Department of Electrical
and
Computer Systems Engineering

Technical Report
MECSE-30-2005

Multiple Cameras Human Motion Capture using Non-linear Manifold Learning and De-noising

T. Tangkuampien

Multiple Cameras Human Motion Capture using Non-Linear Manifold Learning and De-noising

T Tangkuampien[*]
Technical Report
Supervisors: A/Prof David Suter and Prof Ray Jarvis
Department of Electrical and Computer Systems Engineering [†]
Monash University, Melbourne, Australia

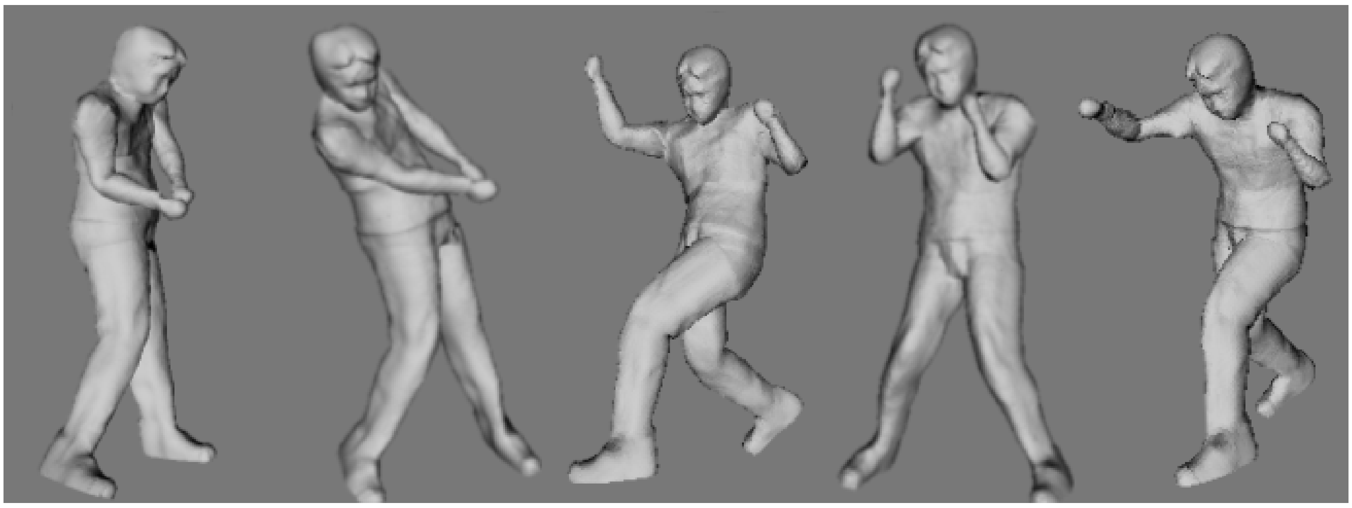

Figure 1: Different Poses from a Motion Capture Database re-targeted to Skinned Mesh Models of the author

Keywords: Motion Capture (**MOCAP**) Database, Skinned Mesh, Principal Component Analysis (PCA), Kernel PCA and de-noising, Manifold Learning, Locally Linear Embedding (**LLE**), Support Vector Machines (**SVM**) Classification and Regression, Positive semi-definite Kernel Matrix, Radial Basis Functions (**RBF**).

1 Introduction

The ability to realistically capture human motion presents numerous opportunities for real world applications, ranging from human computer interaction interfaces to computer animation and control in movies and computer games. Currently to accurately capture human motions, magnetic or optical markers are systematically attached to an actor and an expensive calibrated system is used to capture the positions of these markers as the actor performs the required motions. The main disadvantages of attaching markers are the restriction imposed on the actor's motion and the cost of a system specifically designed to track the markers. The aim of this research is to develop a multiple cameras capturing system that will be able to realistically capture human pose[1] without the use of any markers.

[*]e-mail: therdsak.tangkuampien@eng.monash.edu.au

[†]This technical report was written to fulfil the requirements for a conversion from a research masters candidature in Engineering Science to the degree of Doctor of Philosophy. The author has been a postgraduate student by research at the Department of Electrical and Computer Systems Engineering, Monash University since June 2004. This research work is conducted under the main supervision of A/Prof David Suter. Institute of Vision System Engineering, Monash University. Results and updated information can be found at: http://www-personal.monash.edu.au/∼therdsak/postGrad.htm

[1]Pose - a collection of hierarchical joint angles defining the stance of a person at a point in time - Acclaim Motion Capture format, Section 3.1

The research aims to investigate the possibility of applying non-linear manifold learning techniques like Locally Linear Embedding (LLE) and Kernel Principal Component Analysis to aid in motion capturing. An accurate 3D mesh of the actor (Figure 1), created from point cloud data captured by a laser scanner is used to generate synthetic 3 dimensional representation of the actor in virtual space. A large set of poses ranging the space of possible human motion is then used to animate the mesh and the resultant images captured by virtual cameras. The virtual cameras are calibrates to have the same intrinsic and extrinsic parameters as the real-world cameras that will be used for motion capture. Provided that the synthetic image data is well sampled, the set of all possible images of the mesh should lie on a common lower dimensional manifold as the one generated by the set of possible human poses. This intuition is based on the idea that in a constant and controlled environment, the images of an actor captured by cameras with static intrinsic and extrinsic parameters should mainly be dependent on the current pose (joint angles) of the person at that particular frame. Given a new set of real images of the actor at time **t**, the system can then project the captured image data onto the synthetic common manifold. Once on the manifold, the poses that produce the closest set of synthetic images to the captured images can be used to determined the output pose.

The main steps performed within our markerless camera based motion capture system are summarized below:

1. **Generate an Accurate Mesh of the Actor - Section 4** - In the initialization step, a laser scanner is used to obtained accurate point cloud data of the actor. Radial Basis functions are fitted to the point cloud and resampled to generate a static mesh of the actor in virtual reality space.

2. **Create a Deformable Mesh - Section 3** - A generic skeleton structure is placed inside to mesh. The mesh is then skinned and converted to a deformable mesh. After skinning, the mesh can be animated by loading the motion capture pose onto the inner skeleton structure.

3. **Generate Training Data with the Mesh** - A training motion capture file containing sampled set of poses ranging all the possible stances reachable by the human skeleton are generated. Each pose in the training set is loaded into the deformable mesh to generate its 3 dimensional representation in virtual space.

4. **Set up Virtual Cameras in Virtual Reality Space** - Virtual cameras with the same intrinsic and extrinsic parameters as the real cameras (for motion capture) are created in virtual reality. For each pose, the set of synthetic images of the 3D mesh as it would appear in the real cameras are captured and linked to the original pose. All the other parameters like lighting, camera positions, etc are kept constant during the training process.

5. **Generating the Common Manifold** - Determine the **common** manifold shared by the synthetic images and the set of training poses. This is possible because each set of synthetic images captured in the virtual cameras is mainly dependant on the pose that was loaded into the mesh (all the other variables are kept constant).

6. **Pose Estimation from Manifold Projection** - Once the system has been trained with synthetic data, real images of the person can be captured by real cameras. The captured images can then be preprocessed and projected onto the common manifold, where it can find it closest **k** neighbouring poses and the relative distances between them on the manifold. The output pose can then be generated as a linear combination of the **k** closest neighbours in pose space.

2 Literature Review

The problem of accurately capturing human motion with expensive equipment and sensors is a well solved one. The main disadvantages with these methods are the cost and time required to set up the motion capture system. Sensors need to also be connected to the actor and calibrated before capture, not to mention the constraints imposed on the actor as a result of these sensors. Since the aim of this project is to implement a flexible marker-less capture system, only the literature in this field will be discussed. However in the future it will be useful to compare the results from a marker-less system, with those of other marker based systems.

2.1 Multiple Cameras Motion Capture

The first few problems usually encountered when trying to implement an accurate camera based capturing system are those of occlusion and loss of dimensionality, because the goal of such a capturing system is to find 3D poses from 2 dimensional images. To avoid these problems, multiple cameras are set up surrounding the actor, and the set of images at frame $\boldsymbol{t}$ are segmented and processed to provide voxel information [Mikić et al. 2002][Theobalt et al. 2002]. The obvious advantage of such a system is the reduced constraints on the possible motions of the actor, as he will not have any sensors attached. However, the cameras will need to be well calibrated and will involve complex set up procedures every time the cameras are re-positioned. Furthermore, because voxels are usually generated from segmented pixels in the image space, the quality of the voxel carved out will also be dependant on the segmentation. The manifold motion capture technique that we plan to implement will also initially work with segmented data. However we plan to investigate the possibility of de-noising using manifold projection of badly segmented (noisy) data. We propose that if perfectly segmented synthetic data are used to generate the training manifold of possible motions, then any new input with noise (including segmentation noise) can be projected onto the manifold for noise reduction.

From the voxel or segmented data, a generic skeleton stick figure adjusted to the most probable pose is fitted into the voxel space. A tracking filter, such as a Kalman filer [Welsh and Bishop 2001] is then applied to the joints of the skeleton from prediction and correction, or to implement a form of **Analysis by Synthesis**. Analysis by synthesis is a popular method in motion capture, where the pose obtained is applied to a human model and mapped back to the image space. In this space, the synthesized data is compared to the input images and the resultant pose corrected in order to minimize a predefined loss function. Another common constraint used by many segment based motion capturing systems ([Theobalt et al. 2002],[Moeslund and Granum 2003]) is to limit the pose captured to within the range of possible joint rotations reachable by the human skeleton. An example of this restriction would be to limit the hinge joint of the elbow to have only one degree of freedom.

Other factors affecting the accuracy of a capturing system with feedback for error minimization are the model used in tracking, and how closely the bone structure and model's outline matches that of the person being captured. With advances in processors and graphics devices, it is now possible to generate an accurate mesh of a person and deform the mesh to create realistic human animation [Luna 2004]. This report briefly covers the process of obtaining this accurate mesh model (Figure 4) and how a generic skeleton can be placed inside the model to create a dynamic skinned mesh (Section 4 - Mesh Generation and Skinning). From an analysis by synthesis point of view, it is now possible to use the skinned mesh to generate accurate synthetic silhouette models, which can then be used in the calculation of the errors in the image space.

2.2 Statistical Analysis of Human Motions

Due to the high level of co-ordination in human movements, a high percentage of human motions will intrinsically lie on a low dimensional manifold. [Safonova et al. 2004] and [Bowden et al. 2000] both use Principal Component Analysis (PCA) in order to represent the motion data using a reduced numbers of principal components. At this point it is important to point out the difference between the two. [Safonova et al. 2004] applied PCA to the joints angles of the already captured motion over time, in order to determine a low dimensional manifold of a motion set. [Bowden et al. 2000], and [Ham et al. 2003](to a different area of research) both applied PCA in pixel space of an image of a person at time $\boldsymbol{t}$ to represent the pixel data in lower dimensions D, where $D \ll P_n$ (number of pixels in the image) so that any further data processing can be performed in this reduced dimension. Since the goal of this research is to determine the pose (joint angles) of a person from images, only related literature in the area of Principal Component Analysis on image data will be discussed.

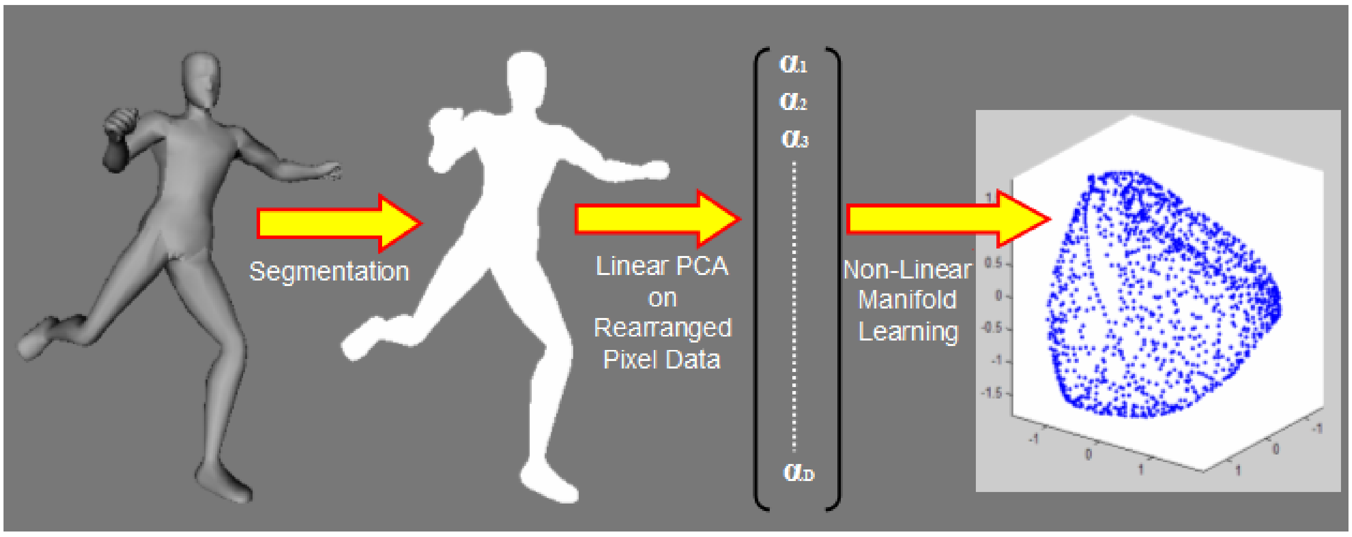
Segmentation
Linear PCA on Rearranged Pixel Data
α_1
α_2
α_3
α_D
Non-Linear Manifold Learning

Figure 2: Diagram to show how the pixel data in the images representation the pose of a person can be reduced by linear PCA and projected onto a lower dimensional manifold. The manifold data shown on the right is generated from a well sampled set of images of an actor rotating his left arm.

The problem of human motion capture using cameras can also be viewed as an semi-supervised learning problem. We can use a 3D mesh of the actor and generate synthetic images of him/her as they would appear in the cameras. These images are then segmented, cropped, resized and rearrange to obtain a set of high dimensional training vectors. PCA is then performed on this set of high dimensional synthetic vectors (Figure 3 - synthetic training set). The first D principal components where variance of the projected training data are more than zero are retained. Given new images of the actor captured from real cameras, these images can be pre-processed using the same steps as applied to the synthetic data and projected onto the D principal components.

