon_i, \forall i \in \{1, \cdots, m\} \end{aligned} \tag{3.33}$$

The problem is almost identical to the one of SVRDM, although somewhat simpler, and the reasoning is similar; the associated Lagrangian is given by:

$$L = \frac{1}{2}h^T h + C\sum_{i=1}^{m}\varepsilon_i - \sum_{i=1}^{m}\alpha_i(h^T\phi(x_i) - T + \varepsilon_i) - \sum_{i=1}^{m}\beta_i\varepsilon_i \tag{3.34}$$

Again, the partial derivatives of the Lagrangian w.r.t. h and ε_i need to be zero at the point of extremum:

$$\frac{\partial L}{\partial \varepsilon_i} = C - \alpha_i - \beta_i = 0$$
$$\frac{\partial L}{\partial h} = h - \sum_{i=1}^{m} \alpha_i y_i \phi(x_i) = 0 \tag{3.35}$$

And the dual of the SVRM problem becomes:

$$maximize : \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j K_{ij} - T \sum_{i=1}^{m} \alpha_i$$
$$s.t. : 0 \leq \alpha_i \leq C \tag{3.36}$$

It is easy to see the SVRM problem is indeed a simplified version of the SVRDM discussed above, with $y_i = 1$ and $\tau_i = T$ for $\forall i \in 1, \cdots, m$. The same SOR technique given by Algorithm 3 is used to compute the vector of weights α_i.

Estimation of σ

Following the work of Yuan and Casasent [44] a sample pseudo-code implementation of the auto-estimation of σ is given in Algorithm 4. In the first part, the algorithm exploits the potential clusters in the training data and estimates its approximate bounding boundary. A cluster radius R is empirically estimated as a factor (2.2 in the original formulation) of the average nearest-neighbor distance between the training samples, and m SVRMs are trained for samples inside spheres of radius R centered around each sample. The σ value for the samples inside a sphere is easily estimated as the average distance from its centroid to each sample.

Further on, the algorithm attempts to estimate the SVRM bounding boundary around training data in feature space. The bounding boundary of a SVRM is a subset of the training samples x_i for which the SVRM decision function f is equal to the threshold value T. Since for the vectors in the training set that are not support vectors (i.e. $\alpha_i = 0$) the decision function

is always $f(x_i) > T$, the bounding boundary is a subset of the SVRM support vectors. Knowing that support vectors with $\alpha_i = C$ are associated with outliers and therefore $f(x_i) < T$ (a direct consequence of the KKT conditions), the bounding boundary is actually the set of support vectors x_k such that $0 < \alpha_k < C$. To compute the estimation of bounding boundary, for each of the m SVRMs, the algorithm tests whether the sample used to generate the SVRM training sphere is located on its local bounding boundary. If it is, then the sample is probably on the bounding boundary of the SVRM representing the entire set of training samples and is therefore included in the estimation.

After obtaining an estimation of the SVRM bounding boundary, several values of σ are used to train a SVRM for the entire set of samples. The best σ value is the one that produces a SVRM bounding boundary that best matches the previously estimated bounding boundary. The match cost is computed as the cardinality of the set intersection between the current σ SVRM bounding boundary and the estimated boundary. Testing increasingly larger values of σ is also important because it prevents choosing too small values that produce SVRMs with many support vectors, and thus a high matching cost, at the expense of poorer generalization ability.

It is worth noting that up to this point the algorithm only estimates a σ value for a SVRM, in other words, for a single class classifier. To extend the procedure to a two class SVRDM, Yuan and Casasent first apply the algorithm for the samples in the first class and then for the samples in the second class; the SVRDM σ value is simply chosen as the arithmetic mean of the σ values obtained for each class [44].

3.7.4 Computing Feature Vectors from Motion Segments

Training a SVRDM is done with feature vectors that can be thought as same length summaries of the class samples to be learned. In this case, the samples consist in variable length motion segments, represented by the motion matrices A_i. Following the argumentation in Li et al., for a given motion sample with matrix A_i, the associated feature vector FV_i is defined as:

Input: Set of training samples x_1,...,x_m
Output: Value of σ

begin
  for $i \leftarrow 1$ **to** m **do** $d_i = min_{j \in \{1,\cdots,m\}, j \neq i} \|x_i - x_j\|$
  $\bar{d} \leftarrow \frac{1}{m} \sum_{i=1}^{m} d_i$
  $R \leftarrow 2.2\bar{d}$
  $EstimatedBoundary \leftarrow \emptyset$
  for $i \leftarrow 1$ **to** m **do**
    $S \leftarrow \{x_j | R > \|x_i - x_j\|\}$
    $c \leftarrow \frac{1}{|S|} \sum_{x_j \in S} x_j$
    $r \leftarrow \frac{1}{|S|} \sum_{x_j \in S} \|x_j - c\|$
    Train SVRM for samples in S with $\sigma \leftarrow r$
    if $x_i \in$ *SVRM Bounding Boundary* **then**
      $EstimatedBoundary = EstimatedBoundary \cup \{x_i\}$
    end
  end
  $\sigma_{best} \leftarrow \sigma_{min}$
  $score_{best} \leftarrow 0$
  for $\sigma \leftarrow \sigma_{min}$ **to** σ_{max} **do**
    Train SVRM for all m samples
    $score \leftarrow |SVRM_{boundary} \cap EstimatedBoundary|$
    if $score \geq score_{best}$ **then**
      $\sigma_{best} \leftarrow \sigma$
      $score_{best} \leftarrow score$
    end
  end
  return σ_{best}
end

Algorithm 4: Estimation of σ for a SVRM

$$FV_i = \left[\frac{\sigma_1}{\sum_{j=1}^{n} \sigma_j} \cdot v_1^T | \cdots | \frac{\sigma_k}{\sum_{j=1}^{n} \sigma_j} \cdot v_k^T \right] \tag{3.37}$$

where $v_1, ..., v_k$ are the dominant right eigenvectors of matrix A_i corresponding to the eigenvalues $\sigma_1, \cdots, \sigma_k$, and k selects the number of (eigenvalue, eigenvector) pairs used in the concatenation [26]. The insight for this definition resembles that of the $kWAS$ measure mentioned above: similar motions will have dominant directions very close, and the 1-normalized eigenvalues are used to scale the importance of the eigenvectors.

Li et al. noticed that since the eigenvalues of matrix A_i are positive, its right eigenvectors can have an opposite sign as long as the corresponding left eigenvectors also have opposite signs. The authors propose a procedure to ensure a set of eigenvectors are properly oriented, and an implementation in pseudo-code is given in Algorithm 5, where the number of training samples is denoted by m, and the motion segments given in the form of motion matrices $A_i, i \in \{1, \cdots, m\}$ have the first k right singular vectors symbolized by $v_1^i, \cdots, v_k^i$.

Input: Set of eigenvectors $v_j^i, i \in \{1, \cdots, m\}, j \in \{1, \cdots, k\}$
Output: The same set of (potentially altered) eigenvectors

begin
  for $j \leftarrow 1$ **to** k **do**
    $\bar{v_j} \leftarrow \frac{1}{m} \sum_{i=1}^{m} v_j^i$
    $S_j \leftarrow \left[v_j^1 - \bar{v_j} | \cdots | v_j^m - \bar{v_j} \right]^T$
    $U \Sigma W^T \leftarrow SVDDecomposition(S_j)$
    $w_1 \leftarrow$ first right eigenvector of matrix S_j
    for $i \leftarrow 1$ **to** m **do** **if** $w_1 \cdot v_j^i < 0$ **then** $v_j^i = -v_j^i$
  end
end

Algorithm 5: Algorithm ensuring that a set of eigenvectors have consistent signs.

3.7.5 Non-Linear Dimensionality Reduction

Due to the fact that in the case of motion recognition the number of feature vector dimensions is high compared to the number of training samples, there is a risk of over-training / over-

fitting when computing the SVRDM decision function. This implementation employs a non-linear dimensionality reduction technique on the training samples, that uses class centroids in high dimensional space as a basis to compute new 2-dimensional feature vectors that are well separated [47]. Formally, given two object classes C_1, C_2, a set of n-dimensional samples $x_i^j \in C_j, i \in \{1, \cdots, n_j\}, j \in \{1, 2\}$, and a generally unknown mapping Ψ from sample space to a N-dimensional feature space, Park and Park first define the centroid matrix [47]:

$$C = \left[\frac{1}{n_1} \sum \Psi(x_i^1), \frac{1}{n_2} \sum \Psi(x_i^2) \right] \in \mathbb{R}^{N \times 2} \tag{3.38}$$

Matrix C can be factored using QR-decomposition and let the *thin QR factorization* of C be $C = Q_1 R_1$, where $Q_1 \in \mathbb{R}^{N \times 2}$ and $R_1 \in \mathbb{R}^{2 \times 2}$. It is easy to see that matrix $C^T C$ can be entirely computed using the kernel trick, since it involves only dot products between vectors mapped to potentially unknown feature space:

$$C^T C = \begin{bmatrix} \frac{1}{n_1^2} \sum K(x_i^1, x_j^1) & \frac{1}{n_1 n_2} \sum K(x_i^1, x_j^2) \\ \frac{1}{n_1 n_2} \sum K(x_i^1, x_j^2) & \frac{1}{n_2^2} \sum K(x_i^2, x_j^2) \end{bmatrix} \tag{3.39}$$

Also, let the kernel function K be the Gaussian RBF; $C^T C$ is therefore symmetric positive definite and it allows for a Cholesky decomposition $C^T C = L L^T$, with $L \in \mathbb{R}^{2 \times 2}$ a lower triangular matrix. Given the fact that matrix C has a full column rank, the R_1 matrix from the QR factorization will be equal to L^T [48].

Given a vector $x \in \mathbb{R}^n$, the non-linear dimensionality reduction technique presented here computes the reduced vector $\hat{x} \in \mathbb{R}^2$ as an orthogonal transformation of vector x in feature space:

$$\begin{aligned} \hat{x} &= Q_1^T \Psi(x) = (R_1^T)^{-1} R_1^T Q_1^T \Psi(x) = (R_1^T)^{-1} (Q_1 R_1)^T \Psi(x) = L^{-1} C^T \Psi(x) \\ &= L^{-1} \begin{bmatrix} \frac{1}{n_1} \sum K(x_i^1, x) \\ \frac{1}{n_2} \sum K(x_j^2, x) \end{bmatrix} \end{aligned} \tag{3.40}$$

The above algorithm uses all training samples in a set in order to compute the cluster matrix

C and therefore may have performance issues for large training sets. However, some of the samples in the training set can be omitted from the computation of the cluster matrix without significantly altering the final result. The representative vectors that describe the centroid of a class are determined here by computing the minimum enclosing hypersphere around its samples in feature space [49]. Formally, the problem is formulated as follows: determine point a such that radius R of a hypersphere containing the points in feature space is minimized:

$$\begin{aligned} minimize : & R^2 \\ s.t. : & \|\psi(x_i) - a\|^2 \leq R^2 + \varepsilon_i \\ : & \varepsilon_i \geq 0 \end{aligned} \tag{3.41}$$

Applying the same reasoning as in the case of SVM, i.e. computing the Lagrangian, imposing the extremum point constraints, performing the kernel trick in the particular case of Gaussian RBF kernel, the dual problem becomes:

$$\begin{aligned} maximize : & \sum_{i,j=1}^{n} \alpha_i \alpha_j K_{ij} \\ s.t. : & \sum_{i=1}^{n} \alpha_i = 1 \\ : & 0 \leq \alpha_i \leq C \end{aligned} \tag{3.42}$$

where again C is the penalty incurred by the incorporation of outliers, α_i are Lagrange multipliers and the sought-for solution a is given by $a = \sum_{i=1}^{n} \alpha_i \psi(x_i)$. The dual problem is again a QPP problem with box constraints and an additional equality constraint. The support vectors for the centers of each class are thus determined as the training samples with non-null coefficients α_i and only these are further considered by the non-linear dimensionality reduction algorithm when computing the transformation to 2-dimensional feature vectors.

3.7.6 Multi-Class Classifier Design

As mentioned before, SVMs have been developed mainly for binary classification. Several approaches exist for solving multi-class classification problems; for instance, one can employ n one-versus-rest binary classifiers. This approach involves training the n classifiers such that they can recognize the desired class and reject the other $n-1$. The one-vs-rest classifiers do not usually exhibit the best possible performance due to the imbalanced training set (it can be expected that the number of negative training samples will be on average $n-1$ times the number of positive training samples) [50, 51], and better classification accuracy is obtained by training binary classifiers for all $n(n-1)/2$ pairs of classes. However, this solution also has some issues: first of all, the classification time is quadratic with the number of classes, making it unpractical for problems with a large number of classes, and second, given a sample $x \in C_1$ to be classified, there will only be $n-1$ classifiers that have actually 'seen' the sample in training and can make an informed decision, the other $(n-2)(n-1)/2$ have a big chance of miss-classification due to the poor rejection ability of the conventional SVM.

A solution with both a reduced classification complexity and better accuracy due to improved rejection capacity is the SVRDM based hierarchical classifier [52]. The idea is to create a classification tree, where each node can discern between two sets of classes (macro-classes). The classification of sample x starts from the root node; a SVRDM output is evaluated to determine whether the sample belongs to the left child node macro-class, the right child node macro-class, or is an unknown (non-class) sample. If the sample is recognized, the classification process continues in a recursive manner, following the edge to the child node that is trained to classify the sample. A complete classification (from root to leaf nodes) takes about $\log_2 n$ SVRDM evaluations, less than the one-vs-all or one-vs-one methods mentioned above.

Wang and Casasent [52] create the hierarchical classifier tree such that at each node the two macro-classes to be recognized are best separated in feature space. A pseudo-code implementation of this technique is given by Algorithm 6, where N denotes the number of classes, $x_i^j, i \in \{1, \cdots, n_j\}, j \in \{1, \cdots, N\}$ represent the training samples and Ψ is a generally unknown mapping from sample space to feature space.

