SYNTHESIZING REALISTIC ANIMATED HUMAN MOTION USING MULTIPLE NATURAL SPACES

By

Reza Ferrydiansyah

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Computer Science

2011

ABSTRACT

SYNTHESIZING REALISTIC ANIMATED HUMAN MOTION
USING MULTIPLE NATURAL SPACES

By

Reza Ferrydiansyah

When animating virtual humans, it is important that the movements created are realistic as well as that they meet various constraints. One way to create motion, given a starting pose, is to first find an ending pose that meets the specified constraints. Then a motion that translates from the starting space to the ending space is computed. Traditional inverse kinematics method are able to find poses that meets constraints, however these poses are not always natural. Linear interpolation between a starting pose and ending pose can be used to create motion. Once again however, the interpolation method does not always create motion that is natural.

This thesis proposes the creation of a *natural space*. The natural space is a hyper-dimensional space in which every point in this space describes a natural pose. Motion can be created by traversing over the points in this space. The natural space is created by reducing the dimensionality of motion capture data using Principal Component Analysis (PCA). Points in the reduced space retain the characteristic of the original data. Multiple natural spaces are created on different segment of the human skeleton.

This thesis describes a method to generate new constrained natural poses that are natural. The poses synthesized are more natural than traditional inverse kinematics, and single space PCA. Motion is created through a space consisting of pose configurations and angular speed. A method to generate realistic looking motion based on this space is presented in this thesis.

Keywords: Principal Component Analysis, Inverse Kinematics, Computer Graphics, Human Animation, Naturalness, Skeletal Model

Dedicated to my beautiful wife Andriana and my energetic son Rivian

Acknowledgements

It is finished.

All praises to Allah, God of the Universe, through his blessings, I was able to finish this dissertation and obtain my PhD at this fine school.

Throughout this long journey I have encountered so many people who have made this work possible. First and most importantly, my wife and soul mate, Andriana, who is my inspiration and who has kept me focused on finishing this work. My parents have always given me support and encouragement, even if they do not understand anything about computers, and without them I would not be where I am today.

Of course, this work would be impossible to finish without my advisor Dr Charles Owen, who taught me what a PhD dissertation is and isn't. Dr Owen's advice was priceless through the whole process of finding a topic, doing the experimentation, and writing the thesis. I have also received generous guidance from my committee. Dr Yiying Tong helped me find the 'Aha' mathematical formula moment. Dr Frank Biocca and Dr Joyce Chai's gave me a different set of very interesting viewpoints that made for some very interesting ideas. I would also like to thak the CSE 101 instructors: Dr Jo Smith, Kevin Ohl, Helen Keefe, Judy Eberlein, Erik Eid, and Vaughn Anderson of whose course I was a big part of during my PhD.

Throughout my time here, I have been blessed with meeting other students who share the same enthusiasm for learning. I will always be thankful of Metlab's Lisa Rebetsnitch, Annette Lettsome, Fan Xiao, Kayra Hopkins, Jerry Pinero, Zubin Abraham, and Sarah Coburn for their part in dissecting my work, often repeatedly. A special shout also for my fellow CSE 101 crew who I spent so much time with: John Hettinger, Jacob

Brown, Alex Peer, Jignesh Patel, and Mayur Mudigonda for their friendship and conversations regarding all things.

It is also important for me to also acknowledge the people that have shaped my path towards where I am know. Dr Yudho gave me my first programming in Basic book in Melbourne. Dr Windy Gambetta and Dr Bambang Parmanto conspired to send me to the United States to pursue my studies. Special thanks to Dr Wayne Dyksen who championed me to the admission committee here at MSU.

Finally I'd like to thank the Elhedon Indonesian community including Putri Arum Jati, Perdinan, Mujiburrohman, Dwi Agus Yuliantoro, Yudi Wicaksono, Ririn Seamount, Irfan Prasetya, Saraswati Haruming, Tiara Ahmad, Ainur Rosyid, and Arinanda Mamahit among others who have kept me sane during my PhD years.

Is it finished? No, a Phd is just the beginning.