Once re-expressed on the principal axis, a non-linear method like Support Vector Machines (SVM [Vapnik 1995],[Cristianini and Shawe-Taylor 2000]) regression can be used to map the data onto an even lower dimensional manifold d, where $d \ll D$. This is obviously provided that we have enough corresponding data points to train the SVM. However, the usual problem encountered when using SVM is how to go about selecting the kernel function and it's parameters so that the training data can be mapped to an optimal feature space in high dimension for fitting. Another potential obstacle with SVM when applied to this problem is that we are trying to map high dimensional segmented pixel data of a person to high

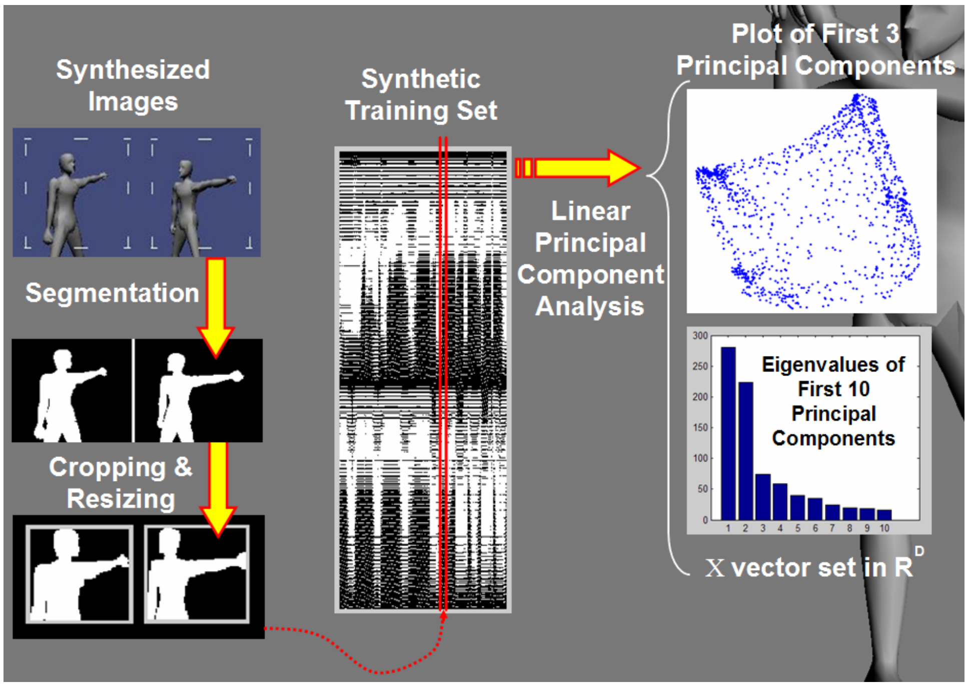
Synthesized Images
Synthetic Training Set
Plot of First 3 Principal Components
Segmentation
Linear Principal Component Analysis
Cropping & Resizing
Eigenvalues of First 10 Principal Components
X vector set in R^D

Figure 3: Diagram to summarize how the synthetic data is preprocessed before Principal Component Analysis is preformed on on the rearranged pixel data.

dimensional pose space (approximately 28 dimension of angles for the human body) representing the person's stance. In order for regression to work effectively, there needs to be a smooth mapping function between these two high dimensional space. For all we know this problem may not even be well-posed and a smooth mapping may not even exist.

A usual way to avoid the complex problem of determining the kernel function and its parameters for the SVM is to bypass the Kernel function calculation completely and go straight to finding the Gram matrix K, comprising of the kernel mappings between the training points[2]. Manifolds learning methods like Locally Linear Embedding (LLE)[Saul and Roweis 2003] and Kernel Matrix Learning using Semidefinite programming (SDE)[Weinberger and Saul 2004][Weinberger et al. 2005] can be used to determine this Gram matrix, which can then be used to determine the lower dimensional manifold. The immediate advantage of performing data processing in low dimensional space is obvious in terms of memory consumption and speed. Another not so obvious advantage is the ability to use the manifold to eliminate noise in the images, as shown by [Mika et al.] via the use of Kernel PCA [Scholkopf and Smola 2002]. By assuming that a new noisy input image will map to a lower dimensional space not on the manifold of the training set, this report shows how to project the new noisy input point onto the manifold (using LLE) for noise reduction and presents how this is similar to kernel PCA de-noising as well as results to support this assumption in terms of motion capture.

3 Motion Capture File Format

This section covers the desired format of the output data from a typical multiple camera motion capture system. Standard motion capture file format usually stores rotation transformation for each articulated

[2]The simplest case of the Kernel matrix is one where the kernel function is a simple dot product in input space. A value in the Kernel Matrix in this case is simply just a way of comparing how close two vectors are by linearly projecting one onto another - more details will be covered in Section 5

bone relative to a start pose. A standard start pose used in this research is a **Tpose** with the actor standing with arms stretched out horizontally. In order to achieve animation, the joint rotations from each frame **t** are applied to each bone of the skeleton hierarchically starting from the root (pelvis) bone. The resultant images are then rendered at high speed onto the screen. Due to the hierarchical structure of the skeleton, only rotation and translation information need to be applied to the root bone (pelvis) in order to move or orientate the entire skeleton. As shown in Figure 4, the main goal of an image based motion capture system is to then determine these joint rotation data from images of a person and store it in a file structure so that it can be applied to a skeletal model for animation. Note that in the diagram, we included a segmented outline of a person as the image to use to determine the joint rotation. This, however, does not mean that the image based systems are only limited to working with segmented data.

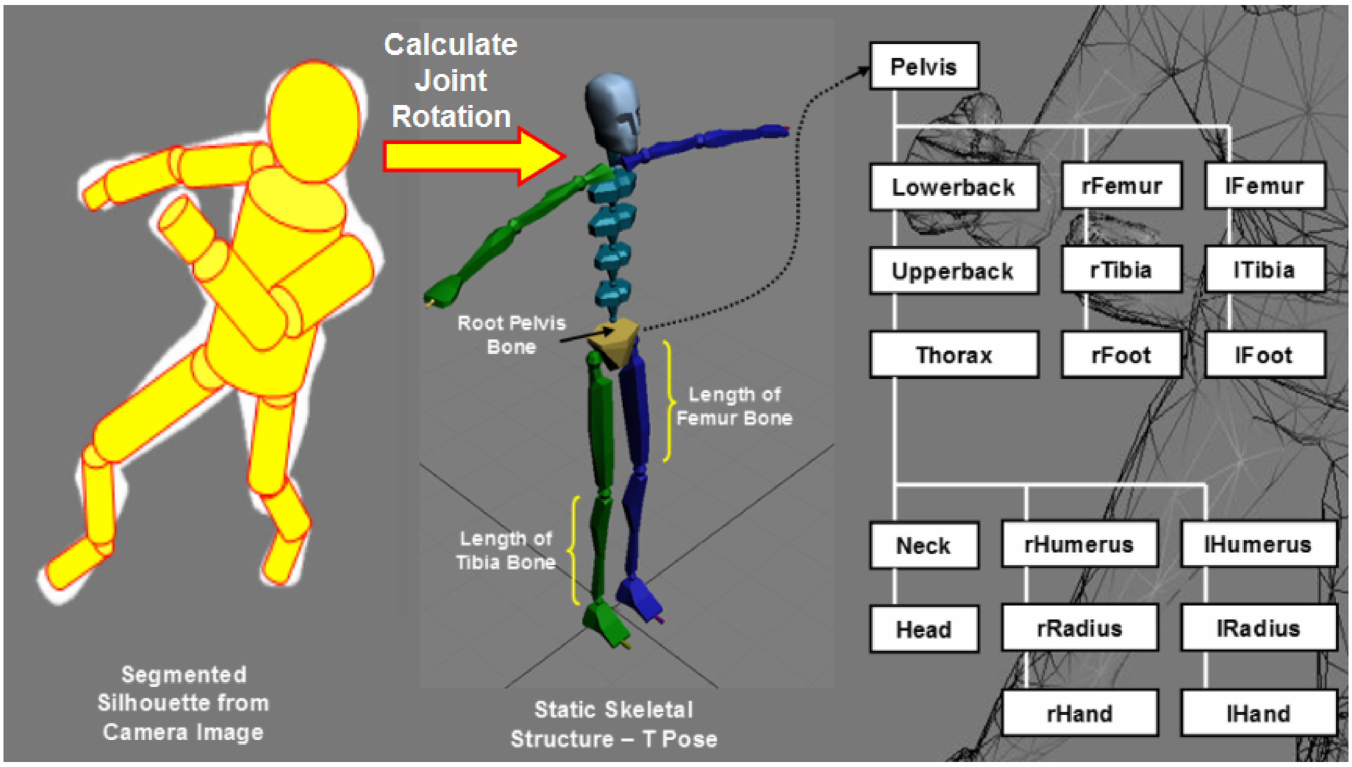

Figure 4: Images to show the input (left) and the output format (right) of an image based motion capturing system.

There are 2 motion formats that are used in this project to store hierarchical rotation motion, the Acclaim Motion Capture and the DirectX format. AMC format was chosen due to high numbers of marker-based motions that are already available at the Carnegie Mellon University Graphics Lab Motion Capture Database and various other web sites. DirectX format was chosen because of its ability to deform a skinned mesh at high speed using built in DirectX graphic functions. The advantage of this deformable mesh will become clear at a later stage, when we try to generate training data for motion capture in Section 5. Both formats, however, are similar because they use the same skeletal hierarchical structure starting from the root (pelvis) node to the leave nodes at the hands and feet, and store their rotation information relative to a start pose. The only standout difference between the two formats is the way these relative rotation data are recorded.

3.1 Acclaim Motion Capture (AMC) Format

The Acclaim motion format stores joint rotation using euler angles.Each joint in an animation frame will have associated with it 3 euler rotations. For ball and socket joints like the shoulder joint, all 3 rotations (***rx***, ***ry*** and ***rz***) will be used. For hinge joints, like the knee and elbow joints, only one degree of rotation will be used, with the others set to zero. The use of euler rotations does have disadvantages, like the problem of gimbal locks[3]. For animation storage, euler rotations are sufficient as it is possible to generate all possible points on a unit sphere using a set of euler rotations, even though there may not be a one to one mapping. In this research, an articulated skeleton with static bone structure is used for each actor, hence the assumption that a normalized joint rotation will lie on a unit sphere. We are interested in capturing the overall motion of a person and therefore chose to ignore the small joints like the wrists, fingers and toes. An animation frame can then be represented by a vector of euler angles (pose), and animation achieved by sequentially rendering using these pose vectors. A set of poses, as defined in section 1, is then simply just these set of euler vectors.

3.2 DirectX Motion Format

DirectX format stores its joint rotation as a 4 by 4 homogeneous matrix. The first 3 columns and rows represent a simple rotation matrix and can easily be generated from euler rotations by concatenating the ***rx***,***ry*** and ***rz*** rotations in the correct order (Figure 5). The other entries in the homogeneous matrix store the bone's length, width and distance relative to it's parent bone. Due to the fact that we are working with articulated skeletons with static bones, these values can be left unchanged and ignored in the motion capturing process.

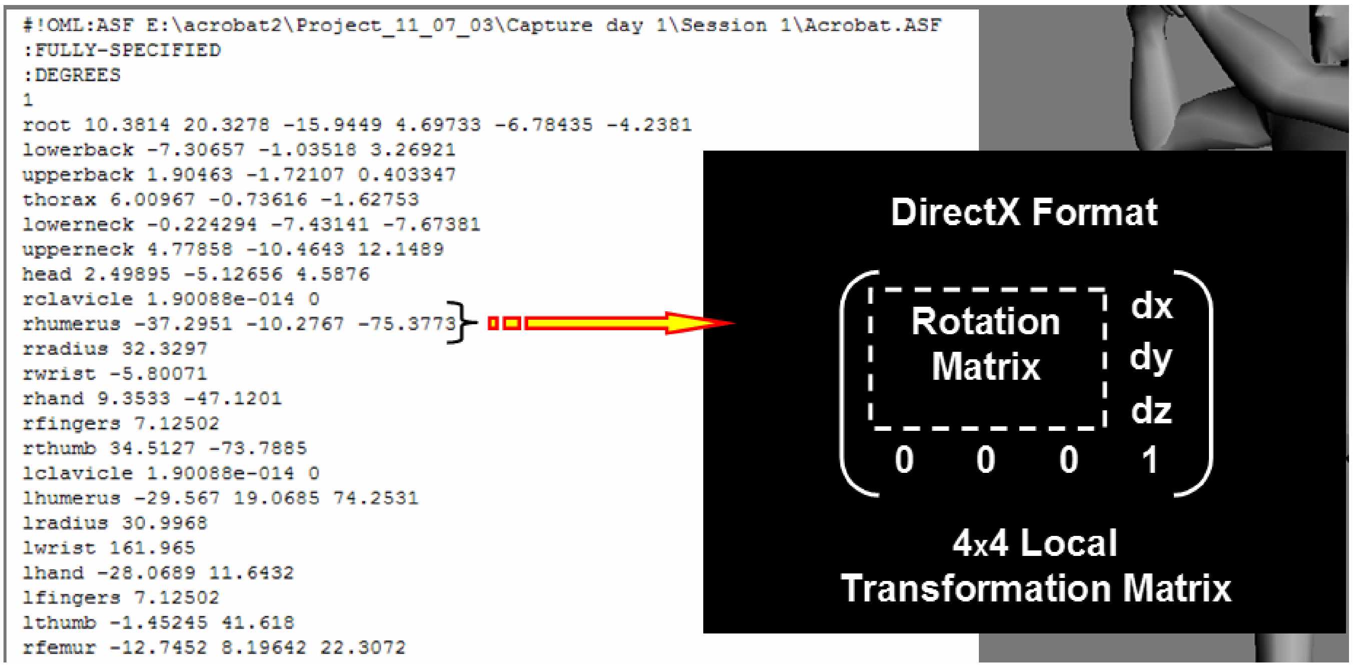
```
#!OML:ASF E:\acrobat2\Project_11_07_03\Capture day 1\Session 1\Acrobat.ASF
:FULLY-SPECIFIED
:DEGREES
1
root 10.3814 20.3278 -15.9449 4.69733 -6.78435 -4.2381
lowerback -7.30657 -1.03518 3.26921
upperback 1.90463 -1.72107 0.403347
thorax 6.00967 -0.73616 -1.62753
lowerneck -0.224294 -7.43141 -7.67381
upperneck 4.77858 -10.4643 12.1489
head 2.49895 -5.12656 4.5876
rclavicle 1.90088e-014 0
rhumerus -37.2951 -10.2767 -75.3773
rradius 32.3297
rwrist -5.80071
rhand 9.3533 -47.1201
rfingers 7.12502
rthumb 34.5127 -73.7885
lclavicle 1.90088e-014 0
lhumerus -29.567 19.0685 74.2531
lradius 30.9968
lwrist 161.965
lhand -28.0689 11.6432
lfingers 7.12502
lthumb -1.45245 41.618
rfemur -12.7452 8.19642 22.3072
```

Figure 5: Images of the 2 different file formats (Acclaim Motion Capture and DirectX) used in this research.

[3]The loss of a degree of rotation when the rotation axis align as a result of the first or second euler rotation.

3.3 Rendering Deformable Mesh in DirectX

DirectX file format also has built in templates for vertex blending of deformable mesh. A mesh is a collection of triangles that join together to create a surface. Each triangle is dependant on it's corresponding vertices, which are shared by other triangles in a mesh. The ability to deform a mesh to display human motion can be viewed differently as the ability to move these vertices in such a way, that when the triangles are rendered using these vertices, a realistic model of the person is formed in virtual reality. DirectX produces deformable mesh animation by using vertex blending [Luna 2004], a process where each vertex in the mesh is assigned a weight factor, which basically stores how much each bones in the skeleton affects each particular vertex. The more weight a vertex has for a particular bone, the more it is affected by that bone's transformation matrix. In DirectX the weights w_i^k for each kth vertex v^k summed over N_b bones in the skeleton must add up to one:

$$\sum_{i=1}^{N_b} w_i^k = 1 \tag{1}$$

The calculation of the position of each vertex v_t^k at frame t in the animation is then simply a weighted linear combination of the vertex v_s^k transformed by the corresponding ***i***th bone's **final combined transformation matrix** F_i. Here v_s^k is the vertex position at the starting Tpose. The final combined transformation matrix F_i is highlighted to distinguish it from the combined transformation matrix C_i, which is calculated as a concatenation of each bones homogeneous (local) transformation matrix L_i as follows,

$$C_i = L_i * C_{pi} \tag{2}$$

where C_{pi} is the combined transformation of the parent of the ***i***th bone.