Wang and Casasent show that Algorithm 6 can be run without explicitly knowing the map-

Input: Training samples x_i^j
Output: Partitioning of the input classes into two best separated macro-classes

begin
  Estimate σ for the set of all training samples using Algorithm 4
  for $j \leftarrow 1$ **to** N **do**
    Train SVRM for class j and compute $h_j = \sum_{i=1}^{n_j} \alpha_i^j \Psi(x_i^j)$
  end
  Assign all h_j randomly into clusters Q_1, Q_2
  repeat
    for $j \leftarrow 1$ **to** N **do**
      Let centroids in feature space be
      $\Psi(m_k) = \frac{1}{|Q_k|} \sum_{h_i \in Q_k} h_i, k \in \{1, 2\}$
      Let the distances to centroids be
      $d_k = \|h_j - \Psi(m_k)\|, k \in \{1, 2\}$
      Assign h_j to cluster Q_i, such that $i = argmin_{k \in \{1,2\}} d_k$
    end
  until *Clustering solution converged*
end

Algorithm 6: Computing the partition into two best separated macro-classes.

ping Ψ to feature space, because the distances d_k can be computed using the kernel trick. Indeed, the formula for distance is reduced to computing a sum of products $h_i^T h_j$:

$$\begin{aligned} d_k &= \|h_j - \Psi(m_k)\|^{\frac{1}{2}} \\ &= \|h_j - \frac{1}{|Q_k|}\sum_{h_i \in Q_k} h_i\|^{\frac{1}{2}} \\ &= \left(\left(h_j - \frac{1}{|Q_k|}\sum_{h_i \in Q_k} h_i\right)^T \left(h_j - \frac{1}{|Q_k|}\sum_{h_i \in Q_k} h_i\right)\right)^{\frac{1}{2}} \\ &= \left(h_j^T h_j - 2\frac{h_j^T}{|Q_k|}\sum_{h_i \in Q_k} h_i + \frac{1}{|Q_k|^2}\sum_{h_{i_1}, h_{i_2} \in Q_k} h_{i_1}^T h_{i_2}\right)^{\frac{1}{2}} \end{aligned} \tag{3.43}$$

For any $i, j \in \{1, \cdots, N\}$,

$$\begin{aligned} h_i^T h_j &= \sum_{k=1}^{n_i} \alpha_k^i \Psi(x_k^i) \sum_{l=1}^{n_j} \alpha_l^j \Psi(x_l^j) \\ &= \sum_{k=1}^{n_i}\sum_{l=1}^{n_j} \alpha_k^i \alpha_l^j \Psi(x_k^i)\Psi(x_l^j) \\ &= \sum_{k=1}^{n_i}\sum_{l=1}^{n_j} \alpha_k^i \alpha_l^j K(x_k^i, x_l^j) \end{aligned} \tag{3.44}$$

where K is again the Gaussian RBF kernel.

3.8 Edit Distance on Real Sequence

As noted in [25, 26], motion recognition algorithms based on SVD decomposition of the motion matrix have a few caveats; they are insensitive to the direction of motion and more generally, to the order of frames inside the matrix. Indeed, given the fact that PCA extracts

the directions with the largest variance in the motion matrix, any permutation of the lines in the matrix will generate the same largest variance directions. Therefore, using eigenvector based classifiers to perform motion stream segmentation leads to poor results in that it soon desynchronizes with the ground-truth segments. An obvious example is the case where the stream contains the same motion performed in rapid succession; desynchronization will occur because segments starting from the middle of an animation and ending in the middle of the next (identical) animation will have similar scores to the ones aligned with the ground-truth.

To overcome this issue, this project applies the same disambiguation technique employed by Li et al. [25]; it summarizes a motion as a sequence of values representing the variation of the dominant variance in the data, (i.e. the projection of motion data on the first right eigenvector of the motion matrix) and computes a match between two such sequences using Edit Distance on Real Sequence (EDR) [53]. The idea is to measure how well does the order of frames in a motion sequence (summarized as a 1D motion vector) match the order of frames in the set of training samples, and thus improve the results of segmentation by only considering the motion segments that are closest to the training samples in the sense of EDR.

In the original formulation, given two real valued strings $r \in \mathbb{R}^m$, $s \in \mathbb{R}^n$, the EDR distance is computed as:

$$EDR(r,s) = \begin{cases} 0, s = null; \\ 0, r = null; \\ \min(& EDR(r_{2\cdots m}, s_{2\cdots n}) + \gamma, \\ & EDR(r_{2\cdots m}, s) + 1, \\ & EDR(r, s_{2\cdots n}) + 1), otherwise \end{cases} \tag{3.45}$$

where $\gamma = 1$ if r_1 matches s_1 and $\gamma = 0$ otherwise. A match is considered when $|r_1 - s_1| \leq 0.25\sigma$, where σ is the standard deviation of all the motion vector components involved in the training phase.

Chapter 4

Implementation

The goal of this project is twofold: on the one hand it aims to build a set of tools that allow to efficiently manage a motion database, while on the other hand it plans to use this database to perform real-time motion segmentation and recognition. The project is implemented in a series of steps summarized by Table 4.1; each step will be thoroughly discussed in the following sections. The result of the implementation consists in two software deliverables:

- A Windows application (motion database manager) that allows a user to group together a set of BVH files into a motion database, implements automatic and manual motion segmentation and clusterization techniques to facilitate the definition of motion classes, is responsible for training a multi-class hierarchical SVRDM classifier to recognize the defined motion classes, and also offers a testbed for evaluating the classifier performance before deployment on the Cell/B.E. A screenshot is available in Figure 4.1.
- A Cell/B.E. library (MoRec) that uses the classifier created by the motion database manager to perform real-time motion stream segmentation and recognition.

Task Name	Start Date	End Date	Status
Importing and filtering raw BVH files	11-May-09	15-May-09	Completed
Reducing raw data dimensionality	18-May-09	22-May-09	Completed
Splitting BVH clips into individual behaviors	25-May-09	29-May-09	Completed
Collecting similar behaviors into classes	01-Jun-09	05-Jun-09	Completed
Training SVRDMs for behavior classes	08-Jun-09	19-Jun-09	Completed
Implementing the motion recognition component	22-Jun-09	26-Jun-09	Completed
System and Integration tests (Windows)	06-Jul-09	10-Jul-09	Completed
Porting the motion recognition component to Cell	13-Jul-09	24-Jul-09	Completed
System tests (Cell)	27-Jul-09	31-Jul-09	Completed

Table 4.1: Project development milestones.

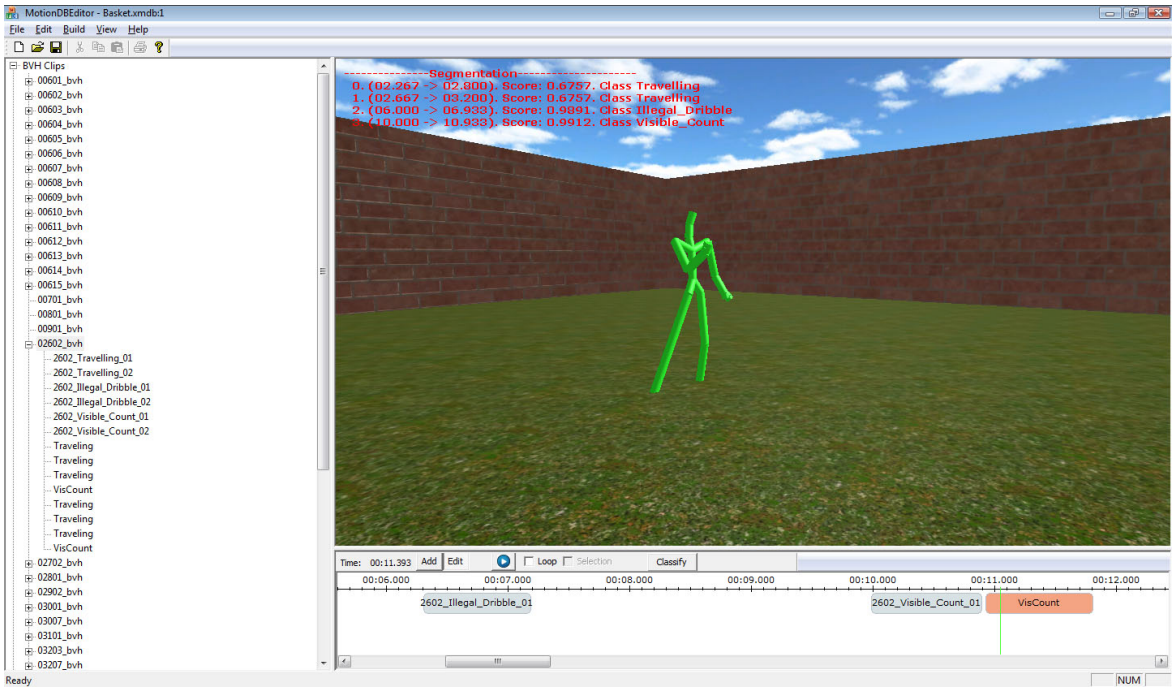

Figure 4.1: Screenshot from the motion database management application.

4.1 Importing and Filtering Raw BVH Files

The input data is represented by BVH files; a large number of motion capture BVH files are freely available from Carnegie Mellon University [4]. To test the proposed method of motion segmentation and recognition, a subset of 44 files containing approx. 7.5 minutes of various basketball and golf animations belonging to 13 human subjects was selected.

A simple tool was developed to convert the original BVH files to a proprietary format, where the joint angles are encoded using quaternions and each animated motion channel (i.e. root translation and joint orientation quaternion components) is individually compressed as a sequence of up to cubic Bézier curves (key-frames). The implementation uses Algorithm 1 and has the effect of smoothing the motion, reducing joint orientation noise / jitter, and also makes the animation sample rate independent. The motion database manager can only work with proprietary motion files but this is not regarded as an important issue because the conversion can be run as a batch process only once, with no need for manual intervention.

4.2 Reducing Raw Data Dimensionality

The skeleton in the BVH files has 31 pivots (bones) but the extremities (i.e. fingers, toes, etc.) can be safely discarded as irrelevant in the context of full body motion recognition; as a result, only 16 pivots are further considered (see Table 4.2). As the final goal of this project is to provide a motion recognition solution independent of the subject performing it and also of his or her relative orientation with respect to the motion capture device, the motion data further considered cannot include positional variables (i.e. relative joint position) because it will implicitly contain the bone lengths and will become dependent on a particular performer. Additionally, the root bone world orientation is also unappealing because the motion will be dependent on the particular orientation of the subject with respect to the MoCap device. Therefore, the project considers the following motion data parameters as relevant:

- the relative orientation quaternion of a joint with respect to its parent for each child joint;
- the relative orientation between the current and the next frame in case of the root joint.

Pivot Name	Pivot Parent
Hips	(none)
LHipJoint	Hips
LeftUpLeg	LHipJoint
LeftLeg	LeftUpLeg
RHipJoint	Hips
RightUpLeg	RHipJoint
RightLeg	RightUpLeg
LowerBack	Hips
Spine	LowerBack
Spine1	Spine
LeftShoulder	Spine1
LeftArm	LeftShoulder
LeftForeArm	LeftArm
RightShoulder	Spine1
RightArm	RightShoulder
RightForeArm	RightArm

Table 4.2: Representative pivots hierarchy.

The choice of parameters for the root bone is a substitute for angular velocity and requires further discussion. While the root angular velocity is not readily available, it can be estimated from the root pose. Assuming that Δ represents a fixed sampling time-step and that $q_W(t)$ stands for the world orientation quaternion of the root node at time t, the root angular velocity can be estimated as [54]:

$$\omega_W(t) = \frac{AxisAngle(q_W(t+\Delta)q_W^*(t))}{\Delta} \tag{4.1}$$

However, during the implementation on the Cell/B.E. it was discovered that the estimation of angular velocity was more prone to numerical imprecisions and produced different values for the same input data than the Windows version, with absolute errors up to 10^{-2} in some cases. To produce consistent numerical results on both platforms, it was decided to encode root motion as only the quaternion $q_{root} = q_W(t+\Delta)q_W^*(t)$; this choice also improves the numerical stability of further algorithms, because the motion data matrix is encoded using only quaternions and therefore any matrix element is confined inside $[-1, 1]$.

The 16 pivot quaternions encode a motion frame with 64 floating point values. However, these values are not independent, because human body joints have usually less than 3 DOF and rotations in child joints are also correlated with those of their parents. The number of necessary dimensions to represent the data can therefore be greatly reduced, with only minor deterioration in quality. In order to extract the most representative dimensions the data set is subjected to PCA. As mentioned in Section 3.4.1, PCA is based on SVD decomposition of the data matrix. To make SVD decomposition fast, this project uses the symmetric real matrix SVD code from CLAPACK [55] that employs the Multiple Relatively Robust Representations algorithm [56, 57].

Dimensionality reduction is the second stage of processing the motion data in this project and is applied on the processed (smoothed) animation files. First of all, each animation clip is sampled at a fixed rate of 120Hz, generating k raw data matrices $A_i, 1 \leq i \leq k$. The average motion vector is computed from all animation samples, and the lines of the matrices A_i are centered to have zero empirical mean. Matrices $M_i = A_i^T A_i$ are formed and the matrix

$$
\begin{aligned}
M &= A^T A \\
&= \begin{bmatrix} A_1^T & A_2^T & \cdots & A_k^T \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_k \end{bmatrix} \\
&= \sum_{i=1}^{k} A_i^T A_i \qquad (4.2) \\
&= \sum_{i=1}^{k} M_i \qquad (4.3) \\
& \qquad (4.4)
\end{aligned}
$$

is computed and analyzed with PCA, which determined that 16 values are sufficient to encode the motion with only 1% loss in quality. For a lower quality setting of 90%, the number of dimensions drops even more to just 8 floating point values. The User Interface (UI) of the motion database manager allows for easy selection of relevant pivots and re-computation

of reduced dimensionality motion matrices. The latter are cached into temporary files for speeding up further database operations, since it was considered that entire dataset dimensionality reduction will be seldom performed but its results will be often used in training classifiers.

4.3 Splitting BVH Clips into Individual Behaviors

Due to the fact that the BVH files usually contain more than one motion, the database manager allows for specifying portions of interest inside a clip. In order to tackle larger amounts of data, an implementation of Algorithm 2 offers the choice of automatic segmentation into logically distinct motion pieces. In this implementation, the values used for K, T and Δ are the ones proposed in the original paper, namely $K = T = 150$ and $\Delta = 10$ [12]. While the algorithm with this particular selection of parameters produces acceptable results in detecting logically distinct motion segments in an animation clip, it is worth mentioning that for cyclic / repeated motions it will not generate split points after each cycle. Therefore, user intervention is still required to select and edit the motion parts that are relevant for classifier training, but the database manager offers an easy-to-use point and drag UI to speed-up the process.

4.4 Collecting Similar Behaviors into Classes

A necessary step to be taken before classifier training is to specify motion classes as groups of logically similar motion segments. This process is implemented by an intuitive drag & drop UI inside the database manager. Also, in order to easily accomodate larger amounts of data, the application also contains an auto-clusterization feature that strives to determine the implicit motion classes and their associated samples by means of computing similarity between motion clips using the $kWAS$ metric discussed in Section 3.6.