Reza Ferrydiansyah

Table of contents

List of Tables

List of Figures

1 Introduction

Object animation in computer applications is virtually a requirement in current computer technology. With the rise in ease of access to computer graphics technology, animation is now used in a wide range of applications as an interface element.

Animation describes the temporal manipulation of elements in a computer graphics system. Animations describe scenarios, events, and other information allowing users to quickly understand what is happening. Animation allows users to easily manipulate scenarios as the application allows, allowing better understanding and insight in the various scenarios and the relation of objects in those scenarios. One of the most important fields of endeavor in computer graphics and animation is the creation of apparently realistic animations

The creation of animated virtual humans can be thought of a sub-problem within this field. The terms *animated human* or *virtual human* in this paper refer to human characters that has been rendered by the system on a display or to use as an element of a virtual/augmented reality system. An autonomous character that can make its own decision based on an algorithm (albeit in a limited way) is called an agent. An agent that is presented as a virtual human is referred to as an *embodied agent*. Creating realistic human animation requires realistic modeling and rendering of objects, modeling of the object's physical characteristics, creation of realistic object behaviors, and object interaction with the user and other objects in an environment [1].

Virtual embodied agents are graphical renderings that represent humans in virtual and augmented environments. Humans respond to animated humans whether controlled by a real person, or by an algorithm [2]. Virtual humans have been used as characters in animated movies [3], actors in interactive story systems [4-6], controllable agents in computer games, tutors in educational software [7-9], and presentation agents [10]. They can also appear as a guide to a

person in an unfamiliar land, serve as a trainer who demonstrates and oversees, or provide the interactive component of a future user interface. These virtual humans can also be controlled by humans or act in a predefined sequence of actions.

Due to our everyday interaction with other humans, we are used to seeing and noticing natural movement everyday. When embodied agents do not act naturally even only slightly, the user's focus will be distracted by the unnaturalness of these virtual humans [11-13]. A prime example of this is in the movie Polar Express, where critics said that the movie was a good movie story wise, but they were bothered by the unnaturalness of the eyes [3]. Unfortunately this was one of the bad points of the movie which was oft repeated.

1.1 The problem of natural movement

There are two main reasons why the animation of natural movement is difficult. First, the human body consists of a set of parts which are joined together. Each part moves according to a specified set of degree of freedoms. Animation for a virtual human body is usually achieved through the use of a skeleton model [14], [15]. The skeletal model consists of various joints. Each joint has up to 3 degrees of freedoms. The joint allows the bone to rotate in the x, y, and z axis. The rotations are adequate to simulate any human pose. Chapter 2 discusses the formulas used to calculate position based on these bone rotations in more detail.

Figure 1-1 shows the skeletal model used for this thesis. This skeletal model is based on the skeletal model for the ASF specification [16]. The skeletal model consists of 30 joints and 56 degrees of freedom.