In summary the local matrix L transform a bone relative to it's parent bone. The combined matrix C transforms a bone relative to the origin of the world space. For the case where we are only concerned with orientation (i.e. pelvis root node fixed at the world origin), the matrix C defines a transformation relative to the root node as shown in Figure 6. The combined final transformation matrix F transforms a bone relative to it's starting Tpose. F_i is calculated as shown below,

$$F_i = C_i * O_i^{-1} \tag{3}$$

where O_i is the offset matrix and transforms the corresponding **i**th bone from the world origin to it's orientation in the Tpose stance. Once all the final combined transformation matrices have been calculated, each vertex v_s^k is mapped to v_t^k at time t according to the equation below,

$$v_t^k = \sum_{i=i}^{N} w_i^k F_i^t v_s^k \tag{4}$$

The only thing left to calculate before we can animate realistic deformable mesh from motion capture data, is the directX weight matrix W_{dx}, which composes of the set of k weight vectors w_i used in Equation 4.

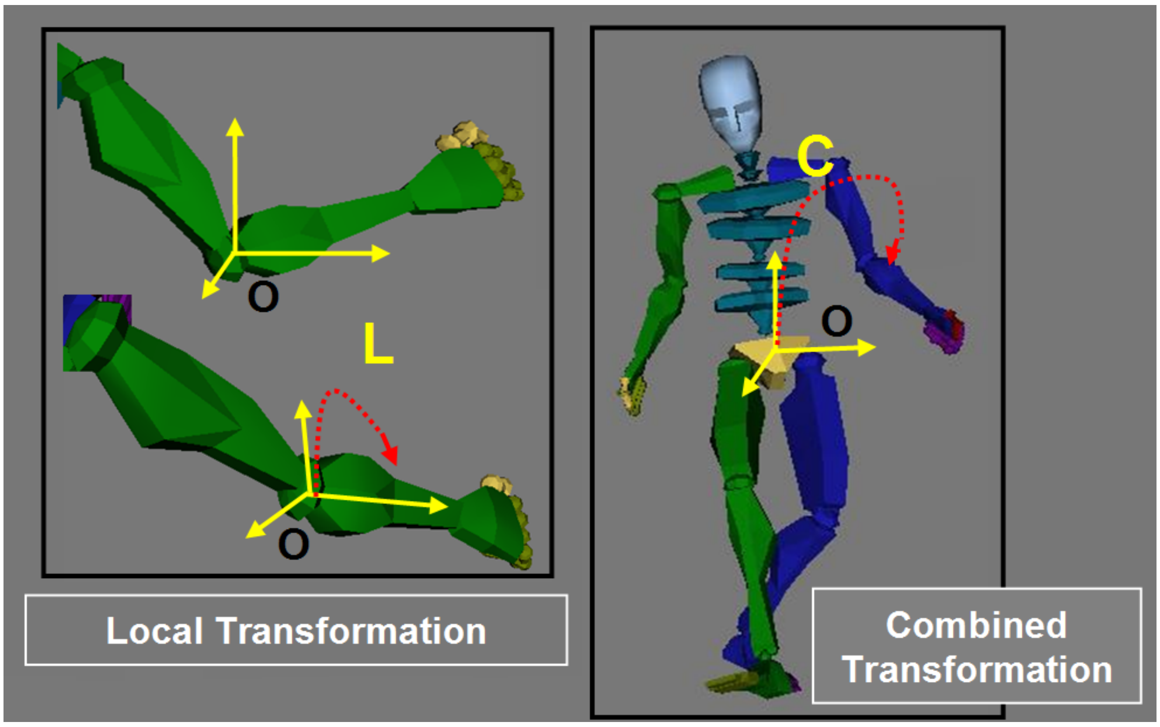

Figure 6: Example to show the mapping as a result of a Local transformation L_i and a Combined transformation matrix C_i with the root node at the origin in world space.

3.4 Creating a Skinned Mesh

The process of transforming a static mesh to a mesh with deformable surface (skin) is sometimes called skinning. This explains the reason why a deformable mesh is sometimes called a skinned mesh. In this research project the skinning of a mesh (which can be worded differently as the problem of determining the weight matrix W_{dx}) is performed in 3D Studio Max, before exporting the resultant skinned mesh to DirectX format for rendering.

In 3DSmax, the weights for each vertex are allocated according to the diagram in Figure 7. Each **i**th bone, is then assigned two closed ended surfaces as shown by the white and black ellipsoids in the image. All the vertices v^k in the mesh that lies inside this white inner closed surface will then be assigned the weight w_i^k of 1, indicating that they are only dependant on the **i**th bone, and no other bones in the skeleton. All the vertices that lies outside the bigger black closed surface, will be assigned a weight w_i^k of zero, indicating no relation to the **i**th bone. The vertices between the two closed ended surfaces will be assigned weights relative to how far they are between the surfaces. Usually a linear or low order polynomial function is used as a fall off function between the surfaces. Therefore in order to transform a static mesh to a skinned mesh, all we need to do is use a skeletal structure that snugly fits in the mesh, then define the bones' closed ended surfaces and the polynomial fall-off functions. The process of skinning a mesh is then simply the calculation of the weight matrix W_{dx} from these surfaces and exporting the static mesh with the weight matrix and hidden skeleton in the same DirectX file. The weight matrix will then ensure that during animation, the vertices in the mesh are smoothly transformed by each bones' final combined transformation matrix F_i and weighted accordingly. Triangular patches are then constructed using these transformed vertices to produce the animation of smooth and realistic deformable mesh.

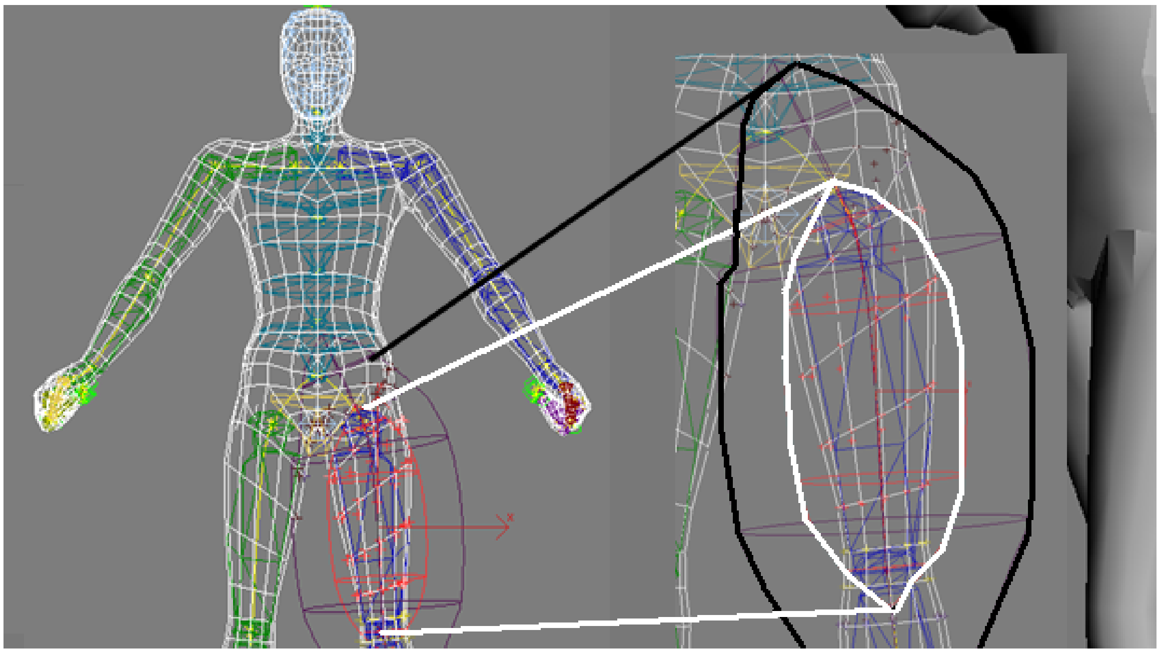

Figure 7: Diagram to show the skinning process in DirectX.

4 Accurate Mesh Generation

By making use of the skinning technique covered in section 3.4, it is now possible to create realistic mesh animation of any static human mesh by fitting in the skeleton and calculating the weight matrix W_{dx}, before exporting both the inner skeleton and mesh for DirectX animation. In this section we move onto how to go about obtaining an accurate mesh of the actor whose motion is being captured. There are various methods that can be used to obtain 3D models ranging from methods using multiple cameras [Hilton et al. 2000],[Hilton et al. 1999] on segmented image data to techniques using Radial Basis Function [Carr et al.] to fit a smooth function to noisy point cloud data for resampling. We plan to design a capturing system that will be able to make use of mesh models generated from camera images. This point coupled with the ability to re-train the mesh in any synthetic camera views adds to the flexibility of the system and enables it to capture motions at any angles. At present, however, the project uses a laser scanner to capture the person's point cloud data and from that, a mesh is generated[4]. Even though the input data and processing of the camera based and laser scanner based systems are different, the resultant output from the methods are the same, in that there are both a collection of vertices defining a surface of connected triangles. Currently, we have an ongoing project which aims to generate an accurate mesh from views in multiple cameras. We plan to eventually merge these two projects and use the camera based mesh to generate training data, hence creating a capturing system which is truly cheap and flexible.

4.1 Laser Scanner Model Capture

In this research project,the accurate human mesh is calculated from the point cloud data obtained from using the Riegl LMS-Z420i Terrestrial Laser Scanner (equipped with a calibrated Nikon D100 6 Mega Pixel digital camera to capture images for model texturing - section 11). The scanner is used to capture point cloud data of the front and back of the actor as shown in Figure 8.

[4]The accurate mesh generated from the laser scanner will be used for training of the capturing system and ground truth for debugging and testing.

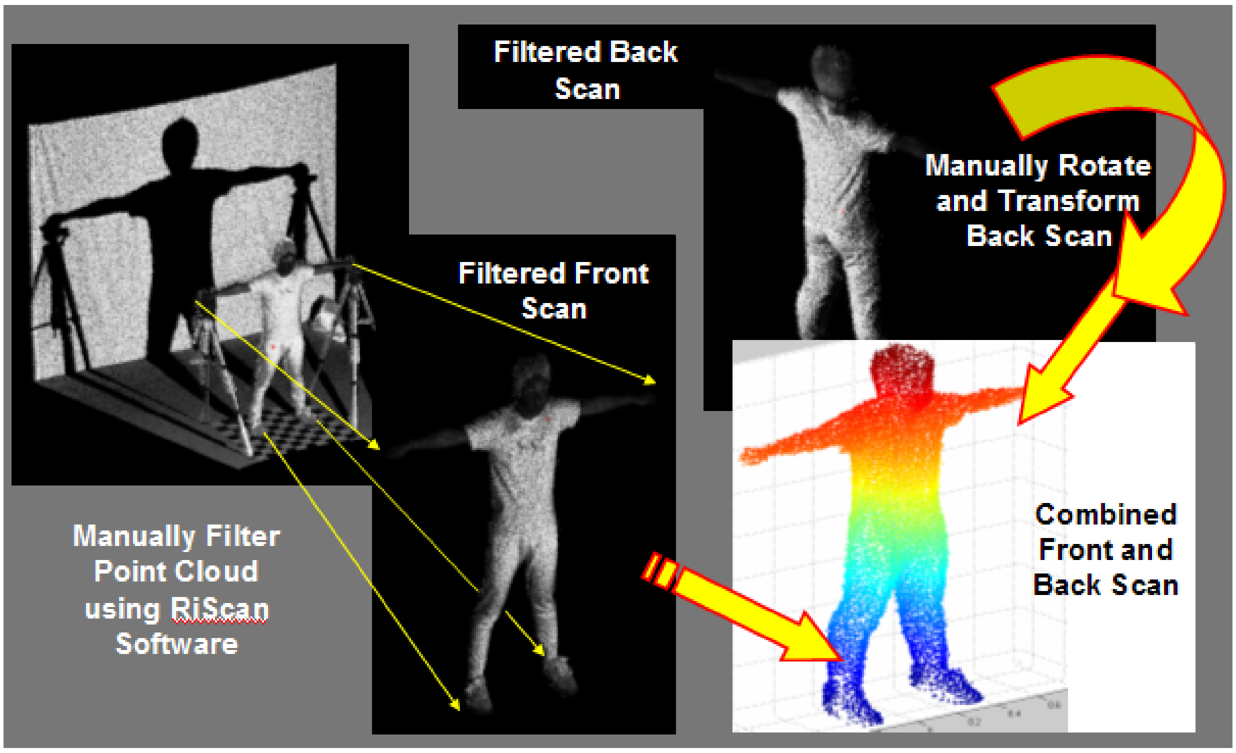
Filtered Back Scan
Manually Rotate and Transform Back Scan
Filtered Front Scan
Manually Filter Point Cloud using RiScan Software
Combined Front and Back Scan

Figure 8: Diagram to show how a person's point cloud data is captures using the Riegl laser scanner.

The actor is requested to have his/her arms stretched horizontally on vertical stands so as to ensure that the mesh model generated from the captured point cloud is that of a Tpose. Another advantage of initially capturing a model in a Tpose is that the resultant static mesh is easier to skin and are less likely to suffer for unrealistic surface deformation. Once the front and back point clouds have been captured, the RiSCAN PRO software is used to eliminate unwanted information like points corresponding to the wall or floor plane. Once all the unwanted information has been eliminated, the back point cloud scan is rotated about the vertical axis and joined to the frontal scan, to create full point clouds data of the actor.

4.2 Radial Basis Function (RBF) for Mesh Smoothing

The next step in the creation of an accurate deformable mesh is to convert the full point cloud data to a realistic and smooth mesh. This is achieved in this project by implicitly modelling a smooth surface with Radial Basis Functions (RBF) generated from point cloud data [Carr et al. 2001]. A radial basis function is a function consisting of a low degree polynomial and a weighted sum of translated symmetric basis functions. An RBF function $S(x)$ can be constructed as follows:

$$S(x) = p(x) + \sum_{i=1}^{N} \gamma_i \psi(x - x_i) \tag{5}$$

where p is a low degree polynomial and $\gamma_i \psi(x - x_i)$, the basis function centered around the point x_i, weighted by the coefficient γ_i. When supplied with the set of input data x_i and the corresponding output values f_i on $S(x)$, the problem of determining the radial function can be restated as a linear system,

$$\begin{pmatrix} A & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \gamma \\ c \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix} \tag{6}$$

where P is a matrix of the monomial basis for the base polynomial p in Equation 5, and c the coefficients. A in this case is the vector of radial basis functions $\psi(x - x_i)$. The top row of the linear system basically ensures the condition stated in equation 5. The bottom constraint is a result of choosing $S(x)$ from $(BL^{(2)})$ Beppo-Levi space, hence implying the orthogonality condition:

$$\sum_{i=1}^{N} \gamma_i = \sum_{i=1}^{N} \gamma_i x_i = \sum_{i=1}^{N} \gamma_i y_i = \sum_{i=1}^{N} \gamma_i z_i = 0 \tag{7}$$

In summary, in order to find a smooth static human mesh from N input points, an RBF surface $S(x)$ needs to be calculated, where

$$S = \{s \in BL^{(2)}(R^3) : s(x_i) = f_i, \ \ i = 1, 2, ..., N\} \tag{8}$$

Once $S(x)$ has been found, the mesh is generated by regular sampling the RBF in a regular grid. The only problem left now is to find f_i for each x_i, such that a smooth RBF $S(x)$ can be generated and resampled. The simplest solution is to assign $f_i = 0$ for all x_i and then resampled the resultant Radial Basis Function at $S(0)$. The problem with this solution is that it leads to the trivial solution of $S(x) = 0$ for all points in R^3. In order to prevent this from happening we need to generate training points x_i where $f_i \neq 0$. This is achieved by generating off surface points from the original point cloud data, and labelling it's corresponding f_i as either more than zero or less than zero for outside and inside point respectively.