In this implementation, the number k of relevant eigenvalues considered in the $kWAS$ measure is specified by the user (defaults to 12). Each pair of motion segments has its $kWAS$ measure computed and pairs with a similarity score larger than a user-defined threshold τ

(defaulting to 0.95) are grouped together in the same motion class. The quality of the clusterization using the $kWAS$ measure is given in Chapter 5.

4.5 Training SVRDMs for Behavior Classes

After the motion classes have been defined, a hierarchical classifier is generated by the motion database manager following Algorithm 7. The feature vectors for the motion samples in each class are computed by Equation 3.37; the number of eigenvectors in this implementation is user-defined, but the best classification accuracy is achieved for small values of k, i.e. $k \in \{2, 3, 4\}$. It is worth mentioning that before the feature vectors are determined, the signs of the eigenvectors entering their definition are processed to have consistent orientations using an implementation of Algorithm 5. The vectors used in ensuring proper signs of the eigenvectors are saved for further use in the real-time motion segmentation and recognition phase and will be referred to as sign vectors.

The theory involved in training a single classifier node is given in Section 3.7; a sketched implementation in pseudo-code is given in Algorithm 8.

Parameters involved in classifier training are generally customizable from the database manager UI; default values include the SVRM / SVRDM outlier penalty $C = 10$, SVRDM thresholds $t = 0.4$, $T = 1$, and number of relevant eigenvectors used to compute feature vectors $k = 2$. As mentioned before, training a SVRDM involves solving a QPP with box constraints; the code used by this project is derived from Numerical Recipes with the peculiarities set forth in Algorithm 3 [39, 45]. Problem 3.41 is also a QPP with box constraints and an additional equality constraint. Given the fact that the kernel matrix K_{ij} is symmetric positive definite in case of the Gaussian RBF kernel, the maximization problem is solved with a variant of the *QuadProg++* code, slightly modified to take advantage of the Cholesky factorization implemented in CLAPACK and optimized for box constraints [58].

Input: Set of motion classes $\{C_1, \cdots, C_m\}$
Output: Classifier tree

begin
  Partition classes $\{C_1, \cdots, C_m\}$ into macro-class sets Q_1, Q_2 using Algorithm 6
  $root.classifier \leftarrow$ `TrainSVRDM(`Q_1, Q_2`)`
  $root.Q_1 \leftarrow Q_1$
  $root.Q_2 \leftarrow Q_2$
  `Append(`*queue*, $root$`)`
  while *queue not empty* **do**
    $parent \leftarrow$ `RemoveHead(`*queue*`)`
    for $k \leftarrow 1$ **to** *2* **do**
      if $|parent.Q_k| > 1$ **then**
        Partition classes $parent.Q_k$ into macro-classes K_1, K_2 using Algorithm 6
        $child.classifier \leftarrow$ `TrainSVRDM(`K_1, K_2`)`
        $child.Q_1 \leftarrow K_1$
        $child.Q_2 \leftarrow K_2$
        `Append(`*queue*, *child*`)`
        $parent.children[k] \leftarrow child$
      end
      else $parent.children[k] \leftarrow NULL$
    end
  end
  return *root*
end

Algorithm 7: Generation of a classifier tree.

Input: Motion segment matrices $A_i^j, i \in \{1, \cdots, n_j\}$
belonging to macro-classes $Q_j, j \in \{1, 2\}$
Output: Classifier node

begin

Compute feature vectors x_i^j from A_i^j using Equation 3.37

Compute σ_{full} by averaging estimated SVRM σ values for each macro-class, using Algorithm 4 with x_i^j as training samples

Compute the support vectors a_i^j for the minimum enclosing spheres around samples x_i^j in feature space by solving Problem 3.41 for Gaussian RBF with σ_{full}

Compute matrix $C^T C$ using Equation 3.39 and a_i^j as class samples

Determine L from the Cholesky decomposition of matrix $C^T C$ and vectors $\hat{x_i}^j$ from Equation 3.40

Compute SVRDM σ_{red} by averaging estimated SVRM σ values for each macro-class, using Algorithm 4 with $\hat{x_i}^j$ as training samples

Solve Problems 3.27 and 3.28 with σ_{red} and training samples $\hat{x_i}^j$

return *SVRDM*

end

Algorithm 8: Training a classifier node.

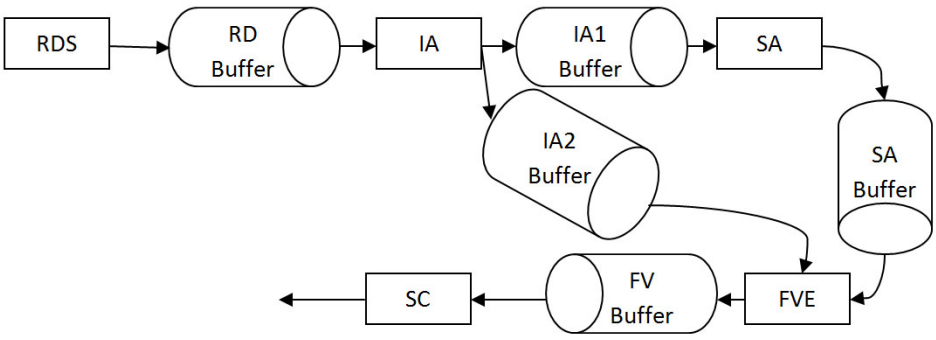

Figure 4.2: Motion recognition pipeline inside the database manager (Windows).

4.6 Implementing the Motion Recognition Component

The role of the motion recognition component is to analyze in real-time a raw motion stream, potentially coming from a MoCap device, and output motion segments corresponding to previously learned motion classes. In order to achieve real-time performance and simplify later deployment on the Cell/B.E., this component was developed using the pipeline paradigm, where each stage runs on a separate thread. The architecture is depicted in Figure 4.2. This section discusses the Windows platform implementation inside the motion database manager that is meant to act more as a visual testbed for different classifier settings. Once it has achieved the desired accuracy, the classifier can be exported as a binary file and used to perform real-time motion recognition on the Cell/B.E. The implementation on the latter platform is based on a slightly different design, mostly imposed by program and memory size limitations of the Synergistic Processor Element (SPE).

4.6.1 Raw Data Source (RDS)

This is the first stage of the pipeline that emulates a MoCap device data source. It is responsible for loading a processed BVH file (raw BVH file with cubic Bezier keyframe extraction, as mentioned before) and generating a raw data vector containing the current pose quaternions 120 times per second. The raw vector is then written to the RDS buffer in order to be processed by the next stage. It is worth noting that all pipeline stages are not directly linked and all communication between them is buffered, in order to allow maximum throughput.

Since there will only be a reader and a writer thread accessing the buffers at any given time, the multi-threaded access policy to the shared buffers was optimized to allow for simultaneous read / write access, as long as the operations do not involve overlapping locations. This has the effect of reducing pipeline stage stalling and improving throughput.

4.6.2 Input Assembly (IA)

As mentioned before, not all pivots are relevant in full body motion recognition, and this stage deals with selecting the proper pivot data and converting it to the format used in classification. Also, because of the fact that analyzing the motion stream at 120Hz in real time is computationally expensive, this stage gathers the incoming motion frames into chunks of $\Delta = 16$ frames each. Formally, every 16 frames this stage performs the following operations:

- Form the raw motion matrix A where each line represents a motion frame (pose);
- Replace the root quaternion with the relative orientation between the current and the next frame;
- Subtract the empirical mean determined in Section 4.2 from each line of matrix A;
- Apply the projection matrix discovered during PCA dimensionality reduction to keep only r most relevant dimensions and determine A_r;
- Compute $M = A_r^T A_r$ and send it to the IA Buffer, to be consumed by the SA stage.

It can be seen from the pipeline diagram that the IA stage generates output in two buffers; the second buffer keeps the A_r matrices computed for each motion chunk as they will be needed later, at the FVE stage.

4.6.3 Segment Assembly (SA)

The Segment Assembly stage generates various motion segments to be run through the classifier in the following stages. A motion segment is a pair (c_s, c_e), where c_s, c_e are the start and respectively the end index of motion chunks generated by the IA stage. When a new motion chunk c_e becomes available in the IA buffer, the segment assembly generates motion

matrices $M_k = \sum_{c_i=c_e-k}^{c_e} A_r^T(c_i)A_r(c_i)$ where the matrix products $A_r^T(c_i)A_r(c_i)$ are received from the IA stage and k is the number of chunks in a segment, $k \in \{4, ..., 20\}$ in the implementation. The insight is to generate increasingly longer sliding windows on the motion stream, analyze them against known motion classes and keep the motion segments with the best scores. As soon as the motion matrices are computed, they are written to the SA buffer, in order to be consumed by the FVE stage.

4.6.4 Feature Vector Extraction (FVE)

This is the most computationally intensive stage, as it performs SVD on the incoming motion matrices sent by the SA stage. The SVD returns the most relevant eigenvectors / eigenvalues; the eigenvectors are further processed to have consistent signs by using the sign vectors introduced in Section 4.5, and finally combined into feature vectors. The motion data matrices in IA2 buffer are also used at this stage to compute an associated motion vector for each motion segment. The motion vector is the projection of the motion data on the direction of maximum variance, or, more formally, the first right eigenvector in the SVD decomposition of motion matrix A_r. The (feature vector, motion vector) pairs associated with motion segments are finally written to the FV buffer, in order to be run through the classifier.

4.6.5 Segment Classification (SC)

This is the final stage of the pipeline and deals with motion recognition and segmentation. As mentioned before, the FVE stage feeds the SC stage with (feature vector, motion vector) pairs for various combinations of motion segments, as generated by the SA stage. For each feature vector, the hierarchical classifier developed in the training stage is run; the feature vector enters the root classification node, the child with maximum SVRDM output is chosen if above class threshold, otherwise it is rejected as a non-class. The classification runs down the tree until a leaf node is reached; is is worth noting that the class threshold T used in classification is different from the one used in training, to allow for a better generalization (i.e. $T = 0.92$ in classification, vs. $T = 1$ in training).

As soon as a potential motion class is detected by the classifier, a confidence level is estimated by computing the EDR distance between the current segment motion vector and the

candidate class motion vectors (see Section 3.8). A problem with this distance measure is that it produces an unnormalized result, that cannot be easily compared with similar distances obtained for different motion classes. To overcome this limitation, the EDR used in this project was normalized by computing the minimum cost path of transforming r to s and dividing this cost to the transformation path length [59].

Given a motion sample, its normalized EDR distances to all motion vectors in the suspected class are computed, and the distance from the motion sample to the class is defined as the minimum of the above computed distances. To further filter the segmentation output, the normalized EDR cost between a motion sample and a class is multiplied by a Gaussian that favors the motion segments with lengths close to the mean length of the samples in the motion class. This has the effect of reducing the estimated confidence of very short motion sequences. Formally, given a motion sample x starting at chunk c_i and ending at chunk c_e, its confidence value is defined as:

$$Conf(x_{c_i,c_e}) = \left(1 - \frac{0.9}{1 + e^{-30*EDR+15}}\right) \cdot e^{-\frac{(len_x - len_{avg})^2}{2(0.25len_{avg})^2}} \tag{4.5}$$

The EDR distance is passed through a manually-tuned sigmoid, that was preferred over a linear dependence due to its better rejection ability for incomplete motion segments. The length in seconds of the sample x is denoted by len_x and the average length of the motion class in which x is assumed to be is len_{avg}. The sigma value for the Gaussian was again empirically chosen to be a quarter of the average motion class length, as it produced the best results.

The confidence values are considered samples of a function defined on $\mathbb{R}^2$ as shown in Figure 4.3. For each motion chunk produced by the Input Assembly, a new line of confidence values is produced (depicted by a red rectangle in the figure); these confidences are given by Equation 4.5. Of course, not all recognized segments are meaningful and segmentation candidates are determined as local maxima points in the confidence graph. A point is considered to be a local maxima if its confidence value is higher than the confidence of its 8 neighbors (depicted by green rectangles in Figure 4.3).

The resulting motion segments can still exhibit a large amount of overlap; for instance, when a point of local maxima is in fact caused by noise / numerical imprecisions (false positive)

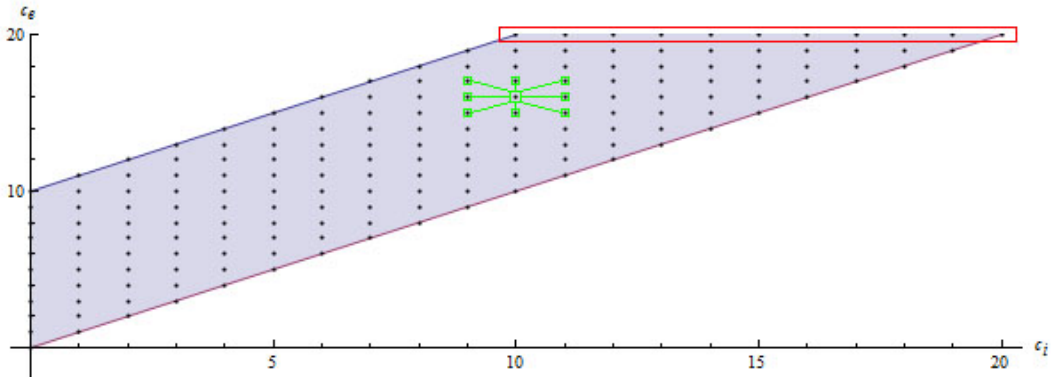

Figure 4.3: Graph of confidence values entering the Segment Classification stage. A motion segment starts at motion chunk c_i (x axis), ends at motion chunk c_e (y axis), and is represented as a single dot in the graph.

and a true local maxima is found nearby. To further clean-up the resulting motion segments, a post-processing stage tracks the motion candidates, merging together any pair overlapping more than 30% in time. The surviving motion segments and their associated classes are reported as the solution of motion segmentation and recognition. A final remark must be made about the tracked segments; as soon as the end time of a tracked segment precedes any possible start time of a newly generated motion candidate, the tracked segment can be removed from memory thus ensuring the scalability of the proposed solution to handle infinitely long input streams.

4.7 Porting the Motion Recognition Component to Cell/B.E.

The final goal of this project is to provide a highly optimized library (further referred to as MoRec) that would enable a Cell/B.E. application to perform real-time motion segmentation and recognition. Cell/B.E. is a heterogeneous multi-core processor developed by the STI alliance (Sony Computer Entertainment, Toshiba and IBM) primarily for the PlayStation®3 gaming console. However, its impressive processing power, high throughput memory access scheme and also its power efficiency make it a strong candidate for scientific applications, as shown in a study by Lawrence Berkeley National Laboratory [60]. In what follows, a brief description of the Cell/B.E. architecture is given in order to provide a clear context for changes in the design of the motion segmentation and recognition component.