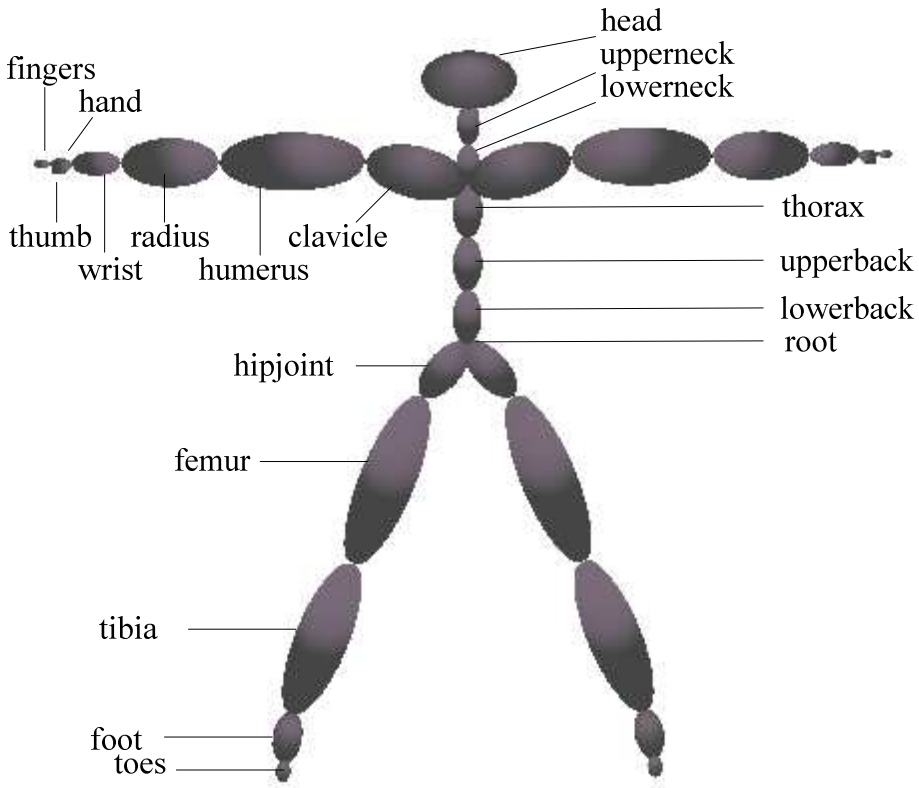

Figure 1-1. An example of a skeleton model for human animation. The skeleton model is used based on [16].

Because there are multiple joints and bones, the human body is flexible enough that it is able to achieve the same task with multiple poses. There are, for example, different poses for a person to retrieve something that is on top of a table. There are many possible different combinations of angles for each pose. Many of them will be deemed unrealistic to the trained human eye.

This flexibility is great for humans, but a headache for animators. Out of all these possible poses, an animator must choose a 'best' one. This usually means that the animator must choose the most natural pose or the pose that sets the human up for the next movement sequence.

What does it mean to have a natural pose? In this paper, naturalness is defined simply as the pose selected by most people given the same constraints. The constraints are positions and orientation of body parts, positions of obstacles, and external physics constraints (such as gravity or motion).

Whether a motion or even a pose is deemed to be natural or not will be observed by different criteria. Naturalness may be graded based on energy, meaning it will be based on the notion that the human will take as small, or least costly, a motion as possible to achieve the constraint, or to hold that constraint for a long time. Naturalness may also be based on the starting pose. If we start with an awkward starting pose, the natural thing to do to get to a constraint may simply continue with the awkward pose even though it is inefficient in the long run.

Another very important factor is the environment. For example the position of target objects (where the animated human wants to touch or avoid) is obviously a primary factor in the resulting pose or animation. The positions of other (non-target) objects are also important because animated humans in most cases are assumed to follow the same physical rules that real humans adhere to. Therefore it is very important that human body parts do not, for example, go through any of the available objects.

Because of interaction requirement, sometimes a virtual human must face a certain direction. Humans naturally point their eyes at what they are attending to or attempt to maintain eye contact with the user. Eye contact can also be used as cue for the users [17]. For example if the application wants to induce the user to pay attention to a particular object, the virtual human may be made to look at the object.

Finally naturalness may also depend on emotions and mood. People with different emotions tend to perform different types of actions [18]. A naturally disgusted motion will be different than an angry motion. A successful virtual human agent must take all of these factors into account when performing a particular motion.

1.2 Current Methods for Natural Animation

In applications where the virtual agent has no need to think, plan or interact with the environment, or in other words have no autonomy, an artist determines the action of the virtual human at each time step. There is no internal agent representation of the current state or of the actions performed by the agent. Everything in the world is fully controlled by the world designer. This makes this class of agent relatively easy to create. However, it can be quite tedious to create animations frame by frame. .

Tomlinson [19] refers to agents having non-autonomous behavior as *linear agents* because this method is not suited to characters that must interact with users. Consequently this method is of more interest to those creating motion pictures or other static animations [20], [21]. Tomlinson describes the various differences between linear and autonomous agents as well as differences in the applications and usage of these agents.

Another way to create linear animations is through the use of scripts. Scripts are code in a human-readable language that will be translated by the animation engine into movements. Script languages are usually created by the animation engine developer and used by the artist to create the animation. Some example of systems based on scripting language are the Improv system [22], and STEP [23]. Similarly the Jack architecture uses a natural language system to describe motion and intention [24].