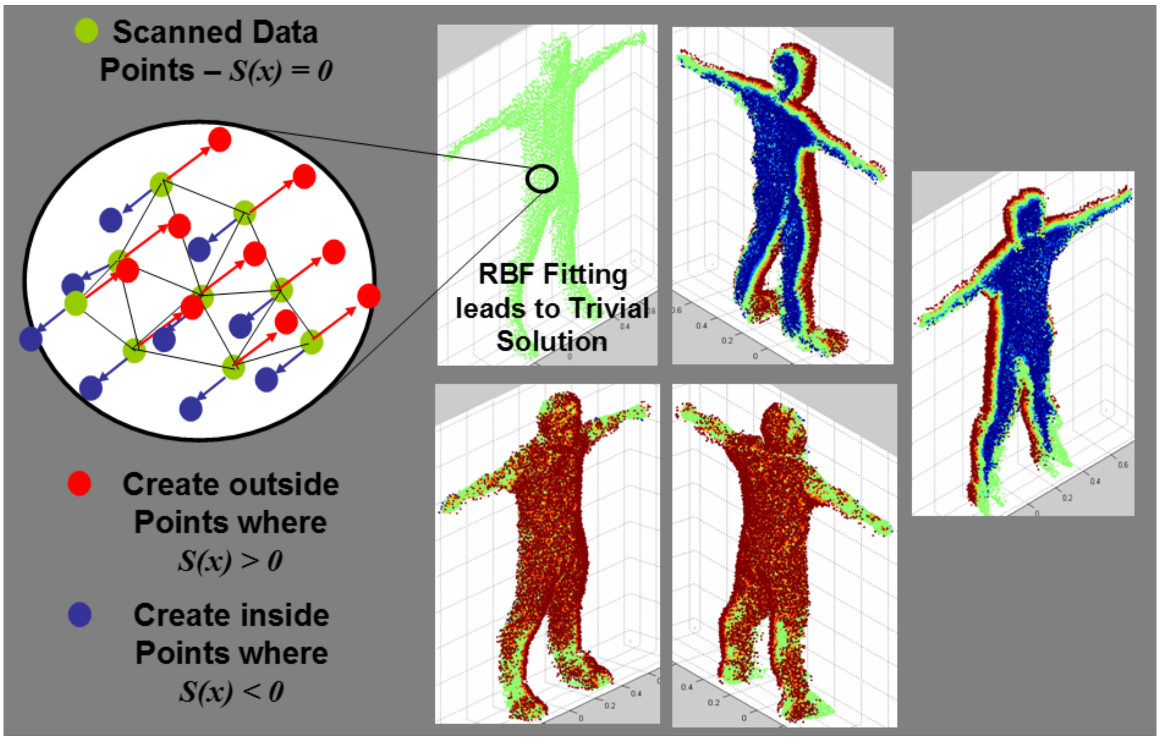
Scanned Data Points – S(x) = 0
RBF Fitting leads to Trivial Solution
Create outside Points where S(x) > 0
Create inside Points where S(x) < 0

Figure 9: Set of images to show how to generate a human mesh using radial basic functions.

As shown above, original capture points from the laser scanner (green circles) are assigned f_i = 0, thereby ensuring that the scanned points lie on or as close as possible to the radial basis function at S(0). From these scanned data points, neighbours are used to generate linear planes, from which off surface outer and inner normals are determined. Outside surface points (red circle) are then generated from these outer normals and assigned (to f_i) the corresponding positive normal distance values from the linear plane. The same procedure is performed on the inner normals to generate inner point clouds (blue circle) except the negative normal distances are now assigned to f_i. The labelled scanned data points, appended with the labelled off surface points are then used to form the linear systems. The mesh is then generated by solving for $S(x)$ and then resampling at $S(0)$.

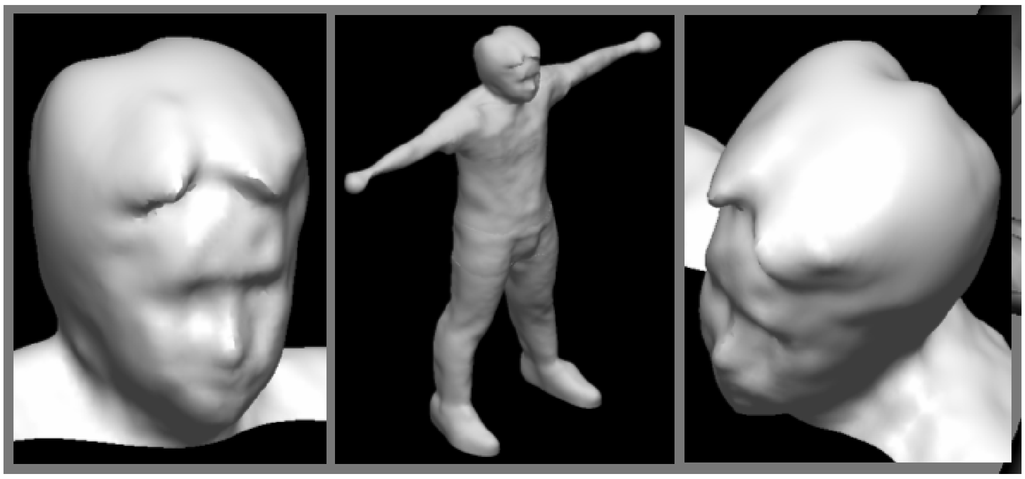
Figure 10: Resultant RBF Mesh generated by using Radial Basis Function $S(x)$ sampled at $S(0)$.

The only problem with solving this system of linear equation is that it is computationally expensive (order N^3) and this cost rapidly increases as N, the number of points (inclusive of both scanned and off surface points) increases. In order to avoid this expensive cost, an evaluation version of Farfield Technology's fastRBF Matlab toolbox was used to generated the mesh. The fastRBF toolbox uses approximation methods to find the radial basis function $S(x)$ such that the maximum fitting and evaluation error,

$$fitting\ \ error_{max} = max|S(x_i) - f_i|, \qquad for\ i = 1,2,......,N \tag{9}$$

$$evaluation\ \ error_{max} = max|S(x_i) - a_i|, \qquad for\ i = 1,2,......,N \tag{10}$$

lie below a predefined fitting and evaluation threshold. In the above equations, a_i is the approximate value of the RBF when it is resampled in order to obtain the static mesh at $S(0)$. The resultant static mesh generated from the Matlab FastRBF toolbox is shown in Figure 10.

4.3 Mesh Texturing

Once the static mesh has been generated from the RBF, it is then possible to texture the mesh using predefined DirectX templates. In DirectX, a two dimensional template space $T_s(x,y)$ is defined for $x \in [0,1]$ and $y \in [0,1]$, as shown in the left image in the diagram below.

In the mesh texturing process, each vertex v^k in the skinned mesh is then assigned a position $T_s(x,y)$ in the texture bitmap. During the rendering process, before each triangle is to be drawn on screen, the

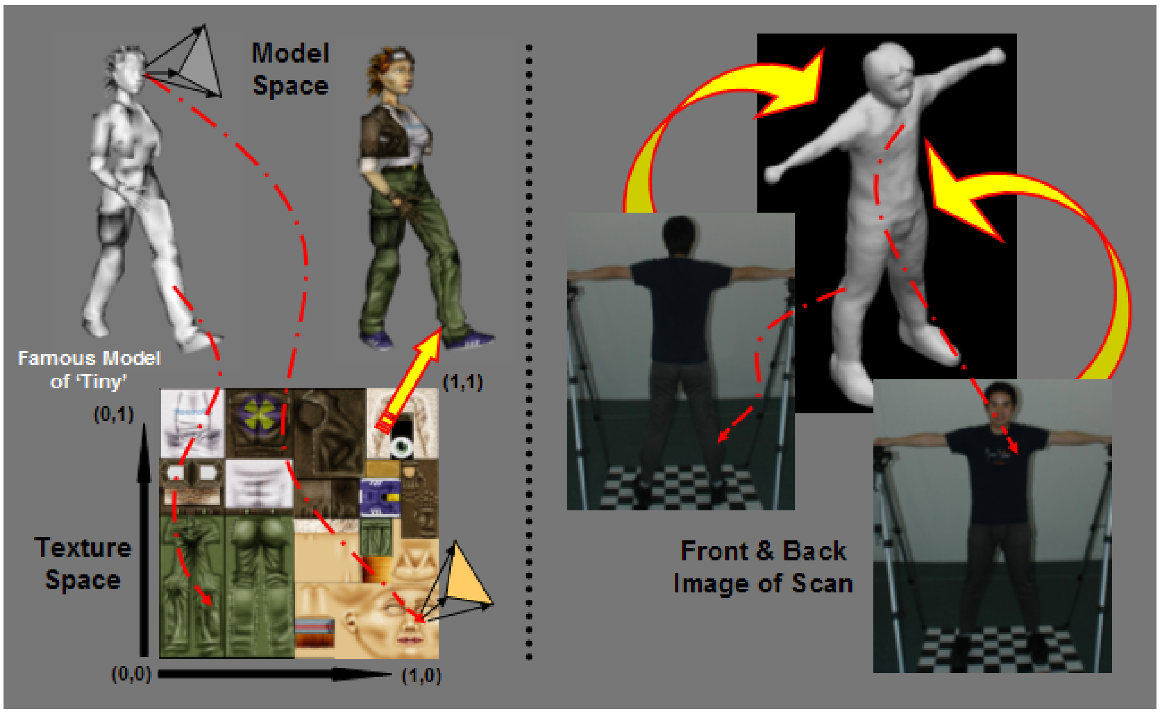

Figure 11: Diagram to show how texturing is performed in DirectX.

corresponding $T_s(x,y)$ locations in the bitmap to the triangle's vertices are found and the enclosed colour values captured and mapped onto the mesh. As mentioned earlier in section 4.1, images of the actor can also be captured with the point cloud data using the Riegl LMS-Z420i Terrestrial Laser Scanner. In future work, we plan to texture map these captured images (shown in the right images of Figure 11) onto the RBF mesh by developing an algorithm to automatically determine the corresponding $T_s(x,y)$ for each vertex v_k in the RBF mesh. Towards the end of this report, we also explain the possible advantages of using accurate textured skinned mesh models to generate training data in multiple cameras, and how it can easily be appended onto the current segment based motion capture technique suggested in section 6. Once an accurate mesh of the actor is generated, we can use the mesh to generate synthetic images of the actor in the cameras. This report now shows how to perform statistical techniques like PCA on the image data before using the processed data to determine a manifold of possible human motions.

5 Dimensionality Reduction and Manifold Learning Techniques

This section covers three dimensionality reduction techniques[5] (Principal Component Analysis (PCA), Kernel PCA and Locally Linear Embedding (LLE)) which can be applied to the data from the camera images to obtain joint angle rotations for motion capture. PCA is a linear reduction method which redefines the original input data as a weighted linear combination of the principal components, which are arranged in order of decreasing variances of the training data. Kernel PCA [Scholkopf and Smola 2002] is a non-linear extension of PCA where input data points[6], X_i are first non-linearly mapped to a higher dimensional space

[5]This section discusses the non-linear manifold learning technique used in the motion capture system in great mathematical details. Readers who are only interested in how manifold learning can be applied to the motion capture system (rather than how to calculate lower dimensional manifolds from high dimensional data) should skip this section.

[6]The capital X_i is used in this case to represent the **centered** data point and to distinguish it from the scanned input points x_i used to generated the RBF mesh.

via a predefined kernel function, before performing standard PCA in this feature space. As discussed earlier in section 2.2, it is usually a complex process to determine the optimal Kernel function (if one exists), and this section therefore shows how it is possible to perform kernel PCA when only the kernel (Gram) matrix is available. In the final part of this section we explore the idea of Kernel PCA de-noising [Mika et al.] via the use of LLE [Saul and Roweis 2003] and show how it is possible to obtain the Kernel Matrix via this non-linear manifold learning technique [Ham et al. 2004].

5.1 Principal Component Analysis

Principal Component Analysis [Jolliffe 1986] is a linear dimensionality reduction where new orthogonal axis are formed from a linear combination of the original axis. These new axis, referred to as principal components are constructed in such a way that the variance of the input data projected onto these principal axis are maximized on the first few components. PCA has been applied to many applications such as data compression or de-noising, where the original data are centered and projected onto the first pre-selected numbers of principal components in order to retain as much of the information as possible, whilst ignoring the projections onto the other lower variance axis, and regarding them as noise. The Principal axis can be obtained from the input data by diagonalizing the covariance matrix,

$$C = \frac{1}{N}\sum_{j=1}^{N} X_j X_j^T \tag{11}$$

and solving the eigenvalue problem

$$\lambda \mathbf{v} = C\mathbf{v}, \qquad \lambda \geq 0 \ \ and \ \ \mathbf{v} \in R^{P_n} \tag{12}$$

where P_n is the original dimension[7] of the data. The eigenvalues λ_i is proportional to the total percentage energy of the data set stored on the corresponding eigenvector $\mathbf{v}_i$ after the projection. Therefore in order to obtain the first k principle components of a data set, the eigenvectors and eigenvalues of the covariance matrix C are first calculated. The eigenvectors are then ordered by decreasing eigenvalues, and the first k eigenvectors are then selected as the first k principal components of the data set.

5.2 Kernel PCA

Kernel Principal Component Analysis [Scholkopf and Smola 2002] is a non-linear extension of PCA, and basically involves mapping the original input data to a higher dimensional Feature space F via a non-linear function Φ. Once in this feature space F, standard linear PCA is then performed, and the non-linear Kernel PCA problem can then be similarly restated as the diagonalization of the mapped covariance matrix in Feature space

$$\overline{C} = \frac{1}{N}\sum_{j=1}^{N} \Phi(X_j)\Phi(X_j)^T \tag{13}$$

[7]The symbol P_n is used here to distinguish it from the N used in section 3 and to emphasize that the original dimension for input data to PCA as used in this research is the number of pixels in an image.