4.7.1 Cell/B.E. Architecture Overview

The main building block on a Cell chip is represented by the PowerPC Processor Element (PPE): a 64-bit two-way multithreaded core based on the PowerPC 970, the PPE is generally responsible for coordination tasks. Its instruction set allows direct access to the eXtreme Data Rate (XDR) main memory via a 512KB L2 and respectively 32KB instruction / 32KB data L1 cache. While being geared towards general-purpose computing such as running an operating system, the PPE also includes IBM's VMX engine for SIMD processing, allowing it to perform the same operation on vectors of data simultaneously (i.e. 4 single precision floating point values) and therefore offer increased execution speed for various categories of scientific applications.

The thing that distinguishes the Cell from regular desktop-oriented processors and positions it closer to the chips found in graphics cards is the set of 8 interconnected SPEs. These are highly specialized SIMD processors working only on 128bit data vectors, with a huge number (128) of registers allowing the compiler to perform more aggressive code optimizations such as resolving more local function variables and function call parameters to register locations rather than memory. A SPE cannot access main memory directly; it can only use 256KB of local on-chip memory called the Local Store (LS). The LS is shared by the program instructions, program stack and also all allocated memory during program execution; in other words the maximum memory requirements of any program running on the SPE must be below 256KB and the burden of ensuring this is generally left to the programmer as there are no triggered exceptions when for instance the stack pointer overwrites user-allocated memory locations.

Each SPE contains a memory flow controller (MFC) that acts as a hardware thread responsible for managing communications in and out of the SPEs. The MFC implements primitives that initiate, manage and track the completion status of Direct Memory Access (DMA) transfers from / to LS to / from main memory / other LS. The transfers take place in parallel with the execution of the SPE core, allowing the latter to interleave data processing with time-consuming memory operations and thus hide memory access latency.

MFCs rely upon the Element Interconnect Bus (EIB) to provide the communication infrastructure. The EIB consists in four 16 byte-wide rings connecting the PPE with the 8 SPEs, main memory interface controller (MIC), and I/O interfaces (IOIF), where two rings carry

bool	*mrInit*(**char** * szClassifierFile); Initializes the motion recognition library with a classifier.
void	*mrTerminate*(); Deallocates all resources held by the library.
bool	*mrSetDataSource*(**char** * szSourceFile); Starts generating raw motion data using as source the animation stored in the given file.
bool	*mrStopDataSource*(); Stops generating raw motion data.
bool	*mrIsDataSourceDepleted*(); Tests if the data source can still produce data.
bool	*mrStart*(); Starts the real-time motion segmentation & recognition process.
bool	*mrStop*(); Stops the real-time motion segmentation & recognition process.
bool	*mrQueryMotionSegments*(**MotionSeg** * pSegments, **int** & nSegments); Queries the solution of the motion segmentation & recognition.

Table 4.3: MoRec API specification.

data in clockwise direction and the other two perform data transfers in the opposite, counter-clockwise direction. According to Kistler et al. the EIB has a theoretical peak data bandwidth of 204.8 Gbytes/s in a 3.2 GHz Cell processor [61]; however, special care must be taken by an application developer in order to achieve high communication bandwidth. If for instance the communication among SPEs is not carefully planned, a lot of EIB bandwidth can be lost due to collisions; also, if the majority of communication accesses the main memory, the EIB will be bottlenecked by the main memory transfer bandwidth, an order of magnitude lower.

4.7.2 MoRec Design

As it has been previously emphasized, MoRec is a port of the motion segmentation and classification component introduced in Section 4.6 in the form of a static library compiled for the Cell/B.E. The API made available to a calling application is listed in Table 4.3, while a functional diagram is given in Figure 4.4. An example utilization of the MoRec library to segment and classify motions in a stream is provided in Listing 4.1.

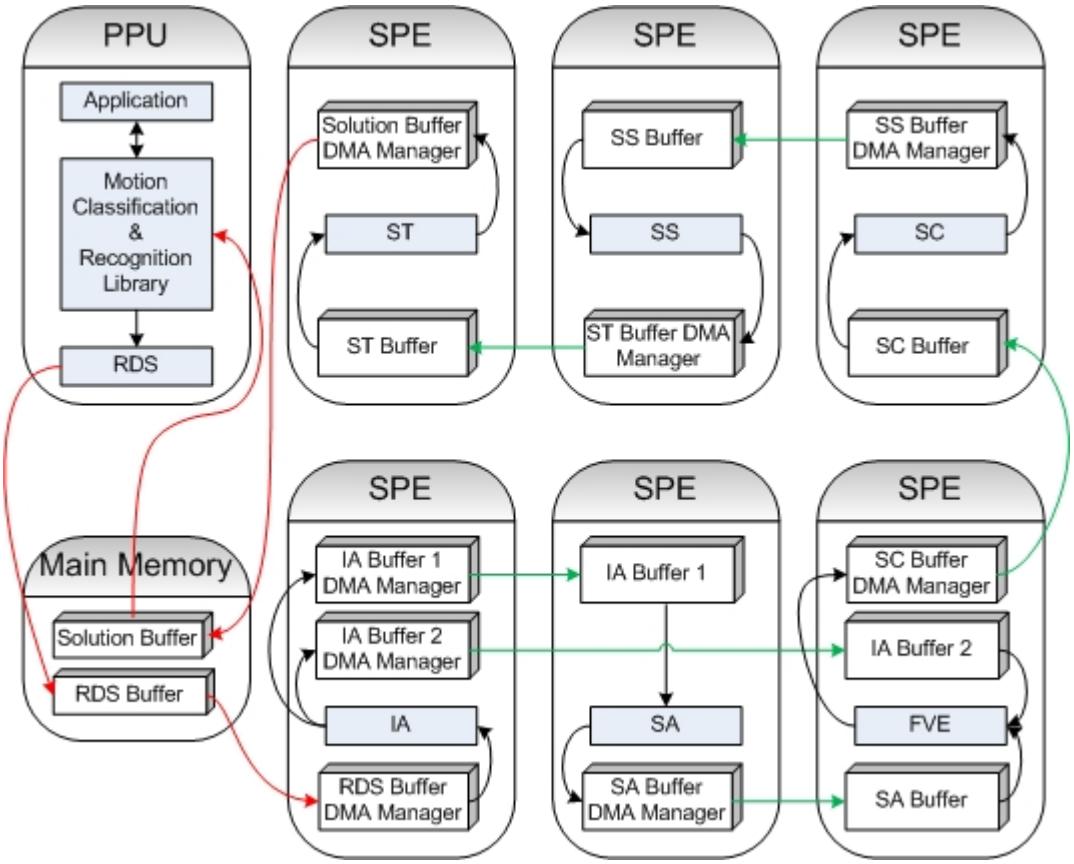

Figure 4.4: Motion segmentation and recognition library architecture (Cell/B.E.).

```
/*  ... Other initialization code ... */

if( !mrInit(szClassifier)       ||  //Init MoRec library
    !mrSetDataSource(szSource)  ||  //Init data source
    !mrStart())                     //Start recognition
  return; //Error!

//Game loop
while(!mrIsDataSourceDepleted())
{
  //Periodically query for segmentation results
  nTestSegs = MAX_TEST_SEGS;
  mrQueryMotionSegments(pTestSegs, nTestSegs);

  /*  ... Do other game specific updates  ... */
}

if( !mrStopDataSource() ||  //Stop the data source
    !mrStop())              //Stop recognition
  return; //Error!

//Free memory
mrTerminate();

/*  ... Other termination code  ... */
```

Listing 4.1: Typical application using MoRec.

At first, an application using MoRec must call *mrInit(...)*; it opens the binary classifier file exported by the database manager running on Windows and allocates all data buffers needed by the classification process in main memory. These include empirical motion data mean and dimensionality reduction projection matrix, sign vectors needed in feature vector computation, motion classes together with non-linear dimensionality reduction parameters, classifier support vectors and motion vectors, and of course, the classifier hierarchy.

A data source is started by *mrSetDataSource(...)*; this currently creates a PPE thread implementing the RDS pipeline component as already presented in Section 4.6.1. It abstracts a raw motion data source for the rest of the MoRec library and can be extended in future versions to include support for real MoCap devices; all that is needed from such a source is the ability to query its skeleton topology, data acquisition rate, and the frame data. The motivation for implementing the RDS component on the PPE was to ensure easier development of future extensions accommodating different data sources, without the need to modify the rest of the system. The output of the RDS stage (i.e. a sequence of motion frames) is written in a circular buffer (RDS Buffer) allocated on main memory, again, in the spirit of minimizing the impact on the rest of the system in case of future extensions.

Once a data source is live, the rest of the motion segmentation and recognition pipeline can be started. This operation is performed by *mrStart()*; it uploads and starts the appropriate pipeline stage programs into each of the 6 employed SPEs, waits for them to allocate working buffers in LS, and configures inter-SPE communication by exchanging memory addresses via mailboxes. At this point, the motion recognition pipeline is fully running and can be queried for a segmentation solution with *mrQueryMotionSegments(...)*. When the motion data source is depleted or the application otherwise decides to end its execution, it must first stop filling the RDS Buffer (*mrStopDataSource()*), signal and wait for the SPEs to stop their execution (*mrStop()*), and finally release the memory resources held by the classifier (*mrTerminate()*).

Figure 4.4 contains a detailed description of the motion segmentation and recognition pipeline. The IA, SA and FVE stages have the same functionality as their correspondents, already covered by Sections 4.6.2, 4.6.3 and respectively 4.6.4. However, the FVE stage on the Cell uses a different implementation to compute the SVD of a motion matrix because the CLAPACK code proved to be too long to fit the LS of a SPE. Given the fact that SVD is applied only to symmetric real matrices, the SPE implementation uses the *tred2* and *tqli*

algorithms in Numerical Recipes to first reduce the symmetric matrix to tridiagonal form using the Householder method and then compute its eigenvalues / eigenvectors with the QL algorithm with implicit shifts [39].

The SC stage described in Section 4.6.5 had to be divided in three in order to reduce memory requirements and fit the constrained space of the LS. The SC stage on the Cell/B.E. only runs the hierarchical classification (the SVRDM tree); it provides to the next stage a set of at most four probable motion classes the segment belongs to. The next stage, Segment Scoring (SS) computes the normalized EDR distance between the incoming motion segment and the classes identified by the SC stage as most probable. The output of this stage is the motion class with the highest score, as computed by Equation 4.5. Finally, the last stage in the pipeline is the Segment Tracking (ST); its responsibility is to analyze input from the SS stage, extract confidence local maxima points as potential segmentation candidates, filter them to eliminate overlaps and update the solution buffer in main memory whenever the former changes (see Section 4.6.5 for a more thorough description). Function *mrQueryMotionSegments(...)* simply copies the solution buffer from main memory to a user specified location using a mutex to ensure data consistency, it does not explicitly query the ST stage for an update.

4.7.3 Communication Primitives

As mentioned in Section 4.7.1, a carefully chosen communication mechanism between the PPE and SPEs can greatly increase the performance of an application. It is easy to notice from the functional diagram in Figure 4.4 that the data flow in MoRec naturally maps to the EIB ring topology. Main memory to SPE LS transfers happen mainly inside the IA stage; the latter should be able to run faster if its SPE is closest to the memory controller w.r.t. the EIB, because its communication with the main memory can take place virtually unaffected by the other SPEs, at a bandwidth limited by the IA processing throughput and the main memory transfer rate. In the same spirit, the bulk of communication takes place using SPE to SPE LS DMA transfers, generally between adjacent SPEs so that the number of simultaneous transfers in flight on the EIB is maximized. An exception is made in the case of IA that outputs data in two distinct SPEs (SA and FVE) of which only one is adjacent (SA); the situation is unavoidable but however not critical, due to the fact that only a SPE

is in between the IA and FVE stages. Given the fact that the EIB has two rings for each direction of transfer, it can be assumed that both the IA and SA stages can send data to the FVE in parallel, using both rings on the EIB.

To ensure the level of interference between the SPEs is kept to a minimum, the best SPE associated with each stage is chosen using the *spe_context_create_affinity* function in the Cell Software Development Kit (SDK) [62]. The IA stage is configured to have an affinity for main memory (flag *SPE_AFFINITY_MEMORY*) and therefore be run on the SPE closest to the MIC. Further on, the SA stage has an affinity for the IA stage and if possible will be placed next to it, the FVE stage has an affinity for the SA stage, and so on, following the natural order: Main Memory (RDS Buffer) → IA → SA → FVE → SC → SS → ST.

Referring again to Figure 4.4, the communication between SPEs takes place via circular buffers protected against simultaneous access to identical addresses. In other words, SPEs can simultaneously read and write from / to the same buffer object, while not at the same location. While the buffers (with the exception of the RDS Buffer) are allocated in the local storage of the SPEs to minimize main memory bandwidth, the concurrent access to their data is arbitrated by mutexes. Unfortunately, a mutex cannot be allocated on the LS of a SPE, so all communication buffers between the SPEs have their mutexes allocated on main memory, and therefore still need to access it every time they require atomic access.

As a general rule, each circular buffer object that is not locally allocated is accessed by the SPE via a DMA manager. This acts as a local copy of the buffer in the LS of the accessing SPE that is responsible for permanently synchronizing its contents with the original, remote buffer. The synchronization takes place in only one direction: the DMA manager either reads from the remote buffer, or exclusively writes to the remote buffer. Naturally, the DMA manager takes advantage of the MFC's ability to manage up to 32 in-flight transfers and uses a multi-buffered approach when synchronizing with the remote buffer object [62]. This allows several completed buffer entries to be sent to the next stage via DMA while the SPU continues to process data in parallel to fill other empty buffer entries.

4.7.4 Caching

As mentioned before, the majority of communication in MoRec is done using the circular buffer / DMA manager paradigm. However, the SC and SS stages deserve special treatment due to the potentially large amount of main memory traffic they might require. First of all, the Segment Classification stage needs both the non-linear dimensionality reduction support vectors and the classifier support vectors. To ensure the design is scalable, this data is constantly requested via double-buffered DMA from the main memory; in this way the SPE can compute the classification independent of the number of classes or support vectors. However, due to the fact that consecutive motion segments arriving at the SC stage for classification only differ by a fraction of a second (in this case 0.16s), the path through the classifier tree is mostly the same, so the number of memory accesses needed to classify the consecutive segments can be reduced by caching the recently used classifier data in the LS.

The same insight is applied in the case of Segment Scoring that needs to transfer motion vectors for a given motion class in order to compute the confidence score of classification: consecutive segments are generally classified as belonging to the same motion class and therefore the same set of motion vectors is needed to evaluate their confidence values. By caching the most recently used motion vectors, the SS can decrease its main memory traffic and improve its throughput.