The main difference between scripts and manually created animation is that scripts are internally known by the agent. Scripts can also make it easier to create new animations as it is quite easy to copy movement from one part to another (or from one agent to another), and simple programming constructs such as loops, sequential presentation, and blending of actions can be easily created.

Because an artist creates individual frames, it is the responsibility of the artist to create the most natural animation. As artists, they may have the knowledge and the experience to do this, possibly to even do this well. In many cases the results are human animations that are very natural. Of course the drawback to this method is that it will take a significant amount of time for the artist to create each frame and a significant amount of talent on the artist's part.

Work has been done to create animations that are fully computer generated. Given a set of constraints, an algorithm calculates the correct angles depending on the starting position and other factors. An algorithm that calculates a set of angles based on one or more target position is classed into the inverse kinematics method [25]. The forward kinematics method calculates the position of each point on the body based on the angles of the joints

In most cases of human animation, inverse kinematics methods are quite difficult to calculate algebraically. Therefore inverse kinematics solutions are often solved iteratively. The number of iterations may be quite high and therefore an application may take some time until a solution converges.

Simple inverse kinematics solution algorithms such as the iterative Jacobian, do not necessary consider naturalness as a goal. The main objectives of these algorithms are simply to get a correct set of angles that satisfy the constraints.

In animation where the virtual humans walk, run, or jump, dynamics and spacetime calculation is often used [26-28]. The dynamics and spacetime calculation are based on physical forces that apply to each body part. Appendages are influenced by gravity, inertia, and other forces. By calculating these forces, a more natural movement is generated.

Sampled movement, or motion capture, captures movement data directly from real people. These people are usually given a suit with reflectors to wear [29], [30]. A set of cameras capture their every movement. It is impossible to capture all possible motions that a human can perform, therefore new motion data must be generated by manipulation of the existing data.

Manipulation of existing data can be as simple as splicing the motion capture data and joining them together to create different motions (motion blending). The motion capture data may also be transformed to fit different characters (motion retargeting). Finally new motion data may be generated which are similar to the pre-existing motion data. The difficulty is, of course, how to generate data that encompasses various positions from the available data and still make them realistic.

Current techniques are often the result of integration of techniques from multiple categories. For example, motion can be created via kinematics techniques, then enhanced with dynamics techniques. Another common example is to use captured motion data and use dynamics techniques to create a new set of movements that both adhere to physical laws and are based on motions of a real representative human.

1.3 Proposed Solution

The problem of animation generation can be stated as follow: A starting position angle that can be represented by the set of angles P_0, and a set of constraints $C_{0..k}$ which are the

constraints of all positions in the movement are supplied as input. Some of the constraints may be *hard constraints* that must be met, while others may be a set of targets which the motion must try its best to fulfill. The objective of the algorithm is to find a motion that allows a smooth natural movement between P_0 and P_t and satisfies all constraints at time t.

In general, constraints can be positions of various body part, the angles of each degree of freedoms, orientation of a particular bone, or the position of other objects that cannot collide with any body part. In this thesis however, the focus is on Cartesian coordinates of the body part or effectors relative to the root bone.

The problem with any animation generation is that there are many possible solutions that meet the constraints. An algorithm must find the correct pose from all possible solution, which may not be natural. This may entail rejecting unnatural movement, or changing angles to make it more natural.

However if all possible animated human poses in the search space are natural poses, there would be no need to attempt to naturalize the poses. The focus will simply be on satisfying all constraints.

All poses can be placed in a hyper-dimensional configuration space, where each dimension represents the value of one joint angle. By pruning all points in the space that represents unnatural poses, it may be possible to use the resulting space to find natural motions. However, it is difficult to define the limits of the area of these unnatural regions. The next problem posed is limiting the movement generation algorithm so that it does not wander into those spaces once the area is found.

The solution proposed in this thesis is to create a hyper-dimensional sub-space where all movement inside this space is natural (instead of defining pockets of unnatural poses). The idea

is to first determine relationships or correlations between the various angles of the bones in a natural movement. This correlation data is found by observation of natural captured data.