From this matrix the first k kernel principal components are selected from the set of ordered eigenvectors $\mathbf{V}$ of $\overline{C}$, where

$$\lambda\mathbf{V} = \overline{C}\mathbf{V}, \qquad \lambda \geq 0 \ \ and \ \ \mathbf{V} \in R^N \tag{14}$$

The term N in this instance refers to the number of input in the data (training) set[8]. Similarly, we can expressed each kernel principal components $\mathbf{V}^n$ in dual format as a linear combination of the mapped feature points, where α represents the coefficient of the principal components

$$\mathbf{V} = \sum_{i=1}^{N} \alpha_i \Phi(X_i) \tag{15}$$

The de-noising of a new data point X is then achieved via Kernel PCA by mapping the data point to the Feature space via Φ, and then projecting the mapped point onto the first k kernel principal components of V and discarding the rest. This projection can be achieved via a dot product in Feature space,

$$(\mathbf{V}^n \cdot \Phi(X)) = \sum_{i=1}^{M} \alpha_i^n (\Phi(X_i) \cdot \Phi(X)) \tag{16}$$

Due to the fact that Kernel PCA can be performed using the dual format above, the problem of explicitly determining Φ is usually avoided and a kernel function K is usually chosen, where

$$K(X_i, X) = (\Phi(X_i) \cdot \Phi(X)) \tag{17}$$

The selection of this function K such that it is a kernel and represents a dot product in feature space presents yet another new problem, but if we ensure that the Gram matrix $\mathbf{K}$ generated from **all** the input points, where

$$\mathbf{K}_{ij} = (\Phi(X_i) \cdot \Phi(X_j)); \qquad i, j = 1, 2, \ldots.., N \tag{18}$$

is positive semi-definite (has non-negative eigenvalues), then by Mercer's theorem, the function K is a kernel. Substituting this into equation 16, the projecting of a synthetic point in the training set can be projected onto the kernel principal components as follows,

$$(\mathbf{V}^n \cdot \Phi(X_j)) = \sum_{i=1}^{M} \alpha_i^n (\Phi(X_i) \cdot \Phi(X_j)) = \sum_{i=1}^{M} \alpha_i^n \mathbf{K}_{ij} \tag{19}$$

Using the equation above, to find the projection of a point X_j onto the principal component V^n, we need to determine the Gram matrix $\mathbf{K}$ and the coefficients of α. [Scholkopf and Smola 2002] showed that it is possible to calculate α by substituting equation 15 into equation 14, and rearranging to obtain

$$M\lambda\alpha = \mathbf{K}\alpha \tag{20}$$

[8]The use of N here is similar to the use of N in the generation of the radial basis function mesh in section 4.2 in that they both represents the total number of data points.

and showing that α is in fact the eigenvectors of the Gram matrix **K**. Therefore the problem of Kernel PCA can be reduced down to finding a positive semi-definite Gram matrix[9] **K** from the data set such that the variance is maximized over the first few kernel principal components of V. The disadvantage with the kernel projection expression in Equation 19 is that it is defined only for points originally within the set. In de-noising, it is usually the case that kernel PCA will need to be applied to a new point X, **not** originally in the set. Later in section 5.3, we introduce the use of Locally Linear Embedding (LLE) and shows how to use LLE to obtain a positive semi-definite Gram matrix **K** and its relation to Kernel PCA. Later in section **??**, we also show how it is possible to take advantage of this relation between LLE and Kernel PCA, and use LLE to project new point X onto a manifold hence avoiding the problem of mapping new points via a kernel function in Kernel PCA.

5.3 Locally Linear Embedding (LLE)

Locally Linear Embedding [Saul and Roweis 2003] is a non-linear dimensionality reduction method whereby low dimensional data are regenerated from a **well-sampled** high dimensional data, whilst preserving euclidian distance ratio between local neighbours. Basically given a training set of N points in high dimensional space R^D, LLE expresses each high dimensional vector in the training set as a linear combination (dual format) of it's k neighbours. A cost function is then defined that measures the quality of this reconstruction,

$$\varepsilon^r(W) = \sum_i |\overrightarrow{X_i} - \sum_j W_{ij}\overrightarrow{X_j}|^2 \tag{21}$$

The weight W_{ij} summarizes the fraction of the **i**th vector in the training that can be synthesized from the **j**th vector in the same set. If the **j**th vector is not a neighbour of point **i**, then W_{ij} will be set to zero. Another constraint imposed by LLE is that $\sum_j W_{ij} = 1$, which together with equation 21 ensures that every output point in the training set are invariant to rotations, rescalings and translations between its k neighbours. Once the matrix W has been calculated, LLE then regenerate a set of lower dimensional vectors Y in R^d by minimizing the embedding cost function,

$$\varepsilon^e(W) = \sum_i |\overrightarrow{Y_i} - \sum_j W_{ij}\overrightarrow{Y_j}|^2 \tag{22}$$

subject to the constraint $(Y_i Y_i^T)$ = 1,The main difference between this embedded cost function $\varepsilon^e(W)$ and the reconstruction cost function $\varepsilon^r(W)$ is that we are now keeping W_{ij} constant, whilst varying the low dimensional vectors in Y. The real advantage of LLE is the ability to solve this minimization as an eigenvalue problem by finding the eigenvectors of the matrix M, where

$$M = (I - W)(I - W)^T \tag{23}$$

The lower dimensional representation Y can then be obtained by taking the eigenvectors of M with the smallest eigenvalues λ as the rows of Y. A special common eigenvector of M with the eigenvalue λ^M of 0 is that of a vector of a set of ones (or minus ones). This special vector is called the uniform vector **e** and

[9]Note that since we are performing normal linear PCA in feature space and PCA is usually performed on centralized data, the feature space data will therefore need to be centralized as well. In input space this is easily achieved by offsetting each input point by the mean of the data. This centralizing problem is a lot more difficult in Feature space, since an explicit feature space mapping Φ is not usually defined. In the appendix section, [Scholkopf and Smola 2002] showed that $\tilde{\mathbf{K}}_{ij}$ in the Gram matrix generated from centralized mapped feature space data is equal to $(\mathbf{K} - 1_M\mathbf{K} - \mathbf{K}1_M + 1_M\mathbf{K}1_M)_{ij}$, where $(1_M)_{ij}$:= 1/M

can easily be shown that it minimizes the embedded cost function $\varepsilon^e(W)$ by substituting **e** into equation 22

$$\sum_i |\overrightarrow{e_i} - \sum_j W_{ij}\overrightarrow{e_j}|^2 = \sum_i |\overrightarrow{e_i} - \sum_j W_{ij}|^2 = \sum_i |\overrightarrow{e_i} - 1|^2 = 0, \quad for\, \mathbf{e} \in R^N : \mathbf{e} = [1,1,...,1] \tag{24}$$

Due to the fact that **e** is always a solution in LLE, it is usually ignored, and the embedding Y is generated from the eigenvectors with the lowest eigenvalues thereafter. In summary, in order to perform LLE manifold learning, two variables need to be preselected, the numbers of neighbours k to use in LLE and the number of output eigenvectors d that will be used to represent the embedded output data. For each input point X_i in R^D, LLE then selects the nearest k neighbours as determined by the euclidian distance in R^D and construct the weight matrix W and M. The low dimensional embedding Y in R^d is then obtained by finding the first **d** eigenvectors of M with the lowest eigenvalues λ^M not equal to zero[10].

It is also possible to determine the LLE Kernel Matrix $\mathbf{K}^{lle}$ of the training set (as discussed previously in section 5.2 - Kernel PCA). Given the matrix M from LLE, [Ham et al. 2004] showed that the positive definite and centered Gram matrix $\mathbf{K}^{lle}$ can be defined by taking the pseudo inverse of the matrix M,

$$\mathbf{K}^{lle} := M^{\dagger} \tag{25}$$

From this relationship, it is then possible to show that the Kernel PCA projections onto the kernel principal components V^n via the Gram matrix $\mathbf{K}^{lle}$ is exactly the same as the LLE embedding, scaled by the square root of the corresponding eigenvalues of $\mathbf{K}^{lle}$. The squared root scaling is necessary because the embedding generated from LLE will be the normalized eigenvectors α, all scaled to have the squared distance of N. In order to ensure that the total variances of the data set are left unchanged after projection onto the kernel principal components, all normalized projected points (from LLE) need to be scaled by the square root of the variance[11]. The projection via kernel PCA is achieved by ensuring that the kernel principal components V^n of the mapped points $\Phi(X_i)$, are all normalized, leading to

$$(V^n \cdot V^n) = \sum_{i,j=1}^{M} \alpha_i^n \alpha_j^n (\Phi(X_i)\cdot\Phi(X_j)) = \sum_{i,j=1}^{M} \alpha_i^n \alpha_j^n \mathbf{K}_{ij}^{lle} = \alpha^n \cdot \mathbf{K}^{lle}\alpha^n = \lambda_n^{lle}(\alpha^n \cdot \alpha^n) = 1 \tag{26}$$

Therefore, the LLE embedding (also known as the eigenvector of the Gram Matrix) α^n multiplied by $\sqrt{\lambda_n^{lle}}$ gives projection of the input point onto the kernel principal component V^n.

6 Application of Non-Linear Manifold Learning to Motion Capture

This section shows how to integrate the ideas and techniques discussed in the previous three sections (**Motion Capture File Format**, **Accurate Mesh Generation** and **Dimensionality Reduction and Manifold Learning Techniques**) to capture human motions using images from multiple cameras. A brief overview of the markerless motion capturing system is first presented in section 6.1, before moving on to how to learn a non-linear manifold using Kernel PCA via LLE. We then end off by presenting a technique that can be used to project new input images (not originally in the training set) onto the manifold.

[10]In this case the eigenvector **e** with the eigenvalue of zero is ignored in the embedding. In some literature this vector **e** may be included in the output embedding, resulting in a lower dimensional manifold in R^{d+1}. This, however, has no effect on the euclidian distances between all points on the manifold because it has a constant value of 1 in the extra dimension for all points in the set.

[11]The variance of the projected data set with the Gram matrix $\mathbf{K}^{lle}$ is stored in the corresponding eigenvalues λ^{lle} of the Gram matrix

6.1 Motion Capture System Overview

In order for our camera based motion capture system to successfully capture human motion, the system must basically convert multiple images, to a pose vector in R^P, where P represents the dimensions of the combined skeletal joint rotations. We plan to investigate the possibility of treating the problem of multiple camera motion capture as an semi-supervised learning problem. The problem can therefore be restated as follows: Given a high dimensional vector I in R^P of rearranged pixels of concatenated images from multiple views, the system must map the vector to a lower dimensional space Y in R^d, where it will be possible to determine the pose of the person for the set of input images.

We plan to perform the non-linear manifold projection by using Kernel PCA via LLE as explained earlier in the previous section. In order for LLE to work properly, a set of well sampled vectors I^{tr} in R^P and it's corresponding lower dimensional points Y^{tr} in R^d are needed for training. The training set will be generated synthetically with an accurate skinned mesh of the person by computationally running through all the possible euler joint angle E representation of the skeleton in R^E. Virtual cameras views with the same intrinsic and extrinsic parameters as the m cameras to be used in the motion capture are then created in virtual space. For each euler pose vector in R^E, it is then possible to find its corresponding set of images in multiple cameras by taking snapshot in the virtual cameras. Initially we plan to test the concept with segmented training data generated from a gray skinned mesh, but in the future, the method will be extended to colour images where the textured accurate skinned mesh will be needed to generated the training set.

For each euler pose vector E_i in the training set, a set of m synthetic camera images of the actor can be captured. The training system then **crops** each image to fully enclose the outline of the person, before resizing the cropped image to a predefined training dimension. The cropping of the image data before training is a very important step as it creates the robustness of this technique of motion capture, by allowing the person to move around anywhere, as long as the person is fully enclosed within the view of the cameras[12]. From the cropped images, each columns are rearranged to form a single column vector and concatenated to produce a high dimensional vector in R^P. The concatenation of images from multiple synthetic cameras can be used to eliminate the problem of occlusions and to produce distinct set of images for each pose.

From the training set I^{tr}, a lower dimensional manifold Y^{tr} can be obtained via LLE. However due to the extremely high dimension of R^P, it would be inefficient to perform LLE on the rearranged pixel data. Linear principal component analysis is first performed on I^{tr}, to obtain the vector X^{tr}. As the purpose of applying PCA is not to compress the data but to re-represent all the data in lower dimensions, the first D principal components with eigenvalues $\lambda^{pca} > 0$ are retained. Each training vector in the set I^{tr} is then projected onto the linear principal components to generate the LLE training set X^{tr} in R^D. LLE is then performed on this training set X^{tr} to reduce the input vector to it's manifold representation Y^{tr} in R^d. Each manifold representation Y_i in Y^{tr} will also have a matching euler E_i, and it is the significant of this link that the system will take advantage of in order to determine the euler pose of a new point for motion capture.

Given a new set of images not in the training set but from the same set of cameras, the images are first cropped and rearranged to get I in R^P. This vector is then centered and projected onto the first D linear principal components of the training set, to get the vector X in R^D. The vector X is then projected onto the kernel principal components via LLE to get the manifold vector Y. Once on the manifold, the nearest k_m neighbours on the manifold defined by Y^{tr} are found, and Y is re-expressed in dual format as a linear combination of these neighbours. The output pose from the motion capture system can then be generated

[12]One may argue at this stage that this cropping process eliminates the ability to determine the world position of the entire person. This may initially be the case, but once the pose of the person has been found, the tracking system can make use of this information with the centre of the cropped image to find the world position of the person

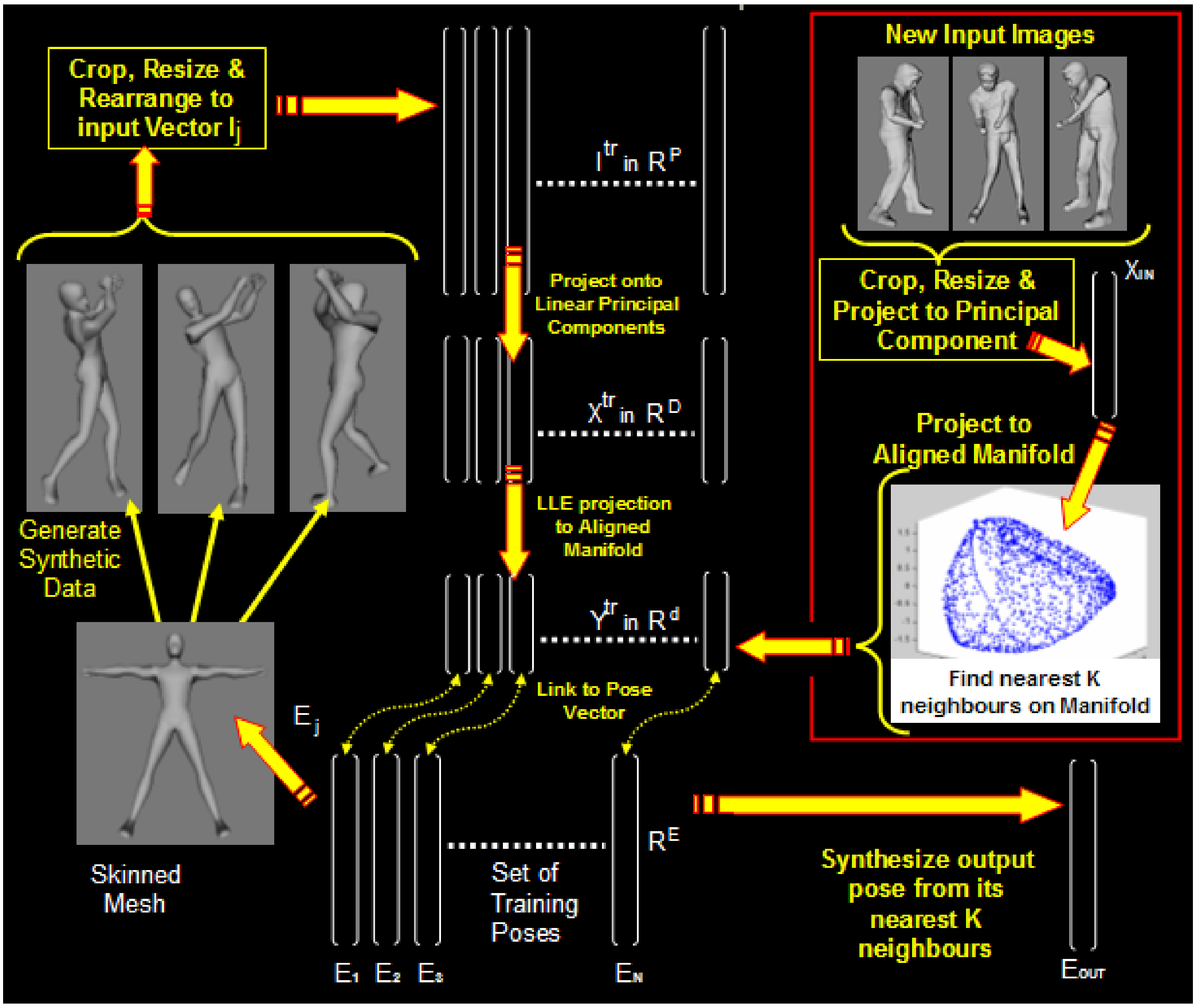
Crop, Resize & Rearrange to input Vector Ij
Generate Synthetic Data
Skinned Mesh
Ej
Itr in RP
Project onto Linear Principal Components
Xtr in RD
LLE projection to Aligned Manifold
Ytr in Rd
Link to Pose Vector
Set of Training Poses
RE
E1 E2 E3 EN
New Input Images
Crop, Resize & Project to Principal Component
XIN
Project to Aligned Manifold
Find nearest K neighbours on Manifold
Synthesize output pose from its nearest K neighbours
EOUT

Figure 12: Diagram to show overview of the Motion Caputring System.

as a linear combination of neighbours' corresponding vector in the set E^{tr}. In order for this dual format regeneration to work, the Y_i on the manifold and E_i of the pose vectors must have a high percentage of common neighbours, that is the manifold Y^{tr} must be aligned to the set of vectors E^{tr}. In the following section, we cover an LLE technique which can be used to align these manifold as well as how to use the same technique to project new point X not in the training set onto the manifold.