Caching is implemented in a manner that takes advantage of the SIMD nature of the SPE and is also optimized for speed. A cache is an object that maps a fixed-length chunk in main memory to a pre-allocated, same-length chunk in the LS of a SPE. To map a main memory address to LS, the following operations are done:

- A hash key is determined by shifting the main memory address right by 7 bits. This exploits the fact that classifier data is allocated at consecutive memory addresses and aligned to 128 byte boundaries in order to accept DMA transfers.
- A `bucket` is computed as the remainder of dividing the hash key to the bucket capacity of the cache (fixed upon creation).
- The `index` of the minimum time-stamp is determined across the bucket entries. A bucket entry contains only four (value, time-stamp) unsigned int pairs, encoded as two 128 bit vector values.

- The main memory address is mapped at the LS address `&ls_buffer[chunk_size * ((bucket << 2) + index)]`. The bucket entry is also updated to reflect the mapping: its value is set to the main memory address and the time-stamp updated to hold the most recent time.

To determine if a main memory location is already in the cache, the following steps must be taken:

- Compute the hash key and `bucket` as above.
- Search if the main memory address corresponds to any of the values in the bucket entry, if so, determine its `index`.
- If a valid index was found, the main memory address is already mapped at the LS address `&ls_buffer[chunk_size * ((bucket << 2) + index)]`. The time-stamp of the bucket entry is also updated to the hold the most recent time.

It can be seen that the cache is 4-way associative: this was preferred due to its natural mapping onto SIMD data types. Because the data to be cached is usually allocated at increasingly larger consecutive addresses and is accessed by the computation on the SPE in the same manner, it will be typically cached in different (consecutive) buckets and therefore at first glance associativity will not be necessary. However, the SPE only performs SIMD operations natively, so writing scalar code for a direct mapped cache will produce a sub-optimal solution due to the implicit scalar to SIMD conversion overhead. Therefore, the cache object in this project is implemented as 4-way associative and optimized with SIMD intrinsics.

A final remark needs to be made about the support and motion vectors needed by the SC and SS stages. They are stored in main memory as concatenated into fixed size chunks, and a SPE will actually DMA-transfer several vectors at a time into the LS. This is done in part to simplify the implementation of fixed-size cache presented above, but also to reduce the number of DMA requests and minimize transfer overhead.

4.7.5 SIMD Optimizations

Almost all motion recognition stages were optimized to use SIMD. The data provided by the RDS Buffer is naturally aligned to 4-valued single precision floating point vectors since

it is formed by bone quaternions. Further on, in the dimensionality reduction stage (see Section 4.2), the number of relevant dimensions is automatically rounded up to the largest multiple of 4 in order to be able to continue using SIMD operations. Therefore, in the IA and SA stages almost all operations involving the manipulation of motion matrices are implicitly assuming multiple-of-4 sized data and process it using SIMD optimized code. It is easy to notice that because the number of reduced dimensions is a multiple of 4, the feature vectors also have a multiple of 4 length. Therefore, the Gaussian RBF kernels involved in classifier non-linear dimensionality reduction are also computed with SIMD code. Even if the support vectors after non-linear dimensionality reduction are only 2-dimensional, they are padded to 4 dimensions and the SIMD Gaussian RBF kernel function simply discards the two uppermost values from the final result. Unfortunately, the majority of code run by the SS and ST stages is scalar; nevertheless, a large portion of the normalized EDR algorithm was successfully SIMD-ised.

Chapter 5

Evaluation & Discussion

In order to evaluate the quality and performance of the proposed framework, a set of 11 motion classes were manually defined as the ground truth. These are briefly summarized in Table 5.1: *Walk* corresponds to a walking cycle starting with the right foot, *VisibleCount*, *Traveling*, *IllegalDribble*, *TechnicalFoul*, *StopClock*, *Pushing* and *NoScore* are well known basketball referee signals, *DribbleLeft*, *DribbleRight* are basketball dribble moves performed with the left and respectively the right hand, and finally *Swing* contains samples of golf swings. The table also contains the minimum, maximum and average length of each motion class in seconds as estimated from its training samples, the number of available motion samples (although only a subset is used in training), and also the number of distinct subjects the samples belong to.

The following aspects of the proposed framework were evaluated:

- the quality of the automatic motion segmentation (Section 4.3);
- the accuracy of automatic segment clusterization into motion classes (Section 4.4);
- the quality of the hierarchical motion classifier (Section 4.5);
- the real-time motion segmentation and recognition accuracy (Section 4.6);
- the performance of the Cell implementation (Section 4.7).

Code	Name	Length (sec)			No. Samples	No. Subjects
		Min	Med	Max		
C_1	*Walk*	0.96	1.078	1.196	12	4
C_2	*VisibleCount*	0.9	0.903	0.906	12	7
C_3	*Traveling*	0.353	0.4365	0.52	20	7
C_4	*Swing*	1.739	1.7995	1.86	10	1
C_5	*IllegalDribble*	0.886	0.8995	0.913	11	7
C_6	*DribbleRight*	0.74	0.8365	0.933	16	3
C_7	*DribbleLeft*	0.866	0.8795	0.893	3	1
C_8	*TechnicalFoul*	0.595	0.598	0.601	10	7
C_9	*StopClock*	0.595	0.601	0.607	8	7
C_{10}	*Pushing*	0.69	0.696	0.702	10	7
C_{11}	*NoScore*	0.696	0.702	0.708	10	7

Table 5.1: Manually defined motion classes.

5.1 Automatic Motion Segmentation

The automatic motion segmentation algorithm in Section 4.3 was applied to several motion clips in order to evaluate its capacity of detecting logically distinct motion segments. A sample run is given in Figure 5.1; the blue segments represent the desired result (the ground truth) and alternating red and black segments are the result of the automatic motion segmentation. It can be seen that in general, the resulting segments include the ground truth and in some cases, more than one ground truth segment. The latter is a characteristic of the employed algorithm, that is unable to detect cyclic / repeated motions as separate segments and simply generates a single motion segment containing all repeated instances. Another problem with the algorithm is the incorporation of potentially long transition / idle animations into the resulting motion segments. This phenomenon can also be witnessed in Figure 5.1; this is the reason why the automatically generated motion segments that include ground truth segments are much longer than the latter.

The issue of not detecting cyclic / repeated animations and the incorporation of transitions / idle frames into the resulting segments turn into major disadvantages in the context of automatically detecting distinct motion segments appropriate for classifier training with minimal user intervention. The resulting motion segments still need to be further tweaked by a human user before they can be used in training, which involves a fair amount of time and can

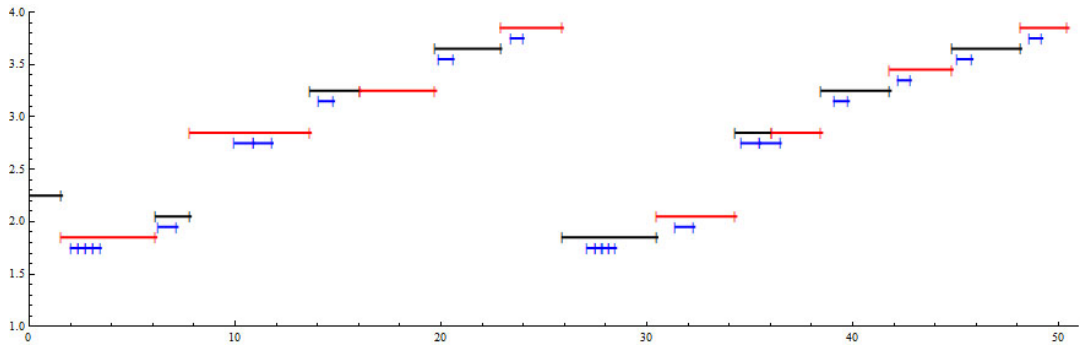

Figure 5.1: Automatic motion segmentation result for clip 2 of subject 26. Blue segments represent the ground truth, alternating red and black segments are the result of the automatic motion segmentation. Class ID is encoded as y height.

		Ground Truth	
		Positive	**Negative**
Observed Outcome	**Positive**	708	3852
	Negative	0	2821

Table 5.2: Auto-clusterization confusion matrix for $\tau = 0.95$. The number of discovered classes is 2, with $RI = 0.47$.

become prohibitive for large motion databases.

5.2 Automatic Segment Clusterization

Given the set of manually defined motion classes, the quality of the automatic clusterization algorithm is evaluated with the *Rand Index* (RI) introduced in Section 3.6.1. The number k of eigenvalues used in the experiment is fixed at 12 but does not greatly affect the final results; for instance, an identical solution was obtained for a much lower setting of just 4. Confusion matrices for several values of τ along with the RI and the number of discovered classes are given in Tables 5.2 to 5.4.

Unfortunately, the auto-clusterizaton technique based on the $kWAS$ similarity measure does not produce satisfactory results. For instance, in the case of $\tau = 0.99$ the *Swing* motion class is perfectly separated in a single cluster while *Traveling* and *TechnicalFoul* are clustered together. Further increasing $\tau \rightarrow 1$ does not help because the number of discovered classes

		Ground Truth	
		Positive	**Negative**
Observed Outcome	**Positive**	708	3042
	Negative	0	3631

Table 5.3: Auto-clusterization confusion matrix for $\tau = 0.98$. The number of discovered classes is 3, with $RI = 0.58$.

		Ground Truth	
		Positive	**Negative**
Observed Outcome	**Positive**	588	438
	Negative	120	6235

Table 5.4: Auto-clusterization confusion matrix for $\tau = 0.99$. The number of discovered classes is 14, with $RI = 0.92$.

is already above 11 (the ground truth), which means that there are already classes spanning multiple clusters. Indeed, *Pushing* spans 3 clusters and both *NoScore* and *VisibleCount* span 2 clusters. Decreasing τ causes nearby clusters to merge but again is ill-advised, since there are already clusters not fully separated. Therefore, the auto-clusterization algorithm based on the $kWAS$ similarity measure cannot be used in an unsupervised manner to discover implicit classes. The problem might be remedied if the similarity criterion is replaced by a more advanced method; research is still needed to provide further clarifications regarding this subject.

5.3 Quality of Classification

A total of 61 motion samples was randomly selected from the set in Table 5.1 such that each class is represented by roughly 50% of its available motion segments. An exception was made for *DribbleLeft* due to the lack of available data and the respective class is represented by all (3) of its motion samples. The 61 selected samples were used to train a hierarchical SVRDM classifier tree (see Section 4.5 for details regarding the implementation) and the result is given in Figure 5.2.

Due to the fact that while training each classifier tree node the dimensionality of the feature vectors is non-linearly reduced to only 2, it is possible to visualize the result of the SVRDM

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}
C_1	10	0	0	0	0	0	0	0	0	0	0
C_2	0	10	0	0	0	0	0	0	0	0	0
C_3	0	0	18	0	0	0	0	0	0	0	0
C_4	0	0	0	10	0	0	0	0	0	0	0
C_5	0	0	0	0	7	0	0	0	0	0	0
C_6	0	0	0	0	0	15	0	0	0	0	0
C_7	0	0	0	0	0	0	2	0	0	0	0
C_8	0	0	0	0	0	0	0	5	0	0	0
C_9	0	0	0	0	0	0	0	0	3	0	0
C_{10}	0	0	0	0	0	0	0	0	0	6	0
C_{11}	0	0	1	0	0	0	0	0	0	0	8
NC	2	2	1	0	4	1	1	5	5	4	2

Table 5.5: Confusion matrix for the hierarchical classifier in Figure 5.2.

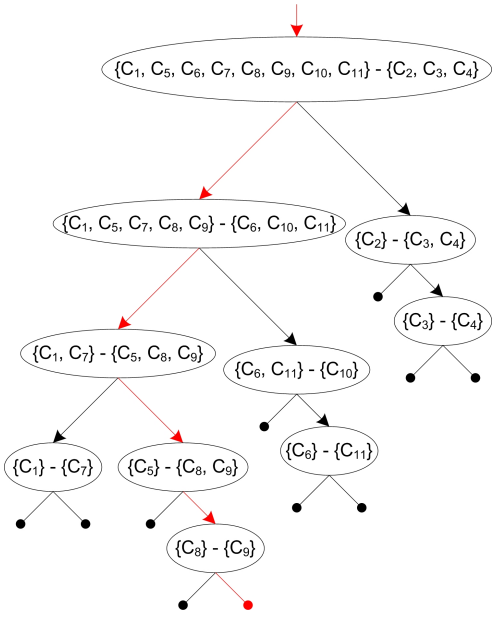

Figure 5.2: Hierarchical SVRDM classifier tree. Red arrows indicate the progress of classification in case of a *StopClock* motion sample.

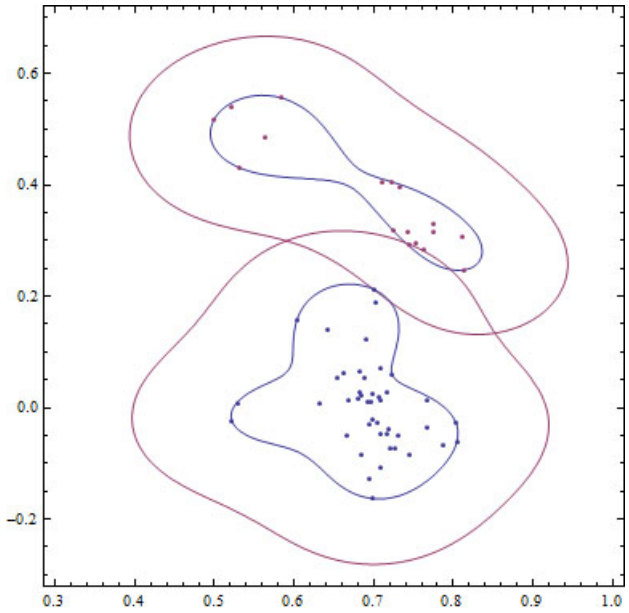

Figure 5.3: SVRDM decision boundaries for macro-classes $\{C_1, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}\}$ - $\{C_2, C_3, C_4\}$. Decision function values are greater than $T = 1$ inside the blue boundary and greater than $t = 0.4$ inside the magenta boundary. $\sigma = 0.1$.