Principal Component Analysis (PCA) is a commonly applied method that captures the correlations between values in a vector. PCA can also create a reduced dimensional space with the strongest correlation between the angles in the natural poses [31]. This PCA reduced space represents the natural space because points in that space can be transformed into an original space which represents poses having the same variance as the motion capture poses, and therefore consists of natural poses. Our algorithm performs the search on the natural space and each point on the natural space is transformed back to the correct angles.

As PCA is a global statistical method, it summarizes data globally. This method tends to merge various data together, and as a result, some of the motion details get lost. The method described here uses multiple PCA spaces, where each space corresponds to a segment of the skeleton (segments can be overlapping). Each space finds the variance of one segment. This enables better control of each individual segment, and a better chance for finding natural poses that meet the constraints.

Natural space will be created from motion capture data. Creating a natural space that encompasses a wide range of movement will require a large amount of data. However due to the number of available data, as well as time and financial resources, the data collected will probably not be large enough. Multiple natural spaces will have to be created from data segregated by motion type or body parts. The motion generator will need to choose the correct spaces and join data from multiple natural spaces.

1.3.1 Thesis Contribution

This thesis:

1. Demonstrates that it is possible to synthesize new user-constrained poses and motion from motion capture data having the same naturalness characteristics as the original data.
2. Describes an algorithm that finds a pose by searching in natural space that meets a user specified positional targets for multiple effectors.
3. Shows that, by segmenting the body into multiple overlapping spaces, it is possible to find poses that are more natural than traditional inverse kinematics and regular single space PCA, while also more reliably achieving specified constraints.
4. Describes a method to measure naturalness of poses based on the probability distribution function of the motion capture data.
5. Gives details on how to create motion by traversing a natural motion/phase space. This method creates motion that are quite similar to natural motion compared to straight interpolation

1.4 Document Structure

The details of kinematics and dynamics calculation techniques as well as data based techniques are discussed in Chapter 2. Chapter 3 discusses various methods of using a statistical summary of motion capture data, including the PCA to create a natural space. Chapter 4 describes the algorithm and results of using multiple natural spaces in creating poses. Chapter 5 shows the results of the motion generating algorithm. Finally a conclusion, possible applications and possible future work is given in Chapter 6.

2 Pose and Motion Calculation

The act of animation is simply rendering multiple frames one after the other at such a speed that the viewer perceives it as a continuous motion. To create animation, a set of key frames is generated between one pose and another. The computer interpolates the positions between key frames, adding additional frames to make a smooth transition between poses for the viewer.

In the simplest case, an artist creates all the necessary key frames leaving the computer to fill in the transition frames. If the key frames are spaced fast enough, the resulting interpolated frames will have a smooth and natural characteristic to them.

The problem that this thesis addresses is when key frames are not created by any artist. Constraints such as the position or orientation of various body parts and physical characteristics are given for the whole animation sequence, or for the final pose only. In this case, the key frames must be generated automatically by the software.

Two ways this can be achieved are to find an end pose then create key frames that lead up to that pose or to generate multiple pose paths from a starting position and choose one path which transistions the animated human to that pose.

The algorithm to solve this problem is linked with the human model used. The skeleton model which is a hierarchical model of human geometry and physical motion is often used, and is the one that will be utilized for this thesis. The skeleton is a set of rigid bodies which move relative to each other in a hierarchical structure and move the overlaid flesh with them. Most models of bipeds in computer graphics systems utilize an internal, invisible armature, the skeleton of the object, meant to accomplish the same functionality. The skeleton does not need to conform to an actual human skeleton, either in size, shape, or relative motions, although some

do. The movement of the animated character is specified by the location (position, rotation) of the skeletal bones. The skin (which is usually a triangle mesh) is correlated with one or more bones and will move according to the movement of the bones.

Work on skeletal representation began in the 1970's [14], [15]. Since then, the skeletal model has been the primary method for human animation. Skeletons have also been used as the basis for captured motion data [32].