6.2 LLE Projection to Common Manifold

The problem of manifold alignment can be reformulated as follows: Given 2 sets of corresponding high dimensional vectors X^{tr} and E^{tr}, how do we project both data set to a common manifold such that each point X_i and E_i share the most numbers of the same neighbours. Provided that this common manifold can be found, given a new point X not in the data set, we can then project it onto the common manifold, find the closest neighbours on this manifold in the training set and generate the output pose using linear distances between neighbours in pose space. A mathematical formulation of the combined matrix Z composing of both X^{tr} and E^{tr} and the new input X is express below,

$$Z = \begin{bmatrix} X^{tr} & X \\ E^{tr} & ? \end{bmatrix} \tag{27}$$

The problem of determining a human pose can then be restated as a learning problem of determining the unknown in the matrix Z. It turns out that this problem can easily be solved via LLE as shown by [Ham et al. 2003]. Provided that there are N training points, we can use LLE independently on the set of vectors in X^{tr} appended with the new input point X, and another on the euler vectors E^{tr},

$$Y^X = LLE([X^{tr}\,X]) \tag{28}$$

$$Y^E = LLE([E^{tr}]) \tag{29}$$

From this we will be able to also obtain the weight matrix for both set, as well as the M matrix[13], M^X (dimension of $N+1$ by $N+1$) and M^E (dimension of N by N). Letting M^X_{nn} represent the first N rows and N columns of matrix M^X and M^X_{ss} the columns and rows in matrix after N, the projection of the two manifold Y^X and Y^E to a common manifold Y can be achieved by combining the M matrices as follows,

$$M^Y = \begin{bmatrix} M^X_{nn} + M^E_{nn} & M^X_{ns} \\ M^X_{sn} & M^X_{ss} \end{bmatrix} \tag{30}$$

The LLE mapping Y of the points $[X^{tr}X]$ to a common manifold can be found by taking the eigenvalues of M^Y. This will produce a low dimensional manifold where the corresponding training points are constrained to be equivalent on the common manifold. The kernel PCA projections onto the kernel principal components can be found by finding the pseudo inverse K^{lle} of M^Y and scaling the LLE mapping Y by the square root of its corresponding eigenvalues of K^{lle}. Furthermore, the closest k neighbours to the new input point X can be found by finding the nearest **k** neighbours to the mapped point on the manifold. The neighbours are then used to generate the output in pose space. This is possible as we have mapped the pixel data to a lower dimensional space which is constrained to be close to the pose space manifold.

An immediate argument against LLE may be that we are generating the output pose from a linear combination of neighbours on the manifold generated by LLE. This combined with the fact that LLE is a neighbouring distance preserving dimensional reduction technique would suggest that LLE is just an extra pointless step, since the same nearest k neighbours in X^{tr} would map to the exactly the same k neighbours in Y. This would then suggest that the LLE manifold learning technique could be completely removed from the system, and the system could regenerate the output pose from a linear combination of its neighbours in linear PCA space. An argument for LLE is that even though it imposes the constraint that each point on the manifold Y can be regenerated as a linear combination of its **k** neighbours on X, it does not necessary mean that the same nearest **k** neighbours would be found on Y as found on X. This is because for any point in the set, LLE can actually project points that were initially not within its set of nearest k neighbours on X to be within its nearest k neighbours on Y. This concept will be clearly illustrated in the results in the following section.

Another main advantage of LLE can be shown by taking a closer look at the generation of the combined vector Z. In equation 27, an input X with unknown mapping is appended to X^{tr}. A similar extension can

[13]The matrix M is obtained as follows (as discussed in 5.3), $M = (I-W)(I-W)^T$

be applied for E^{tr} by adding more synthetic pose vectors E^{syn} which do not have known correspondence in image space. All of these new synthetic points will then be mapped to the common manifold, resulting in more neighbours for the input point to chose from when mapped onto the same manifold. This further supports the argument that a neighbour in linear PCA space does not necessary have to be a neighbour after performing LLE. Since we know how to accurately generate these synthetic pose vectors, we can generate as many as we want. There is obviously an optimal ratio between poses with correspondence and poses without that will give the best animation. The new common manifold will now be based on the new definition of the pose manifold Y^E,

$$Y^E = LLE([E^{tr}\ E^{syn}]) \tag{31}$$

From this, the new matrix M^Y is generated as follows,

$$M^Y = \begin{bmatrix} M^X_{nn} + M^E_{nn} & M^E_{ns} & M^X_{ns} \\ M^E_{sn} & M^E_{ss} & \mathbf{0} \\ M^X_{sn} & \mathbf{0} & M^X_{ss} \end{bmatrix} \tag{32}$$

From this matrix, we perform the same step and obtain the LLE embedding from the eigenvalues of the matrix M^Y. Another advantage of applying this method is that LLE need to only be performed once on the set $[E^{tr}\ E^{syn}]$, because once we have the matrix M^E, it does not need to be recalculated again. In summary, given a new set of input images, it is cropped, resized and rearranged to get the vector I. This vector is then projected using linear PCA to X. The system then preforms LLE on the set $[X^{tr}\ X]$ to get Y^X and more importantly M^X. The matrix M^X is then combined with M^E generated at system initialization to obtain M^Y, representing the common manifold. The first square roots of the first n eigenvalues of the pseudo inverse of M^Y is calculated and then used to rescaled the LLE embedding. The input point is then identified on the manifold[14], and the nearest k neighbours found. The output pose is then generated as a linear combination of these neighbours in pose space E.

7 Motion Capture Tests and Results

In this section, we apply the methods described in the previous section to synthetic binary images. In order to test the accuracy of the system, the output pose calculated by the system is compared with the pose that was used to initially generate the images in virtual space. To initially test the concept, only the left shoulder joint of a generic mesh is used to generate the training images (Figure 13 - left images). Only part of the images within the white borders are passed to the system for processing, where the images within the borders are segmented and cropped to fully enclosed the mesh, and then resized to an image size of 60×60 pixels. For testing a motion capture data is selected and the euler joint angles for the left shoulder joint filtered out and applied to the accurate skinned RBF mesh (Figure 13 - right images). The same process of bordering, segmentation, cropping and resizing are applied to the test images to obtain the test vector I_i for frame $\mathbf{i}$.

The input vector is then mapped to the common manifold (Figure 21 at the end of the report) using the synthetic data from the generic mesh. The output pose is then generated as a linear combination of the $\mathbf{k}$ neighbours of the training data on the manifold. In order to present the result visually for this case where there is just one bone rotating, we substituted the euler representation E with the axis of the arm A which

[14]If exactly the same arrangement as the one in equation 32 is used, then the input point will correspond to the last point in the embedding

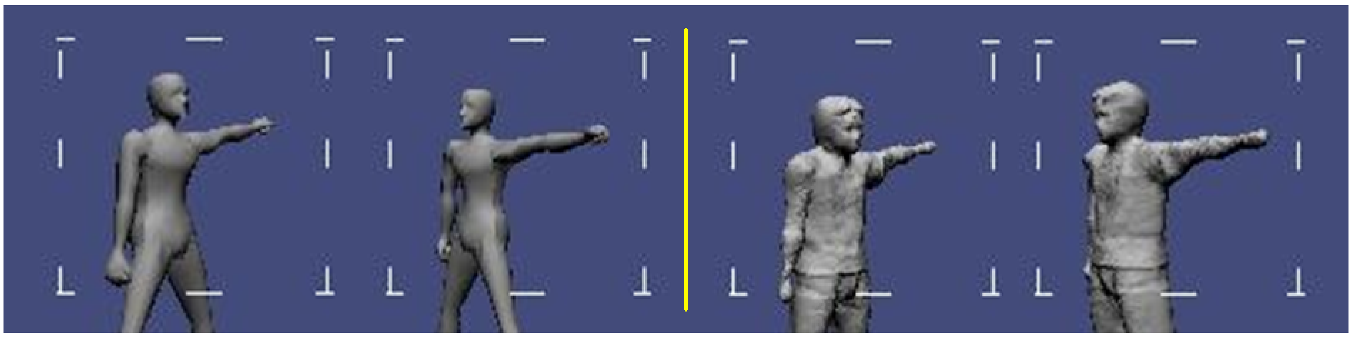
Figure 13: Example of the training images (left) and the test images (right).

is generated by applying the euler rotation to the unit vector $[1,0,0]$[15]. The goal of the system for this experiment is to then find a unit vector $\overline{A}_i^{out}$ from the input camera images which is closest to the vector $\overline{A}_i^{in}$, which is the unit vector $[1,0,0]$, transformed by the euler rotation E_i. This output vector $\overline{A}^{out}$ is then compared with the input pose $\overline{A}^{in}$ (that was used to generate the images) using the following error function,

$$e_i = acos(\overline{A}_i^{out} \cdot \overline{A}_i^{in}) \tag{33}$$

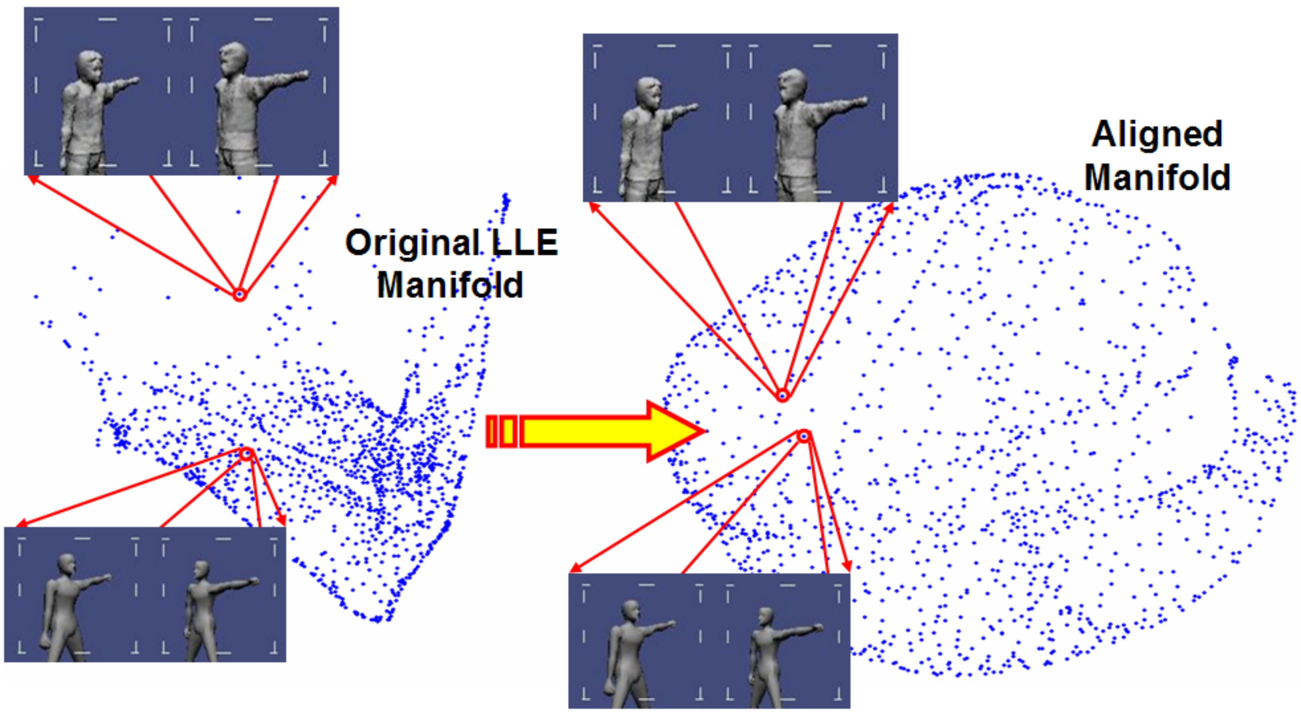

Figure 14: Images of the manifold originally produced by LLE before alignment (left) and after alignment using uniform unit axis correspondence (right).

From the plot images in figure 14, it is clear that mapping to an aligned common manifold results in a much smoother manifold for selecting neighbours.

[15]The vector [1,0,0] is a vector that runs from the centre of the parent node to it's child in each bone of the skeleton. In the case of the left shoulder bone, this vector will always run from the shoulder joint to the elbow.

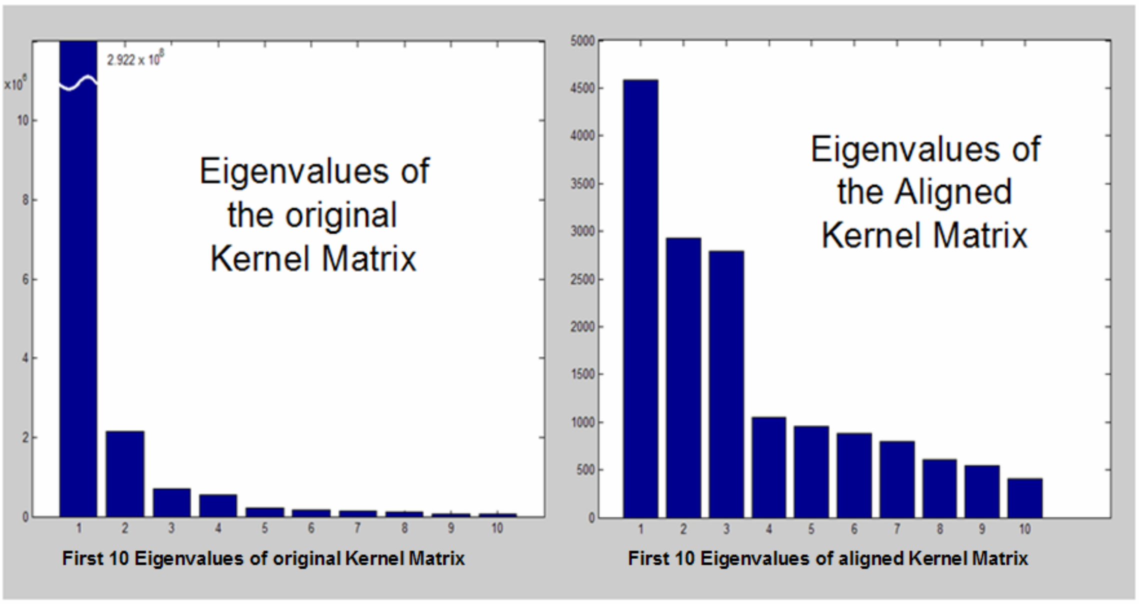

Figure 15: Bar graphs of the eigenvalues of the Kernel Matrix before alignment and after alignment. The ratio of the eigenvalues λ^n to the sum of all the eigenvalues are proportional to the energy stored on the kernel principal component V^n. The numbers of dominant eigenvalues therefore reveal the dimensionality of the manifold, which is correct for the case of the aligned manifold, considering that a shoulder joint has 3 degree of freedom.