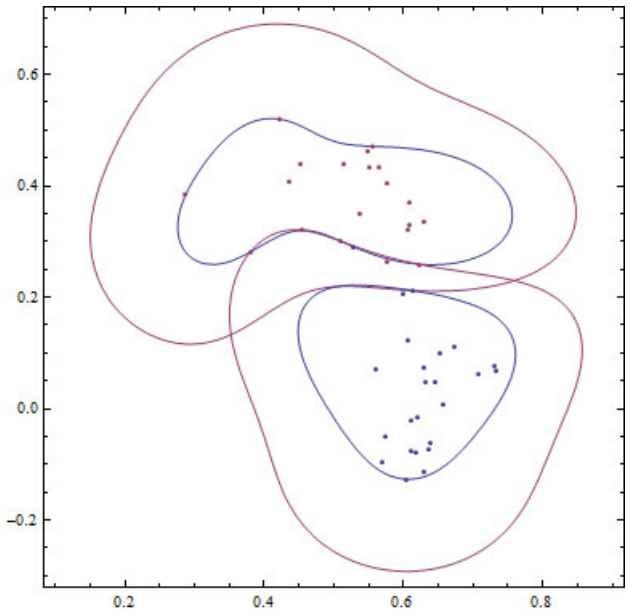

Figure 5.4: SVRDM decision boundaries for macro-classes $\{C_1, C_5, C_7, C_8, C_9\}$ - $\{C_6, C_{10}, C_{11}\}$. Decision function values are greater than $T = 1$ inside the blue boundary and greater than $t = 0.4$ inside the magenta boundary. $\sigma = 0.1375$.

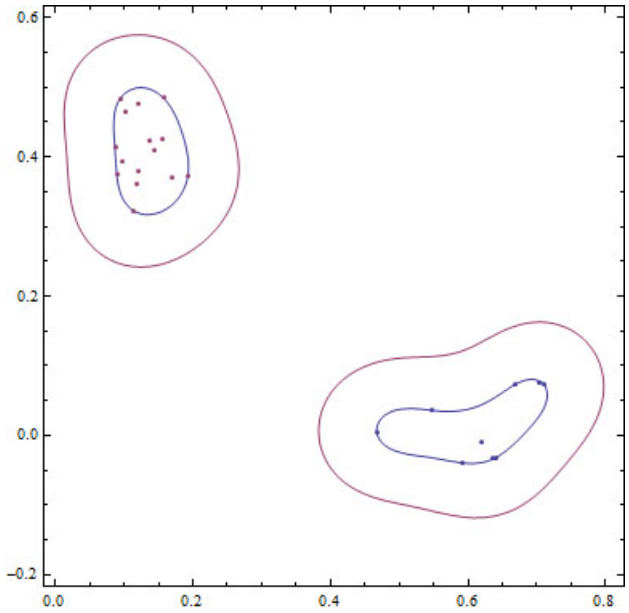

Figure 5.5: SVRDM decision boundaries for macro-classes $\{C_1, C_7\}$ - $\{C_5, C_8, C_9\}$. Decision function values are greater than $T = 1$ inside the blue boundary and greater than $t = 0.4$ inside the magenta boundary. $\sigma = 0.07$.

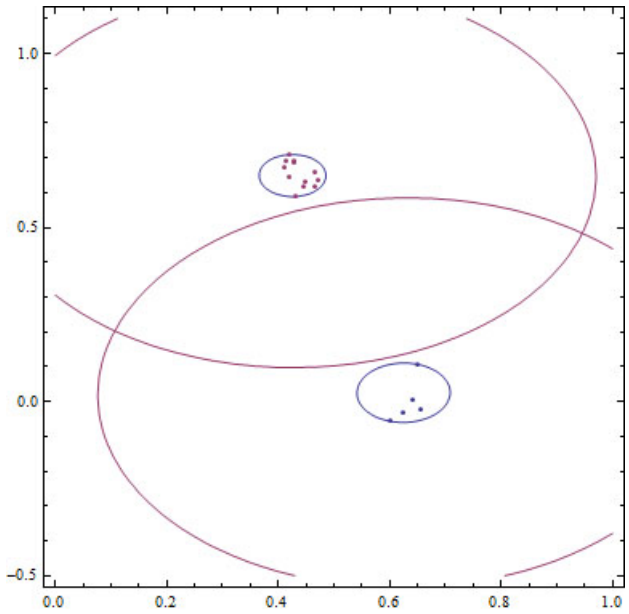

Figure 5.6: SVRDM decision boundaries for macro-classes $\{C_2\}$ - $\{C_3, C_4\}$. Decision function values are greater than $T = 1$ inside the blue boundary and greater than $t = 0.4$ inside the magenta boundary. $\sigma = 0.4$.

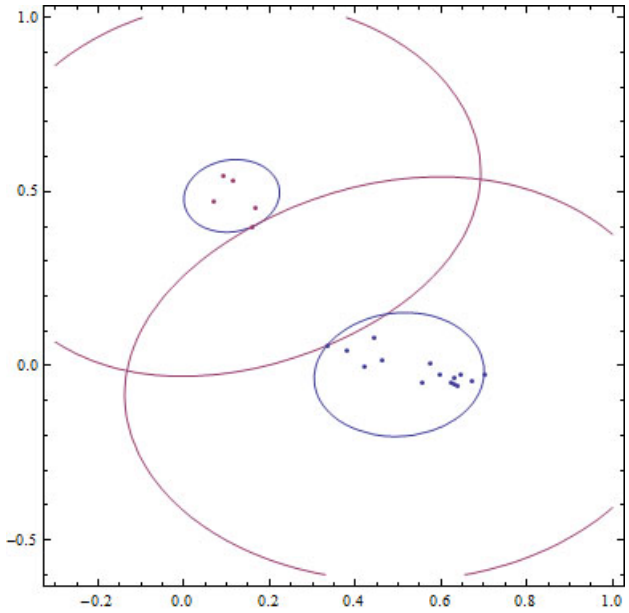

Figure 5.7: SVRDM decision boundaries for macro-classes $\{C_6, C_{11}\}$ - $\{C_{10}\}$. Decision function values are greater than $T = 1$ inside the blue boundary and greater than $t = 0.4$ inside the magenta boundary. $\sigma = 0.4325$.

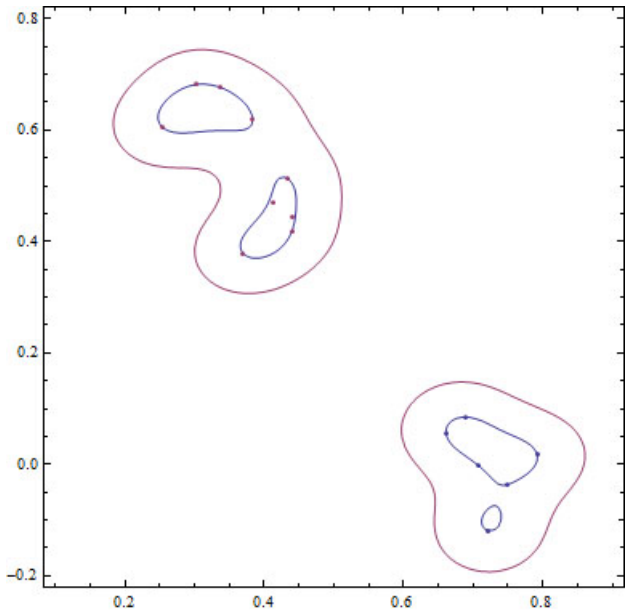

Figure 5.8: SVRDM decision boundaries for macro-classes $\{C_5\}$ - $\{C_8, C_9\}$. Decision function values are greater than $T = 1$ inside the blue boundary and greater than $t = 0.4$ inside the magenta boundary. $\sigma = 0.0575$.

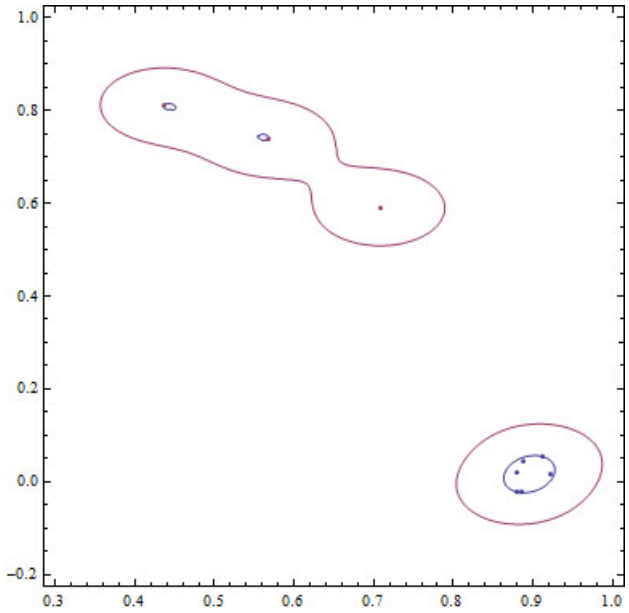

Figure 5.9: SVRDM decision boundaries for classes $\{C_1\}$ - $\{C_7\}$. Decision function values are greater than $T = 1$ inside the blue boundary and greater than $t = 0.4$ inside the magenta boundary. $\sigma = 0.06$.

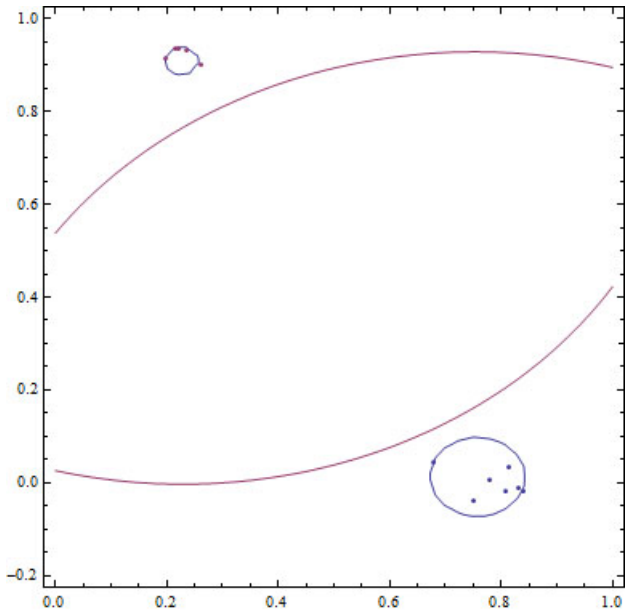

Figure 5.10: SVRDM decision boundaries for classes $\{C_3\}$ - $\{C_4\}$. Decision function values are greater than $T = 1$ inside the blue boundary and greater than $t = 0.4$ inside the magenta boundary. $\sigma = 0.6725$.

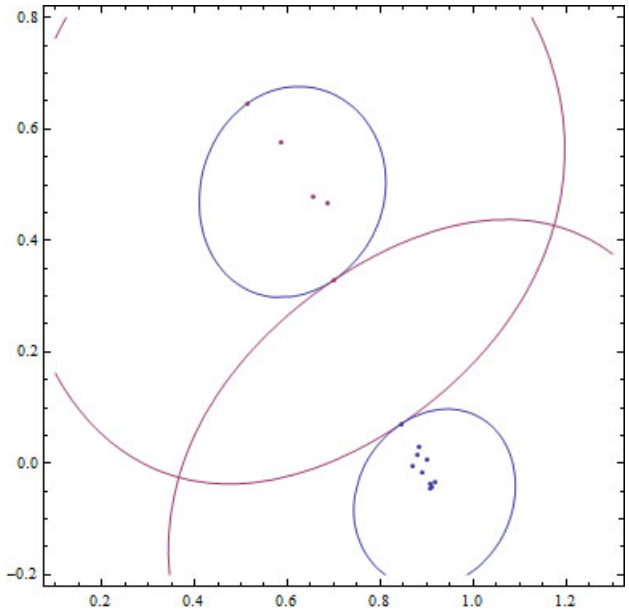

Figure 5.11: SVRDM decision boundaries for classes $\{C_6\}$ - $\{C_{11}\}$. Decision function values are greater than $T = 1$ inside the blue boundary and greater than $t = 0.4$ inside the magenta boundary. $\sigma = 0.42$.

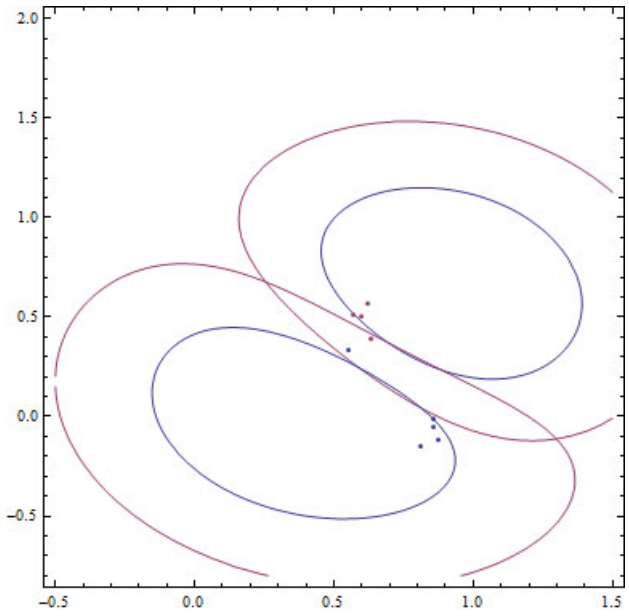

Figure 5.12: SVRDM decision boundaries for classes $\{C_8\}$ - $\{C_9\}$. Decision function values are greater than $T = 1$ inside the blue boundary and greater than $t = 0.4$ inside the magenta boundary. $\sigma = 0.5225$.

decision functions in a simple 2D plot. This helps a lot in understanding how individual nodes work and why certain classifications decisions are made; it also simplifies the debugging process and allows for rapid feedback while numerically tuning the training process. For exemplification, the decision boundaries and training samples involved in the classifier depicted by Figure 5.2 are given in Figures 5.3 to 5.12.

An area where the 2D visualization of SVRDM bounding boundaries was particularly helpful was in identifying a problem with the automatic estimation of σ (see Section 3.7.3). In the original formulation, the cluster radius R is empirically estimated as a fixed factor (2.2) of the average nearest-neighbor distance between the training samples. However, in some cases, the 2.2 value is too large (the samples are very well clustered together) and the estimated bounding boundary can be the empty set. To resolve this issue, Algorithm 4 was slightly modified to progressively decrease the factor in fixed steps (0.1), until the estimated bounding boundary becomes non-null.

To evaluate the quality of the hierarchical classifier developed in this section, the entire set of 122 motion samples was run through the classifier tree and the results were gathered in a confusion matrix (Table 5.5). Its columns represent the ground truth class labels and the rows symbolize the observed outcome; because the employed classifier design also has a good rejection capability, in addition to the set of 11 user-defined class labels, the set of labels in the observed outcome contains an additional symbol: the non-class (NC). This contains the samples that are rejected by the classifier as not known; any other cell (row, col) in the confusion matrix contains the number of samples belonging to class C_{col} classified as C_{row}.