There are many standards and recommendation in creating skeleton, each standard allowing various bone properties. In most cases, bones are a 3 dimensional object having an origin and a length. The origin of a bone can be determined from either the end position of its parent bone [16], an offset from the end position of its parent bone, or an offset from the center coordinate[33]. Most bones have a rotation value in the x, y, z axis which allows them to move into various position. Some standards allow scaling in one or more axis or the specification of limits. The terms joint and bone are used interchangeably, since it is only the joint location that is utilized. The bone itself is not assumed to have a specific geometry. In general, the length is provided mainly for display and user interface purposes.

2.1 Kinematics Method

Kinematics method uses angles of bones to perform movement. Forward kinematics is the calculation of a position of a particular point in the skeleton given various angles of all the bone in the skeleton. Forward kinematics calculation is a chain of rotations (representing angles of joints in the correct axis) and translations (which represent bone length). A simple set of matrix multiplications composes the rotation matrices and translation matrices. The rotation matrices relative to the x, y, and z axis are given in Equations (2-1), (2-2), and (2-3) respectively.

$$Mr_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2\text{-}1)$$

$$Mr_y = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2\text{-}2)$$

$$Mr_z = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2\text{-}3)$$

A translation (t, u, v) in the x, y, z coordinates can also be represented as a matrix:

$$Mt = \begin{bmatrix} 1 & 0 & 0 & t \\ 0 & 1 & 0 & u \\ 0 & 0 & 1 & v \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2\text{-}4)$$

Given the coordinates of the root bone as (x_r, y_r, z_r), the end position (x_p, y_p, z_p) can be calculated by multiplying the root bone with the rotation of every bone, by the translation (or direction and length) of every bone Equation (2-5).

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix} \times Mr_x(\alpha_{x1}) \times Mr_y(\alpha_{y1}) \times Mr_z(\alpha_{z1}) \times Mt_1 \times Mr_x(\alpha_{x2}) \times Mr_y(\alpha_{y2}) \quad (2\text{-}5)$$

A common problem in animation is to find a pose that will satisfy a constraint that a particular point on a specified bone is at a coordinate T. This problem is called the inverse kinematics problem, and it is an inverse of the forward kinematics formula. Unfortunately this calculation is very difficult to solve algebraically. There have been works on solving systems with limited (6) degrees of freedom [34]. However an animated human skeleton typically consists of more than 20 degrees of freedom.

To solve inverse kinematics problems, iterative methods are often utilized. One basic iterative method used is the Iterative Jacobian method [25]. The coordinate of the effector X is a function of the various joint angles (Equation (2-6)). The derivative of X can be calculated using the Jacobian given in Equation (2-7). The Jacobian calculates the partial differential for each value for each axis (x, y, or z) over the partial differential for a particular angle Equation (2-8).

$$X = f(\alpha) \tag{2-6}$$

$$dX = J(\alpha)d\alpha \tag{2-7}$$

$$J(\alpha) = \partial f_i \big/ \partial \alpha_j \tag{2-8}$$

By inverting the Jacobian the angle movement can be calculated.

$$d\alpha = J^{-1}(\alpha)dX \tag{2-9}$$

The new angles can be calculated as:

$$\alpha_{t+1} = \alpha_t + d\alpha \tag{2-10}$$

This step is repeated until the ending solution is close enough to the solution. This iteration can actually take a while depending on the size of dX. A large values for dX will result in a more imprecise angle difference. A small value for dX means that it will be slower to converge to a solution.

Other iterative methods calculate the difference of angles per degree of freedom instead of for the whole skeleton. One of these methods is the CCD or Cyclic Coordinate Descent method [35]. In CCD, the angle between a joint and the effector and between the joint and the target is calculated. The difference of that is the rotation angle needed for that degree of freedom. The CCD iterates from the joint closest to the effector to the root of the skeleton.

The problem with these two inverse kinematics problems is that there is no notion of naturalism. All bones move equally with the main goal of finding a solution, any solution. Weighted inverse kinematics proposed by Meredith [36] allows different motion which depends on the weight given to different part of the bodies. Ideally, such a scheme would create a more

natural movement, however it is not clear how to determine the weights for each of the bones to create the most natural movement. Creating weights for each bone does not solve the problem of the speed or the number of iterations it takes to find a solution.