It is also possible to determine the dimensionality of the data by looking at the eigenvalues of the aligned manifold, which has three main eigenvalues, hence re-enforcing that most of the variance in the data set is stored in the 3 main kernel principal components. This is not the case with the original LLE manifold, where the eigenvalues show that most of the information is stored in the first principal components.

An interesting observation was that the system was getting similar errors e_i when completely ignoring LLE and interpolation using X^{tr} in the first 600 principal components. This result can easily be explained by examining the eigenvalues plot of the aligned LLE manifold (Figure 15 - right) and the eigenvalues of the linear principal components. In the calculation of the neighbours in both the PCA and the aligned LLE space, the euclidian distances were used, and since both space have most of there variances stored in the first three components as shown in the images, the system would be getting similar results in both spaces, even if it were to calculate euclidian distances using all the components in both space.

It is expected that as the system is extended to include more joints, the variances of the data will spread out over more different **linear** principal components, hence reducing the ability to interpolate using neighbours in the linear PCA space. However with LLE, the system will be able to reduce the dimension to close to the dimension of the joint data, hence preventing it from breaking down as with the case of linear PCA.

There are also 3 main variables that needs to be selected when using LLE for manifold learning, namely the number of neighbours k^{lle} used for LLE manifold construction, the ratio[16] R of the number of points in correspondence and euler pose data E^{syn} without any correspondence, and finally the number of neighbours k^{gen} that will be used to make up the output pose vector $\overline{A}_i^{out}$. From the error plot (figure 17), we can show that it is possible to get similar errors for using neighbours generation in linear PCA space and LLE space. Furthermore, from the rightmost surface plot of LLE error minus PCA error (figure 17), we show that LLE performs better (negative region of the surface plot) using k^{gen} of between 3 to 6 neighbours for regeneration and R between 0.3 to 0.5, further re-enforcing the ability to use less numbers of training

[16]Ratio of points in correspondence to total points including E^{syn}. A ratio $R = 1$ implies that all training points in the set have correspondences.

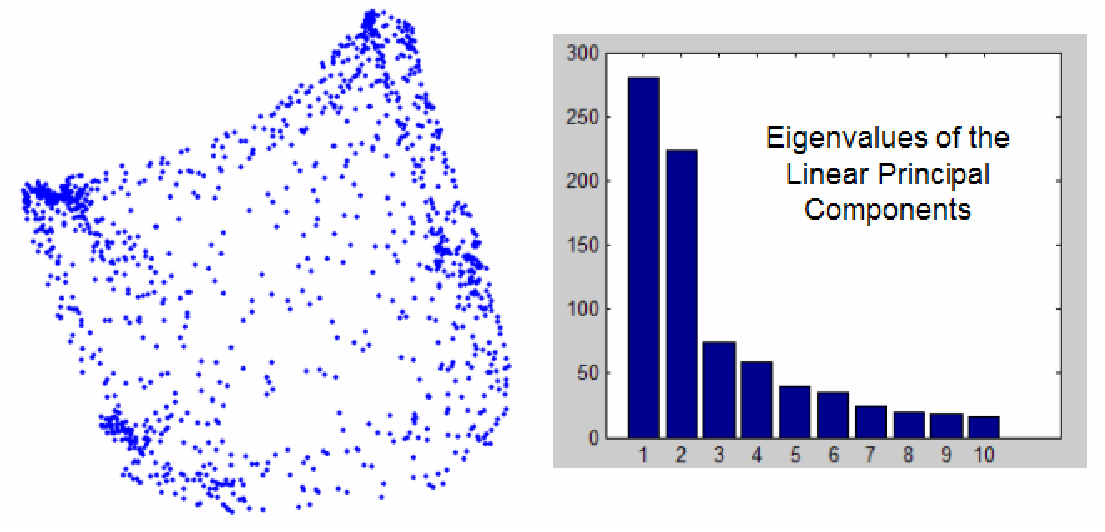

Figure 16: Scatter plot to show the distribution of the training data projected onto the first three linear principal components, and their corresponding eigenvalues.

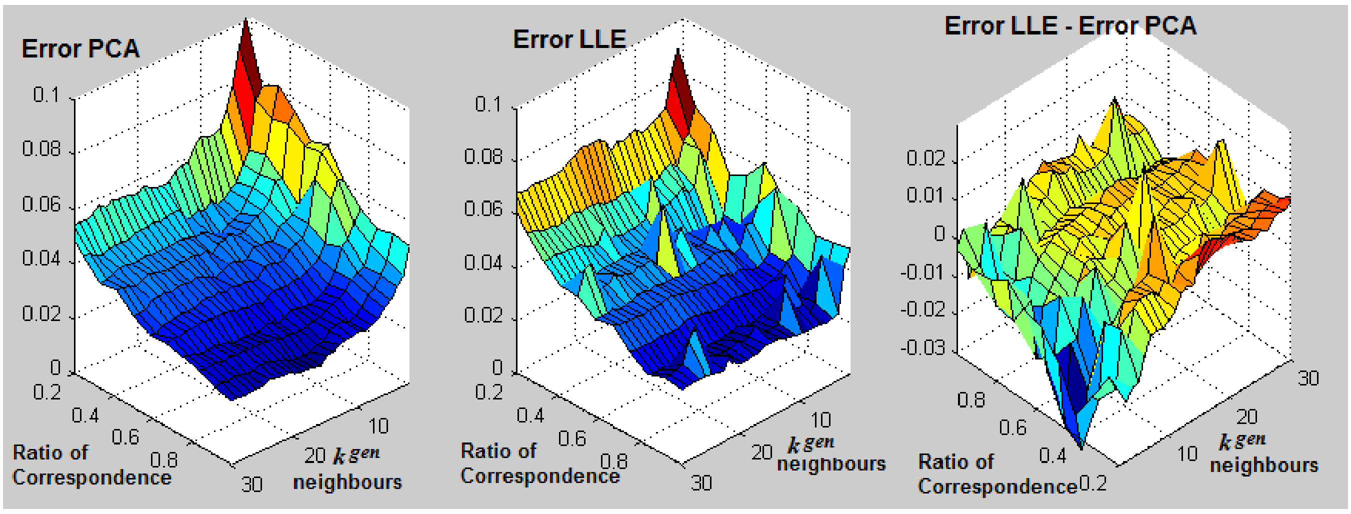

Figure 17: Average error plots of 52 test pose using different values for the parameters R and k^{gen}, with k^{lle} set to 15 neighbours.

points for regenerating motion on the aligned LLE manifold. We plan to further investigate and learn the optimal values of these parameters in future experiments with more skeletal joints.

Another way to visually see how well a manifold is for reconstruction of a new point is by displaying the set of the closest neighbours in pose space. In this case where only the left shoulder is rotated, the system can instead display the set of neighbours closest to the input point on a 3D sphere, as we are looking at single joint rotation. Suppose a trained system is given a pose E, we can map the vector $[1,0,0]$ to axis A using E, giving a vector which lies on a unit sphere. Similarly we can use E^{trn} to map the unit vector to a set of training points on the sphere. The same pose E could the be applied to the skinned mesh and the rearranged cropped vector I generated, before being mapped via PCA to X and then via LLE to the aligned manifold Y. The ideal manifold would then be one where the nearest neighbours to Y on its manifold are the same nearest neighbours to A on the sphere (See figure 18).

Another point to take into consideration is that the only difference between the LLE manifold and kernel principal components projections is the scale factor of the square root of the eigenvalues. For a kernel

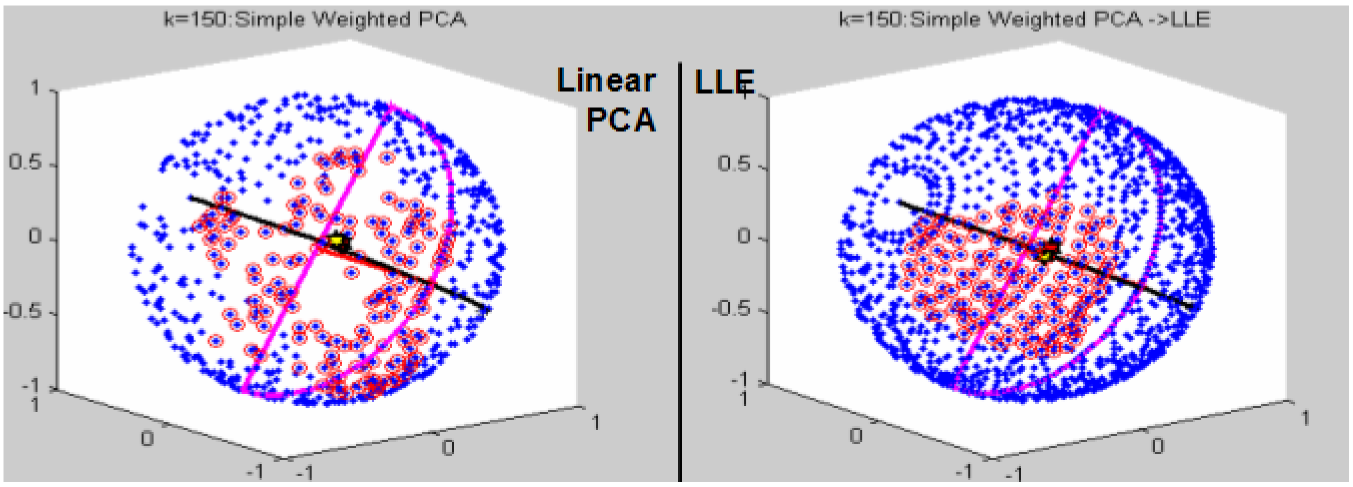

Figure 18: Spherical plots of the first 150 neighbours to a test point in the training set using euclidian distances in PCA and LLE. The stars shows the actual position of the elbow joint. Ideally we want to find a manifold where given a new input point, we can use it to find nearest neighbours on the manifold, which would correspond to nearest neighbours in the output pose space. The red circles in the images mark the nearest 150 neighbours found by using euclidian distances in the PCA and LLE space respectively. Note how the neighbors found in LLE space map to closer neighbours in output axis space.

matrix where the set of biggest eigenvalues are almost equal, the use of euclidian distances search will give the same neighbours for both LLE and KPCA calculations, hence eliminating the step of finding λ^{lle} and scaling by its square root.

8 Discussions and Future Directions

This report has so far presented ideas for a complete camera based system for human motion capture. The real advantage of the techniques presented are it's flexibility and lack of constraints in terms of movement of cameras and system initialization. Taking into account that it is possible to have constant preset intrinsic camera calibrations, all that is needed every time the cameras are moved around is to update the extrinsic parameters and generate training data with the mesh and re-capture the images. It is also extremely easy to increase the capturing accuracy by adding more cameras to the motion capture system, the only changes that will need to be made is that the system would need to be trained with the extra images and this can be implemented automatically without user intervention. One extremely exciting idea that will need further investigation is the ability for the system to capture human motion using training data from a generic mesh (Figure 21), instead of using an accurate mesh obtained from the laser scanner (results shown in figure 19). This has the potential of an extremely portable and inexpensive motion capturing system, especially in areas of computer games and human computer interaction. Imagine being able to buy a computer game, which comes with 2-3 cameras, and all that needs to be fed into the system for training are the extrinsic parameters for the camera. Provided that good quality segmentation can be achieved, then the images can be mapped to the manifold and the poses found. The quality of the motion captured generated will obviously not be as accurate as the ones found by a marker based system, but the system makes up for it in terms of lower cost, portability and flexibility.

Future directions for the project includes extending the training and test motions to include all the joints on upper part of the body, and testing with real camera images captured from firewire cameras. The possibilities for a flexible human interface system using low cost cameras are endless. For example the

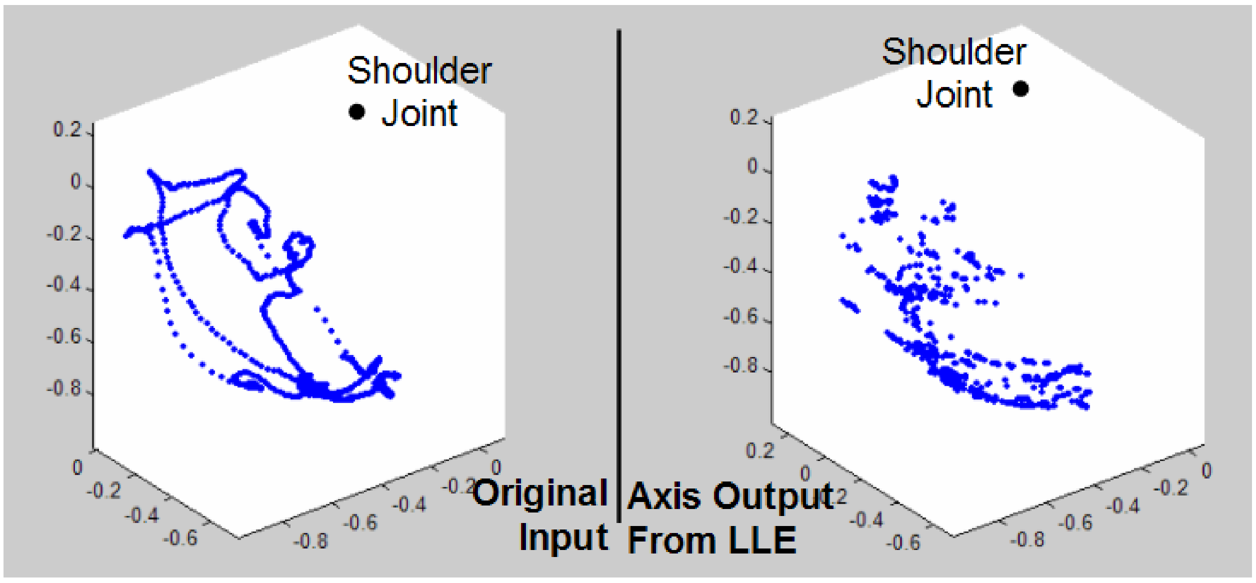

Figure 19: Axis plot (blue dots) of the transformed unit vector $[1, 0, 0]$ of the arm motions for the original motion capture data and the axis captured using LLE manifold projection.

data can be used to drive the mouse or control movements in a computer game, hence give more realism to the experience. Currently the system uses segmented data for training and testing, another improvement that we plan to investigate is to extend the system to work with colour image data. In this case, instead of simply performing segmentation, cropping and resizing in that order, the system will add in an extra step of filling in the foreground pixels with it's original colours captured in the images. This step will follow immediately from the segmentation step, before moving on to cropping and resizing. The system with then be trained with a **textured** skinned mesh rather than using a gray mesh. This should further increase the accuracy of the system, without adding further processing time, as the input dimensions would remain exactly the same as the case of using binary segmented images.