The classifier performance can be quantified at a higher level by its classification rate (P_C), error rate (P_E) and respectively its rejection rate (P_R), defined as [44]:

$$\begin{aligned} P_C &= \frac{\#correctly\ classified\ samples}{\#total\ samples} \\ P_E &= \frac{\#incorrectly\ classified\ samples}{\#total\ samples} \\ P_R &= \frac{\#rejected\ samples}{\#total\ samples} \end{aligned} \tag{5.1}$$

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}
C_1	8	0	0	0	0	0	0	0	0	0	0
C_2	0	7	0	0	0	0	0	0	0	0	0
C_3	0	0	20	0	0	0	0	0	0	0	0
C_4	0	0	0	7	0	0	0	0	0	0	0
C_5	0	0	0	0	5	0	0	0	0	0	0
C_6	0	0	0	0	0	10	0	0	0	0	0
C_7	0	0	0	0	0	0	3	0	0	0	0
C_8	0	0	0	0	0	0	0	8	0	0	0
C_9	0	0	0	0	0	0	0	0	7	0	0
C_{10}	0	0	0	0	0	0	0	0	0	6	1
C_{11}	0	0	0	0	0	0	0	0	0	0	5
NC	4	5	0	3	6	6	0	2	1	4	4

Table 5.6: Confusion matrix for a hierarchical classifier trained with a mostly disjoint set of samples than the one in Figure 5.2.

In the case of the classifier developed in this section $P_C = 77.04\%$, $P_E = 0.81\%$, and $P_R = 22.13\%$.

To test how the hierarchical classifier performance is related to a particular selection of training data, a mostly disjoint set of samples was used to train a different classifier with the same numerical parameters as the previous one. In general, the samples that were omitted from training the former classifier were used to train the latter. An exception was again made in the case of *DribbleLeft*, where due to the reduced number of available motion data, the same set of samples was used in training both classifiers. The resulting confusion matrix for the new classifier is given in Table 5.6. It has a classification rate $P_C = 70.49\%$, error rate $P_E = 0.81\%$, and $P_R = 28.68\%$.

The variation in classification rate is most likely related to the fact that the training samples were not uniformly distributed among subjects; due to the fact that some subjects perform motions in different manners (i.e. with more ample arm movements, or starting from different initial poses), by omitting some subjects' samples from training, the variation in input feature vectors decreases, the SVRDM decision regions naturally shrink to fit the training data, and the classifier performance can be reduced. The fact that the error rate remains the same and only the rejection rate increases further suggests more compact SVRDM decision boundaries caused by an insufficient diversity of training samples.

It was mentioned in Section 4.5 that the feature vectors used in training the SVRDM nodes use Equation 3.37 as proposed by Li et al. [26]. However, after comparing several classification results, the eigenvectors entering the equation were no longer scaled by the normalized eigenvalues but instead translated and scaled such that each component falls inside the $[0, 1]$ interval. This results in a slightly more accurate classifier output, most likely because in the original formulation the eigenvectors $2, \cdots, k$ become small after scaling by eigenvalue and cause numerical difficulties in solving the SVM / SVRDM minimization problems [63].

5.4 Accuracy of Real-Time Motion Segmentation & Recognition

To evaluate the quality of the proposed architecture for real-time motion segmentation and recognition, the hierarchical classifier developed in Section 5.3 was applied to 29 motion clips belonging to 12 distinct human subjects. The motion clips contain a superset of the training samples but as said before, the classifier was trained on only half the available annotated data.

To quantify the quality of the segmentation, an approach similar to the definition of the *Rand Index* was preferred. For each ground truth motion segment, be it annotated or not, the best matching segment is located in the solution of the real-time segmentation. Two such segments are considered to match if their intersection on the time axis represents more than 50% of their minimum length; in other words, one of the segments must be contained in a proportion higher than 50% in the other segment. Thus being said, four situations may arise:

- **True Positive** - a ground truth segment matches a motion recognition segment and their classes also match;
- **False Positive** - a ground truth segment matches a motion recognition segment but their classes differ;
- **True Negative** - a ground truth segment with no assigned class does not match any motion recognition segment (in other words, the real-time motion recognition does not recognize a valid class and the ground truth indeed contains no class in the respective portion of the motion stream);

		Ground Truth	
		Positive	**Negative**
Observed Outcome	**Positive**	124	33
	Negative	21	36

Table 5.7: Real-time motion segmentation and recognition confusion matrix.

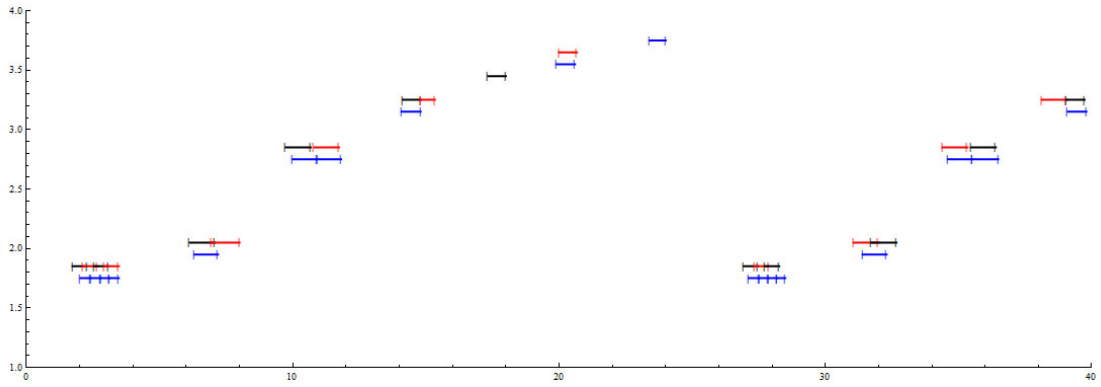

Figure 5.13: Motion segmentation result for clip 2 of subject 26. Blue segments represent the ground truth, alternating red and black segments are the result of the motion segmentation and recognition pipeline and motion class ID is encoded as y height.

- **False Negative** - a ground truth segment matches no motion recognition segment (i.e. the ground truth motion interval is labeled as belonging to a class, but the motion recognition failed to identify it).

The above mentioned technique was applied on each of the 29 motion clips and the resulting confusion matrix is given in Table 5.7. The accuracy of the real-time segmentation and classification can be estimated in the same way as RI, i.e. the ratio of correct decisions w.r.t. the total number of decisions taken:

$$Accuracy = RI = \frac{TP + TN}{TP + FP + TN + FN} \tag{5.2}$$

Therefore, the resulting accuracy of the real-time segmentation and classification technique using the classifier discussed in this section is 74.76%. The results of segmenting several motion files are shown in Figures 5.13 to 5.14. Blue segments represent the ground truth, alternating red and black segments are the result of the motion segmentation and recognition pipeline and motion class ID is encoded as y height.

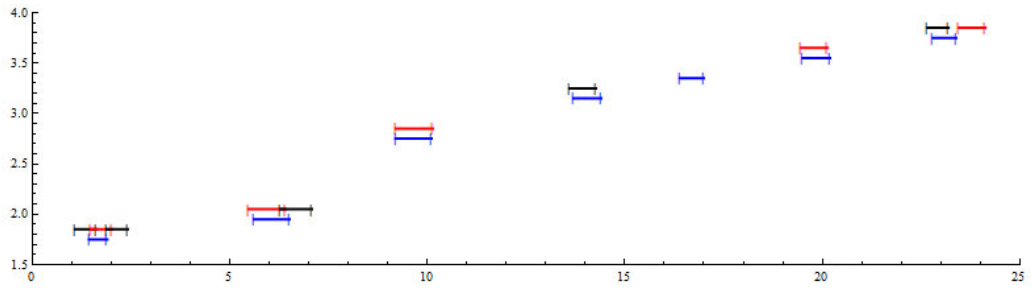

Figure 5.14: Motion segmentation result for clip 2 of subject 27. Blue segments represent the ground truth, alternating red and black segments are the result of the motion segmentation and recognition pipeline and motion class ID is encoded as y height.

5.5 Performance Analysis for MoRec on the Cell/B.E.

As mentioned in Section 4.7.3, MoRec uses gang contexts in order to minimize communication conflicts on the EIB. Therefore, the six motion segmentation and recognition pipeline stages are generally placed as in Figure 5.15. The RDS runs on the PPE and outputs data directly into the main memory, IA (SPE-1) is as close as possible to the MIC because it constantly needs to access raw motion data, SA (SPE-3) is placed immediately next to IA in order to minimize communication interferences with other stages, and so on.

The performance of the motion pipeline was evaluated using IBM's Full-System Simulator (SystemSim) [64] on several motion clips and the most important statistics are summarized in Table 5.8. Clocks per Instruction (CPI) represents the average number of cycles needed to execute an instruction. An SPE has two instruction pipelines implementing distinct operations and therefore is capable of issuing two instructions per cycle; the percentage of cycles that were used to execute a single instruction is given by the *Single cycle* line in the table, whereas the *Dual cycle* line lists the percentage of cycles where a dual issue was possible.

Further on, when a branch is reached, the SPE prepares to execute a statically predicted code path, as hinted by the compiler. However, if the prediction is wrong, the SPE needs to flush the instruction pipelines and run the code following the correct branch. This has the effect of stalling program execution and the percentage of cycles lost due to branch miss stalls is also given in Table 5.8. Finally, the table also lists the percentage of cycles lost due to dependency stalls - these are caused by instruction parameters that are not yet available at the moment an instruction is about to be executed.

It is worth mentioning that for all six SPEs, the percentage of cycles wasted on waiting for channel operations is 0.0%, therefore the time spent in SPE communication (i.e. DMA transfers, mutex acquisition) is not affecting the performance of the MoRec pipeline. As pointed out in Table 5.8, the most important performance related factors are poor dual-issue rate (6.5% on average) and relatively high branch miss and dependency stalls which negatively impact CPI; in general, the goal is to achieve subunitary CPI values (0.7 - 0.9).

While branch misses can be reduced by allowing the compiler to use run-time information in static branch prediction, the improvement in dual issue rate and dependency stalls is more involved. For instance, the FVE stage computes the SVD decomposition with scalar code that incurs a lot of overhead to position the operands in the SIMD registers. In general, scalar operations involves various shuffle operations for proper operand positioning and in the case of MoRec, it accounts for more than 35% of the dependency stalls.

Another issue negatively affecting MoRec SPE performance is the fact that the code is written in C++ using object-oriented programming (OOP). This translates into additional overhead when calling member functions or addressing member variables, but also implies better code readability and reuse. For instance, the DMA communication primitives (circular buffer and associated DMA manager) were implemented as classes and the same code was reused for all SPEs and the PPE. It would be possible to modify the SPE stages to use C-style functions in order to further increase performance, but this would make the code very hard to maintain, an undesirable characteristic for a proof-of-concept application.

On the positive side, it is worth mentioning that MoRec is capable of processing on average 4487 motion frames per second, which is approximatively 37 times faster than a usual MoCap device providing data at 120Hz. The processing speed can be expected to deteriorate for larger classifiers, with more support vectors and motion vectors to enter the computation, but there is still a large enough margin to allow for an order of magnitude in performance drop. Also, given the fact that MoRec is geared towards human motion recognition, in the case of poor performance (for instance a large classifier involving frequent memory to LS DMA transfers for support vectors / motion vectors and causing a lot of channel stalls) the Mocap data source can be sampled at a lower resolution (i.e. 60Hz) thus reducing the workload by half while in the same time preserving a relatively high quality of the captured motion.

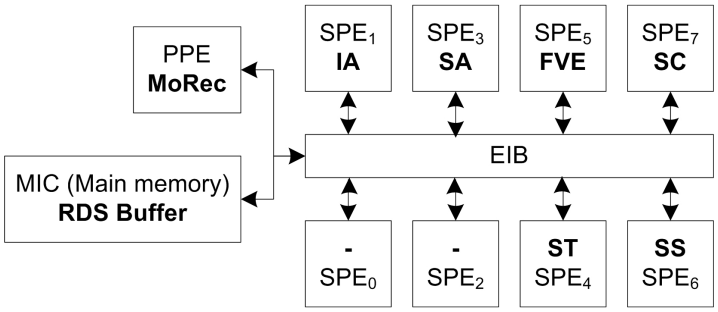

Figure 5.15: Typical placement of MoRec pipeline stages on the Cell/B.E.

Stage	**IA**	**SA**	**FVE**	**SC**	**SS**	**ST**
SPE ID	1	3	5	7	6	4
Code & data size (KB)	148.33	126.84	149.22	140.08	147.73	102.08
Free space / stack size (KB)	107.67	129.16	106.78	115.92	108.27	153.92
CPI	1.84	1.92	1.82	1.86	1.21	2.02
Single cycle	37.50%	36.30%	37.7%	37.5%	49.7%	35.5%
Dual cycle	5.30%	5.1%	5.7%	5.2%	13.3%	4.5%
NOP cycle	3.20%	3.3%	3.2%	3.3%	1.6%	2.8%
Branch miss stalls	23.20%	23.8%	17.7%	20.5%	0.7%	24.3%
Dependency stalls	29.30%	29.6%	33.9%	31.5%	34.7%	30.8%

Table 5.8: MoRec performance statistics on the Cell/B.E. captured with SystemSim.

Chapter 6

Conclusions & Future Work

The primary goal of this project was to create a representation (index) of a motion database that would allow an application to scan a potentially infinite stream for known motion categories (i.e. cartwheel, golf swing, etc.) in real-time. The first step was to develop a motion database management application; this allows a human user to group together various motion data files, define portions of interest inside the files, gather similar motion segments into classes, and also train and test classifiers capable of recognizing the defined motion classes.

Typically, motion files inside a database contain multiple logically distinct motions and in order to minimize user interaction in identifying them, an automatic technique based on PPCA [12] was implemented inside the motion database editor. This has proven to be inadequate due to the fact that it does not discover individual instances in the case of cyclic / repeated motions and also because it includes transition / idle frames in the returned segments. The problem is however of great importance in the case of large databases where manually annotating motion clips can become prohibitive, and thus requires further investigation. As future work, it would be interesting to test whether a technique based on partially training a classifier for a manually annotated subset of the database and using it to segment the remaining set of clips proves to be a better choice. However, such a technique would still not be able to identify segments belonging to untrained motion classes and would still require manual intervention.

After motion segments have been either automatically or manually defined, the next impor-

tant operation is to group them into motion classes by similarity. Again, in the spirit of minimizing user intervention, the motion database manager implements an automatic estimation of the number of classes using the $kWAS$ similarity measure [25]. However, the resulting set of clusters does not produce a solution sufficiently close to the ground truth and other clusterization techniques should be further investigated as future work. For instance it would be interesting to test whether a SVM-based hierarchical clustering method [65] with user control over the hierarchy depth would provide a better solution. This would still require user intervention but it would be minimal, consisting in stopping further single-class cluster subdivisions. In the same spirit, the algorithm that builds the hierarchical classifier already performs class clusterization; it can be modified to consider each training sample as belonging to a unique class and split the samples into a binary tree where each node best separates its samples. Again, the discovered class hierarchy and division process can be controlled by user intervention.