2.2 Dynamics Methods

In dynamics-based models, the movements of animated objects are based on the physical laws that they are subject to, such as momentum, gravity, friction, and acceleration. The model must also take into account the properties of each object such as the mass and the shape of the object. A set of formula is created that, based on the physical laws and the object's properties, determines where each object is at each time frame. Various models also allow objects to link with other objects and thus constrain both object movement [37], [38]. The problem of calculating the position and state of each object given forces that are known is called the *forward dynamics problem*.

The solution to the *inverse dynamics problem* determines the forces or velocity needed for an object to move from one position to another or to stay in motion with respect to its current position, mass and shape, as well as existing external forces [37]. Given a set of constraints, inverse dynamics can be used to calculate the motion from a starting pose to a pose that meets the constraint.

A combination of dynamics and kinematics are often used [39], [40]. The inverse kinematics element of the algorithm determines the position of the joints at various points. The dynamics element calculates the speed and the trajectory of each bone segment.

The Newton-Euler formulation calculates linear acceleration and angular acceleration and is often used to calculate the desired speed of movement [28], [41]. The Newton formulation

states that, in linear systems, force equals mass times acceleration. In the Euler formulation of angles, the moment (or sometimes torque) is the inertia multiplied by the angular acceleration.

Newton equation:

$$F = ma \tag{2-11}$$

Euler equation

$$M = I\omega \tag{2-12}$$

Other methods include using the potential and kinematics energy for the Lagrange formulation. Witkin and Kass [42] proposed a spacetime computation which uses Newton's law and takes into account the starting and ending position of the object, the starting and ending time of the movement, as well as any additional constraints to calculate the trajectory and the forces needed to perform a motion.

Ko and Badler used inverse dynamics to animate human locomotion [43]. Based on the location of the objects at a time t (determined by a script), they determined the positional and angular acceleration of all links in the system using the Newton-Euler dynamics method. The kinematics system adapts to the dynamics calculation result. A set of dynamic equations for walking motion was created in [26]. A numerical integrator approximated the forces and the torques needed for the motion of both the upper and lower body.

A very complex and detailed model of the body has been created by Lee et al [44]. The model includes a complete skeletal/bone model, muscles, and skin. Although such a model may be more precise, the sheer size and complexity may make it unfeasible to build. For specialized programs, it may be better to create dynamic models of the body part that is needed, such as a hand dynamic model for grasping [45].

Motions created by dynamics calculation are much more natural than those created by the pure kinematics calculations. The motions depend on the physics models used. The more

complete the physics model, the more physically correct it is, however it may take longer to find a solution to such problems.

2.3 Multiple Constraints

Whether solving for inverse dynamics or a combination of inverse kinematics and inverse dynamics, many problems in animation requires the animation to meet several constraints. These can be pose constraints, dynamic constraints, time constraints, and/or mechanical constraints [28], [46], [47]. Pose constraints determine where the position of a particular bone or bone segment is at a particular time. Mechanical constraints determine various physics laws at work on the body. Time constraints refer to the time that an agent must be in a particular position. Finally, dynamic constraints ensure that physics laws such as Newton's second law are met at all times. Other constraints that are also commonly used in the calculation of movement, such as constraints for collision detection [40], energy [48], balance, and comfort [43].

In some cases not all constraints can be met at once. There might be conflicts that prohibit all constraints to be met. To solve this problem a priority scheme or a weighting scheme is used [47]. The solver will try to solve the constraints with the highest priority first, or tries to iterate towards finding solutions having the highest weights.

There are often other *soft* constraints, which do not have to be satisfied, but in which the algorithm tries its best to achieve. These soft constraints are sometimes represented as an objective function, the idea being that the algorithm must minimize (or maximize) the value of these objective function. The objective function is used to find the correct answer in the event of multiple answers (for example, taking into account the minimization of energy).