Another possible improvement for the system is its speed. Currently when a new image is received, the system needs to solve for the eigenvectors of an $N+1$ by $N+1$ matrix at each frame for N training points. Instead of recalculating the eigenvalues every frame, we plan to investigate techniques for out of sample extension of the kernel matrix like [Bengio et al. 2004], where new points can be projected onto the manifold and the Kernel matrix extended without fully recalculating for new eigenvectors. Another potential limitation of the system is that temporal information is not used at all in the capturing process. A closer look at resultant axis in figure 19 will show that the output axis will not produce a smooth animation when the poses are concatenated to each other in virtual space. A possible way to fix this is to apply Kalman filters to the joint angles in axis space to introduce temporal constraints. Another interesting area to investigate is the performance of **linear** kalman filtering in feature space, as the main point of kernel principal components analysis is to perform linear PCA on mapped points in close to linear feature space. The same argument should therefore be equally valid for linear Kalman predictions and corrections on data projected onto the kernel principal components in linear feature space.

8.1 Time Line of Thesis

The time line on the following page summarizes the future directions of the project. I plan to spend the next 3 months to extend the motion capture system to enable it to capture the upper part of the human body. The system can then be used to capture real human motion in our Digital Perception Lab. In the beginning of 2006, I plan to investigate the possibility of applying Kalman Filtering to the output motion capture data in order the smooth out the stored animation.

From February 2006, I will focus on implementing a real time motion capture system. For the latter part of 2006, I plan to investigate the possibility of using a generic mesh to generate synthetic training data, before integrating everything together to make a complete motion capture system. The final thesis write-up have been planned for 2007. At the end of the various sections (as shown in the diagram), I will be writing up technical reports and will also be planning to submit papers based on the same topics to international conferences in the Computer Vision and Graphics Community.

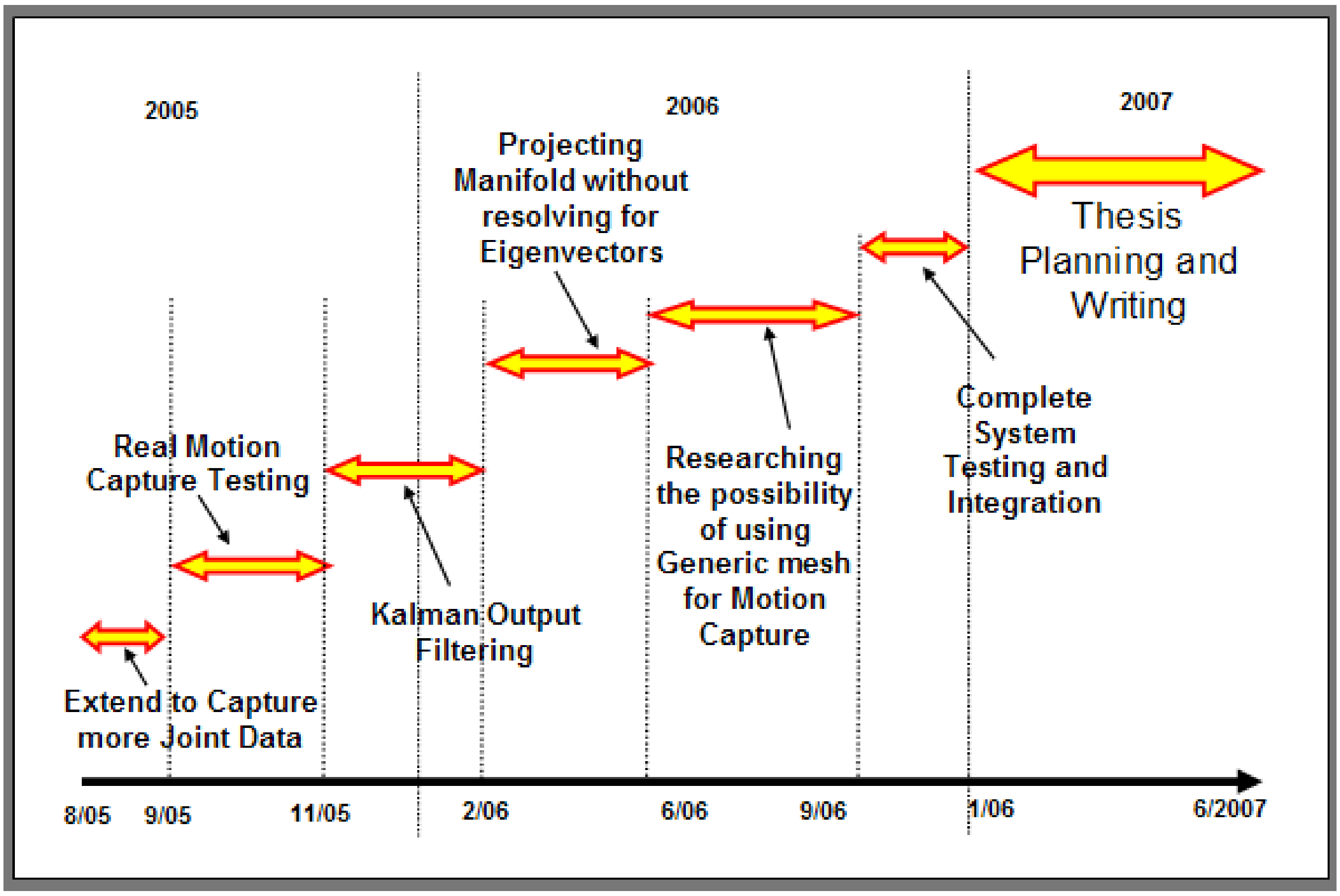
2005
2006
2007
Projecting Manifold without resolving for Eigenvectors
Thesis Planning and Writing
Real Motion Capture Testing
Researching the possibility of using Generic mesh for Motion Capture
Complete System Testing and Integration
Kalman Output Filtering
Extend to Capture more Joint Data
8/05
9/05
11/05
2/06
6/06
9/06
1/06
6/2007

Figure 20: Time Line for future directions.

References

ALEXA, M., AND MÜLLER, W. 2000. Representing animations by principal components. In *EUROGRAPHICS 2000/M. Gross and F.R.A. Hopgood (Guest Editors)*, vol. 19.

BENGIO, Y., PAIEMENT, J., VINCENT, P., DELALLEAU, O., ROUX, N. L., AND OUIMET, M. 2004. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. *Advances in Neural Information Processing Systems 16*.

BOWDEN, R., MITCHELL, T., AND SARHADI, M. 2000. Non-linear statistical models for the 3d reconstruction of human pose and motion from monocular image sequences. *Image and Vision Computing 18*, 729–737.

CARR, J. C., BEATSON, R. K., MCCALLUM, B. C., FRIGHT, W. R., MCLENNAN, T. J., AND MITCHELL, T. J. Smooth surface reconstruction from noisy range data.

CARR, C., BEATSON, R. K., CHERRIE, J., MITCHELL, T. J., FRIGHT, W. R., MCCALLUM, B. C., AND EVANS, T. R. 2001. Reconstruction and representation of 3d objects with radial basis functions. *ACM SIGGRAPH 2001, Los Angeles, CA*, 67–76.

CRISTIANINI, N., AND SHAWE-TAYLOR, J. 2000. *An Introduction to Support Vector Machines*. Cambridge.

DAVIS, J., AGRAWALA, M., CHUANG, E., POPOVIĆ;, Z., AND SALESIN, D. 2003. A sketching interface for articulated figure animation. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation*, Eurographics Association, 320–328.

FRALEY, C., AND RAFTERY, A. E. How many clusters? which clustering methods? answers via model-based cluster analysis. Technical Report 329, Department of Statistics, University of Washington, Box 354322. Funded by the office of Naval Research.

HAM, J. H., LEE, D. D., AND SAUL, L. K. 2003. Learning high dimensional correspondences from low dimensional manifolds. *Proceedings of the ICML 2003 Workshop on The Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, 34–41.

HAM, J., LEE, D. D., MIKA, S., AND SCHOLKOPF, B. 2004. A kernel view of the dimensionality reduction of manifolds. *Proceeedings of the 21st International Conference on Machine Learning, Banff, Canada*.

HILTON, A., BERESFORD, D., GENTILS, T., SMITH, R., AND SUN, W. 1999. Virtual people: Capturing human models to populate virtual worlds. *In IEEE International Conference on Computer Animation,*, 174–185.

HILTON, A., BERESFORD, D., GENTILS, T., SMITH, R., SUN, W., AND ILLINGWORTH, J. 2000. Whole-body modelling of people from multi-view images to populate virtual worlds. *Visual Computer: International Journal of Computer Graphics 16(7)*, 411–436.

JOLLIFFE, I. T. 1986. *Principal Component Analysis*. New York: Springer Verlag.

KEOGH, E., PALPANAS, T., ZORDAN, V. B., GUNOPULOS, D., AND CARDLE, M. Indexing large human-motion databases. *Proceeding of the 30th VLDB Conference, Toronto, Canada, 2004*.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 473–482.

LEE, J., CHAI, J., REITSMA, P. S. A., HODGINS, J. K., AND POLLARD, N. S. 2002. Interactive control of avatars animated with human motion data. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 491–500.

LI, Y., WANG, T., AND SHUM, H.-Y. 2002. Motion texture: a two-level statistical model for character motion synthesis. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 465–472.

LUNA, F. 2004. Skinned mesh character animation with direct3d 9.0c. Tech. rep., moon-labs, www.moon-labs.com, September 2004.

MIKA, S., SCHOLKOPF, B., SMOLA, A., MULLER, K.-R., SCHOLZ, M., AND RATSCH, G. Kernel pca and de-noising in feature space. *GMD FIRST, Rudower Chaussee5, 12489 Berlin, Germany*.

MIKIĆ, I., TRIVEDI, M. M., HUNTER, E., AND COSMAN, P. C. 2002. Human body model acquisition and motion capture using voxel data. In *AMDO '02: Proceedings of the Second International Workshop on Articulated Motion and Deformable Objects*, Springer-Verlag, 104–118.

MOESLUND, T. B., AND GRANUM, E. 2003. Modelling and estimating the pose of a human arm. *Mach. Vision Appl. 14*, 4, 237–247.

PULLEN, K., AND BREGLER, C. 2002. Motion capture assisted animation: texturing and synthesis. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 501–508.

SAFONOVA, A., HODGINS, J. K., AND POLLARD, N. S. 2004. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Trans. Graph. 23*, 3, 514–521.

SAUL, L. K., AND ROWEIS, S. T. 2003. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research 4*, 119–155.

SCHOLKOPF, B., AND SMOLA, A. 2002. *Chapter14 – Learning with Kernels*. The MIT Press.

TANGKUAMPIEN, T. 2003. Multiple camera tracking: A virtual reality approach. Undergraduate thesis, Digital Image Processing Lab, Department of Electrical Engineering, University of Cape Town, November 2003.

TANGKUAMPIEN, T. 2005. Motion capture using principal component analysis. Technical report, Institute of Vision Systems Engineering, Monash University, Clayton, Melbourne, January 2005.

THEOBALT, C., MAGNOR, M., SCHLER, P., AND SEIDEL, H.-P. 2002. Combining 2d feature tracking and volume reconstruction for online video-based human motion capture. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, 96.

UNUMA, M., ANJYO, K., AND TAKEUCHI, R. 1995. Fourier principles for emotion-based human figure animation. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM Press, 91–96.

VAPNIK, V. N. 1995. *The Nature of Statistical Learning Theory*. Springer.

WEINBERGER, K. Q., AND SAUL, L. K. 2004. Unsupervised learning of image manifolds by semidefinite programming. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition - CVPR 2*, 988–995.

WEINBERGER, K. Q., PACKER, B. D., AND SAUL, L. K. 2005. Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*.

WELSH, G., AND BISHOP, G. 2001. An introduction to the kalman filter, siggraph 2001. Article, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175.

WITKIN, A., AND KASS, M. 1988. Spacetime constraints. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, ACM Press, 159–168.

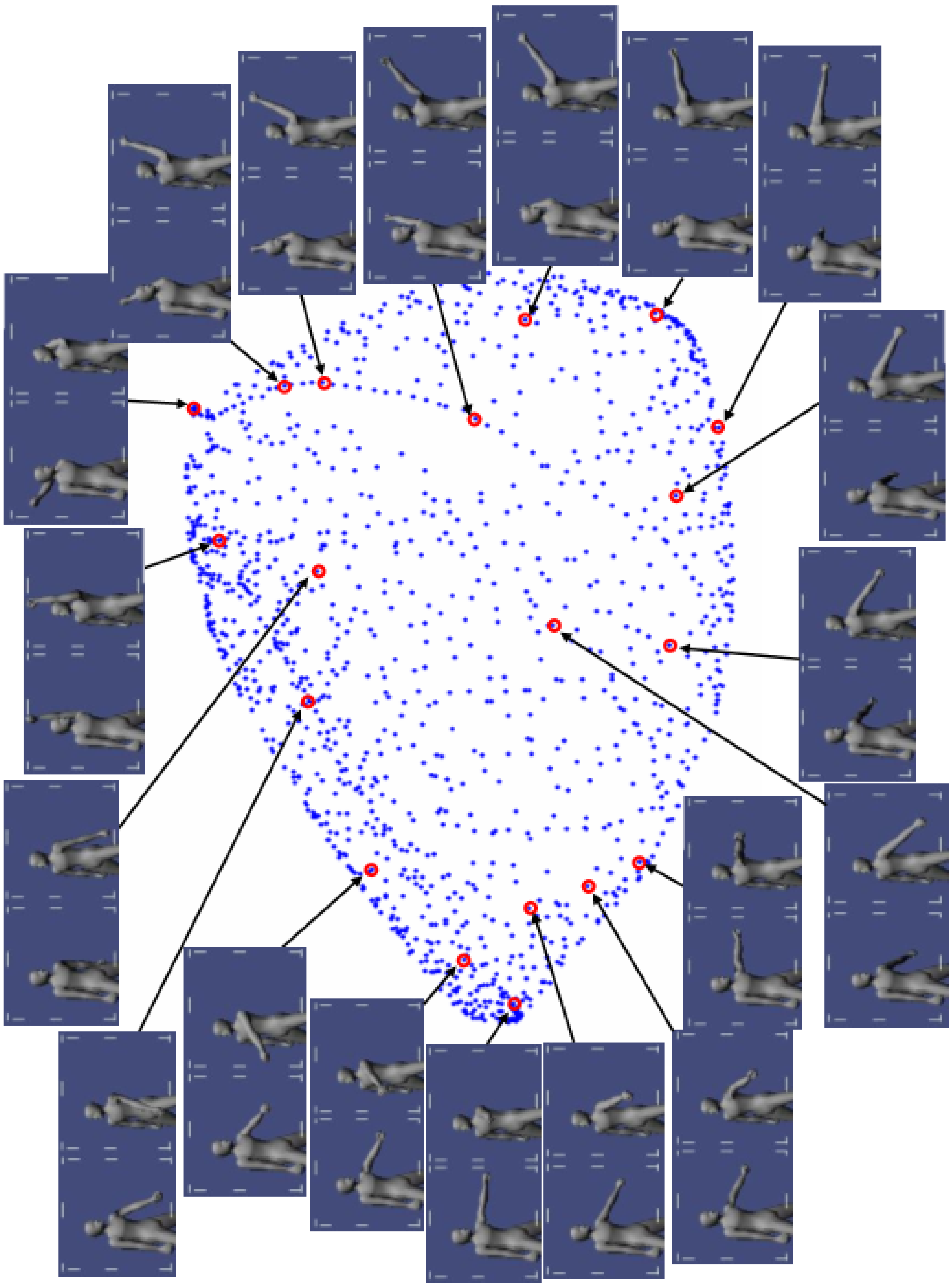
Figure 21: Plots of the LLE aligned manifold and its corresponding images for some of the points.