As the motion classes and associated motion samples have been defined, the database manager allows the creation of a SVRDM-based hierarchical classifier [52] capable of recognizing trained motion categories. The resulting classifier achieves good results and is also capable of rejecting untrained motion classes. Its only downside is sensitivity to different human subject versions of the same motion, but that is more a problem related to the set of training data than the actual classifier design and therefore it is easily overcome by using a more diversified set of training samples. An interesting direction of further development is to devise a procedure of estimating the thresholds T, t used in evaluating the SVRDMs, as they must be a little lower, respectively higher than the values used in training in order to improve the generalization capability. However, if for instance T is too low, the SVRDM classification ability is also negatively affected, and therefore it is important to strike a balance between the values used in training / evaluation in order to maximize the generalization ability and keep the classification accuracy high.

Finally, as soon as the hierarchical classifier is created by the database manager, it can be exported to a binary file as the database *index*. This is a very small file (approx. 50KB for 5MB of motion data) containing the hierarchical classifier tree together with the support and motion vectors, and is used by the real-time motion segmentation and recognition component running on the Cell/B.E. (MoRec) to split a motion data stream into known portions belonging to previously trained classes. The main goal of the Cell/B.E. implementation was to run

at real-time rates and the proposed architecture successfully does so, being able to process more than 4000 frames of motion per second while a MoCap device typically produces data at 120Hz.

As mentioned before, the architecture of MoRec is modular and consists of several stages organized in a pipeline. In the test cases analyzed above, the classifier data was able to fit entirely in the SPE caches and therefore the communication between the pipeline stages and the main memory was minimal. However, this is not always possible, and for large classifier configurations it is highly probable that support vectors and motion vectors need constant transfer from main memory to SPE LS via DMA, altering the performance of the other stages. In order to accommodate for this undesirable scenario, the classification stage can be easily broken down and distributed on several SPEs, such that each handles a subtree small enough to fit entirely in the SPE LS cache. The same reasoning can be applied to the segment scoring stage, where several copies of it handling different classes can be run simultaneously, in order to minimize communication with the main memory. As future work, it would be interesting to investigate the overall performance of such an architecture, and whether it is still fit for real-time motion segmentation and recognition. Another interesting development would be to implement an interface to a real MoCap data source and test how the proposed architecture is affected by it, whether it needs specific filtering of the raw data, data conversions, and so on.

To summarize, the proposed architecture for real-time motion segmentation and recognition performs well on the Cell/B.E. from both the perspective of performance and overall quality and accuracy of the returned results. Further tests with larger classifiers as well as an implementation of a real MoCap data source are needed in order to show potential problems (if any) in the implementation of MoRec. The motion database manager application developed to generate the classifier allows a human user to easily perform various operations on motion clips but nevertheless, more research needs to be done in order to minimize user intervention and automate critical processes running on large datasets.

Appendix A - List of Acronyms

API Application Programming Interface ix

BVH Biovision Hierarchy vi

CPI Clocks per Instruction 79

DMA Direct Memory Access 55

DOF degree of freedom 4

EIB Element Interconnect Bus 55

FVE Feature Vector Extraction vii

GMM Gaussian Mixture Model 6

HMM Hidden Markov Model 8

IA Input Assembly vii

ICA Independent Component Analysis 5

LS Local Store 55

MFC memory flow controller 55

MIC main memory interface controller 55

MoCap Motion Capture v

OOP object-oriented programming 80

PCA Principal Component Analysis 4

PPCA Probabilistic Principal Component Analysis 6

PPE PowerPC Processor Element 55

QPP Quadratic Programming Problem 27

RDS Raw Data Source vii

SA Segment Assembly vii

SC Segment Classification vii

SIMD Single Instruction Multiple Data vii

SOR successive over-relaxation 28

SPE Synergistic Processor Element 50

SS Segment Scoring 60

ST Segment Tracking 60

SVD Singular Value Decomposition 8

SVM Support Vector Machine 7

SVRDM Support Vector Representation and Discrimination Machine vii

SVRM Support Vector Representation Machine 30

XDR eXtreme Data Rate 55

Bibliography

[1] Sony Computer Entertainment Europe, "EyeToy®." http://www.eyetoy.com/.

[2] Sony Computer Entertainment Inc., "The Eye of Judgment™." http://www.us.playstation.com/EyeofJudgment/.

[3] 3DV Systems, "ZCam." http://www.3dvsystems.com/.

[4] CMU, "Carnegie-Mellon Motion Capture Database." http://mocap.cs.cmu.edu.

[5] D. Pham, S. Asano, M. Bolliger, M. Day, H. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa, "The design and implementation of a first-generation cell processor," in *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, pp. 184–592 Vol. 1, Feb. 2005.

[6] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, vol. 2, no. 6, pp. 559–572, 1901.

[7] H. Hotelling, "Analysis of a complex of statistical variables into principal components," *J. Edu. Psych.*, vol. 24, pp. 417–441, 1933.

[8] G. Liu, J. Zhang, W. Wang, and L. 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*, (New York, NY, USA), pp. 924–926, ACM, 2005.

[9] G. Sukthankar and K. Sycara, "A cost minimization approach to human behavior recog-

nition," in *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, (New York, NY, USA), pp. 1067–1074, ACM, 2005.

[10] P. Beaudoin, S. Coros, M. van de Panne, and P. Poulin, "Motion-motif graphs," in *Symposium on Computer Animation 2008*, pp. 117–126, July 2008.

[11] Z. Deng, Q. Gu, and Q. Li, "Perceptually consistent example-based human motion retrieval," in *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, (New York, NY, USA), pp. 191–198, ACM, 2009.

[12] J. Barbič, A. Safonova, J.-Y. Pan, C. Faloutsos, J. K. Hodgins, and N. S. Pollard, "Segmenting motion capture data into distinct behaviors," in *GI '04: Proceedings of Graphics Interface 2004*, (School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada), pp. 185–194, Canadian Human-Computer Communications Society, 2004.

[13] O. Arikan, "Compression of motion capture databases," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 890–897, 2006.

[14] J.-Y. Pan, H. Kitagawa, C. Faloutsos, and M. Hamamoto, "Autosplit: Fast and scalable discovery of hidden variables in stream and multimedia databases," in *Proceedings of the The Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2004)*, 2004.

[15] P. Comon, "Independent component analysis, a new concept?," *Signal Process.*, vol. 36, no. 3, pp. 287–314, 1994.

[16] A. Hyvärinen, J. Karhunen, and Oja, *Independent Component Analysis*. John Wiley & Sons, 2001.

[17] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction.," *Science*, vol. 290, pp. 2319–2323, December 2000.

[18] O. C. Jenkins and M. J. Matarić, "A spatio-temporal extension to isomap nonlinear dimension reduction," in *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, (New York, NY, USA), p. 56, ACM, 2004.

[19] O. C. Jenkins and M. J. Matarić, "Performance-derived behaviour vocabularies: data-driven acquisition of skills from motion," vol. 1, pp. 237–288, 2004.

[20] N. Lawrence, "Probabilistic non-linear principal component analysis with gaussian process latent variable models," *J. Mach. Learn. Res.*, vol. 6, pp. 1783–1816, 2005.

[21] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović, "Style-based inverse kinematics," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 522–531, 2004.

[22] A. Fod, M. J. Mataric, and O. C. Jenkins, "Automated derivation of primitives for movement classification," *Autonomous Robots*, vol. 12, pp. 39–54, 2002.

[23] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society, Series B*, vol. 61, pp. 611–622, 1999.

[24] R. Billon, A. Nédélec, and J. Tisseau, "Gesture recognition in flow based on pca analysis using multiagent system," in *ACE '08: Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, (New York, NY, USA), pp. 139–146, ACM, 2008.

[25] C. Li, S. Q. Zheng, and B. Prabhakaran, "Segmentation and recognition of motion streams by similarity search," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 3, no. 3, p. 16, 2007.

[26] C. Li, P. R. Kulkarni, and B. Prabhakaran, "Segmentation and recognition of motion capture data stream by classification," *Multimedia Tools Appl.*, vol. 35, no. 1, pp. 55–70, 2007.

[27] C. Li, G. Pradhan, S. Q. Zheng, and B. Prabhakaran, "Indexing of variable length multi-attribute motion data," in *MMDB '04: Proceedings of the 2nd ACM international workshop on Multimedia databases*, (New York, NY, USA), pp. 75–84, ACM, 2004.

[28] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh, "Indexing multidimensional time-series with support for multiple distance measures," in *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 216–225, ACM, 2003.

[29] G. Qian, F. Guo, T. Ingalls, L. Olson, J. James, and T. Rikakis, "A gesture-driven multimodal interactive dance system," vol. 3, pp. 1579–1582 Vol.3, June 2004.

[30] F. Bashir, W. Qu, A. Khokhar, and D. Schonfeld, "Hmm-based motion recognition system using segmented pca," vol. 3, pp. III–1288–91, Sept. 2005.

[31] X. Zhang and F. Naghdy, "Human motion recognition through fuzzy hidden markov model," in *CIMCA '05: Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-2 (CIMCA-IAWTIC'06)*, (Washington, DC, USA), pp. 450–456, IEEE Computer Society, 2005.

[32] C. Li, L. Khan, and B. Prabhakaran, "Real-time classification of variable length multi-attribute motion data," *International Journal of Knowledge and Information Systems (KAIS)*, vol. 10, no. 2, pp. 163–183, 2006.

[33] W. Wan, H. Liu, L. Wang, G. Shi, and W. Li, "A hybrid hmm/svm classifier for motion recognition using ?imu data," in *Robotics and Biomimetics, 2007. ROBIO 2007. IEEE International Conference on*, pp. 115–120, Dec. 2007.

[34] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Gaussian process dynamical models," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 18, pp. 1441–1448, MIT Press, 2005.

[35] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Gaussian process dynamical models for human motion," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, no. 2, pp. 283–298, 2008.

[36] Bruce Hahne, "The MotionBuilder-friendly BVH Conversion Release of CMU's Motion Capture Database." `http://sites.google.com/a/cgspeed.com/cgspeed/motion-capture/cmu-bvh-conversion`.

[37] O. Onder, U. Gudukbay, B. Ozguc, T. Erdem, C. Erdem, and M. Ozkan, "Keyframe reduction techniques for motion capture data," pp. 293–296, May 2008.

[38] D. Salomon, *Curves and Surfaces for Computer Graphics*. Berlin, Germany / Heidelberg, Germany / London, UK / etc.: Springer-Verlag, 2006.

[39] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press, 2007.

[40] I. T. Jolliffe, *Principal Component Analysis*. Springer, second ed., October 2002.

[41] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971.

[42] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning)*. The MIT Press, December 2001.

[43] S. Abe, *Support Vector Machines for Pattern Classification*. Springer-Verlag London Limited, 2005.

[44] C. Yuan and D. Casasent, "Support vector machines for class representation and discrimination," in *Neural Networks, 2003. Proceedings of the International Joint Conference on*, vol. 2, pp. 1611–1616 vol.2, July 2003.

[45] V. Kecman, T.-M. Huang, and M. Vogt, "Iterative single data algorithm for training kernel machines from huge data sets: Theory and performance," in *PERFORMANCE, SUPPORT VECTOR MACHINES: THEORY AND APPLICATIONS, SPRINGER-VERLAG,.STUDIES IN FUZZINESS AND SOFT COMPUTING*, pp. 255–274, Springer Verlag, 2005.

[46] S. S. Keerthi and C.-J. Lin, "Asymptotic behaviors of support vector machines with gaussian kernel," *Neural Comput.*, vol. 15, no. 7, pp. 1667–1689, 2003.

[47] C. H. Park and H. Park, "Efficient nonlinear dimension reduction for clustered data using kernel functions," in *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, (Washington, DC, USA), p. 243, IEEE Computer Society, 2003.

[48] G. H. Golub and C. F. Van Loan, *Matrix computations (3rd ed.)*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[49] A. Ben-hur, D. Horn, H. Siegelmann, and V. Vapnik, "A support vector method for hierarchical clustering," in *In Advances in Neural Information Processing Systems 13*, pp. 367–373, MIT Press, 2001.

[50] R. Akbani, S. Kwek, and N. Japkowicz, "Applying support vector machines to im-

balanced datasets," in *In Proceedings of the 15th European Conference on Machine Learning (ECML*, pp. 39–50, 2004.

[51] T. Imam, K. Ting, and J. Kamruzzaman, "z-svm: An svm for improved classification of imbalanced data," pp. 264–273, 2006.

[52] Y.-C. F. Wang and D. Casasent, "2008 special issue: New support vector-based design method for binary hierarchical classifiers for multi-class classification problems," *Neural Netw.*, vol. 21, no. 2-3, pp. 502–510, 2008.

[53] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 491–502, ACM, 2005.

[54] P. Wrotek, O. C. Jenkins, and M. McGuire, "Dynamo: dynamic, data-driven character control with adjustable balance," in *sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, (New York, NY, USA), pp. 61–70, ACM, 2006.

[55] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, third ed., 1999.

[56] I. S. Dhillon, *A new O (N(2)) algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem*. PhD thesis, Berkeley, CA, USA, 1998.

[57] I. S. Dhillon, B. N. Parlett, and C. Vömel, "The design and implementation of the mrrr algorithm," *ACM Trans. Math. Softw.*, vol. 32, no. 4, pp. 533–560, 2006.

[58] Luca Di Gaspero, "QuadProg++." `http://www.diegm.uniud.it/digaspero/index.php?page=software-and-data`, 2009.

[59] A. Marzal and E. Vidal, "Computation of normalized edit distance and applications," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 15, pp. 926–932, Sep 1993.

[60] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick, "The potential of the cell processor for scientific computing," in *CF '06: Proceedings of the 3rd conference on Computing frontiers*, (New York, NY, USA), pp. 9–20, ACM, 2006.

[61] M. Kistler, M. Perrone, and F. Petrini, "Cell multiprocessor communication network: Built for speed," *Micro, IEEE*, vol. 26, pp. 10–23, May-June 2006.

[62] M. Scarpino, *Programming the Cell Processor: For Games, Graphics, and Computation*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2008.

[63] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[64] IBM, "Full-System Simulator User's Guide ." `https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/B494BF3165274F67002573530070049B?Open`, 2007.

[65] A. Ben-hur, D. Horn, H. Siegelmann, and V. Vapnik, "A support vector method for hierarchical clustering," in *In Advances in Neural Information Processing Systems 13*, pp. 367–373, MIT Press, 2001.