Constraints can be formulated as functions. Usually constraints are formulated as a function that results in 0 when met, that is $C_i(X_j)=0$. The objective function is useful when the problem is under constrained which will allow us to have multiple solutions to the problem.

When the constraints and the objective functions are all linear, a linear programming algorithm such as the simplex algorithm can be used. In most cases however a non linear algorithm must be used in order to solve these problems. The steps in solving a non-linear constrained optimization problem is to simply find repeatedly an estimate solution based on the gradient of the objective function with regard to the solution. The algorithm used is a in a class of algorithms called Sequential Quadratic Programming methods. Due to space constraints this document does not go into details on the various solution algorithms and instead refer readers to [49].

2.4 Data driven animation

A lot of work has examined using captured human-motion in so-called *data-driven computer graphics*. An instrumented human is asked to perform movements which are then captured by a computer system. The advantage of using captured human motion data is that there is no need for an animator to input the animation data manually. Furthermore, the result of captured human motion data is natural and human-like because it is a sampling of real human motion.

In a common setting for capturing data, a subject is placed in a studio, wearing a special (usually plain and dark) suit. The subject will have special markers at different places to track the location of various bones. Cameras (or perhaps only one camera) located at various angles record the subject as he or she proceeds with various motions.

Each recorded frame is automatically stored in the database. The computer locates the coordinates of each reflector. The main reason that the suit is plain is to reduce interference with the reflector. Once the location of every reflector is found, the angles of various joints can be calculated [29], [30].

The captured data consists of the parameters of the movement for each frame, namely the position and orientation of each skeleton segment. A single capture sequence can have many angles and joints. ASF formatted data, for example, has 29 joints with up to 3 degrees of freedom for each joint. The captured data will typically be stored at a frame rate of 15-30 frames per second.

When playing back a captured motion sequence, the motion player simply sets the angles of the joints according to the frame data. Once the pose is drawn, the player waits a set time (depending on the number of frames per second there are), clears the image and draws the next frame in the same way.

The amount of data accumulated this way is quite large. To limit the size of the files or the database, subjects typically perform a single short motion. This motion can be as simple as walking, running, jumping, or it may be more complex such as dribbling a basketball [50]. By grouping sequences, users of the motion data can choose the data that is useful easily. The users do not have to select frames within the motion capture database. Instead users select the motion needed and use all of it. There is also some work on automatically segmenting long motion capture sequnences based on similar actions [51].

One problem with this method is that the range of possible human motion is quite large. Even a simple basic movement may have many variations. For example, a person can jump while facing to the left and also jump while facing to the right. There are infinite variations to a

basic movement, and as such, it is impossible capture all of it using human actors. New motions must be created from existing captured data sequence.

2.4.1 Motion blending

One way to create a multitude of animation sequences is creating a weighted combination of frames from different captured sequences, a method known as *motion blending*. In Kovar [52], Gleicher [53], and Arikan and Forsyth [54], animation sequences are created by creating connection graphs for frames that are similar. Captured human motion is transferred to a skeletal format. Each clip is broken into a small set of frames and each frame set is connected to other frame sets on different clips based on a similarity function.

A simple approach to finding similar frames is by calculating the difference between the angles for all bones (perhaps include velocity) [55]. Kovar [52] opposes this idea because this idea does not take into account the importance of joints, differences caused by translation or rotation on the root bone, nor velocities and acceleration of the motion. Instead they find the similarity between point-clouds created over k contiguous frames. The similarity is the total distance of each point in one point cloud with a corresponding point in the other point cloud (which comes from a different motion). Pairs of frames between two motions which have very low difference are joined together to create graph edges.

Linear blending simply switches motion when the two clips are very similar. An angular method blending proposed by Shum joins motions based on the angular momentum trajectory [56]. Blending can be performed in the middle of an existing motion, instead of waiting for it to end, creating a smoother transition between clips. Heck et al. used a splicing method, which joins upper body from one clip with the lower body from another clip [57]. To join the two segments together, they perform time and spatial alignment, as well as a posture transfer which matches

the location of the shoulders and hips for each clip. Mutlu blends frames together to create new walking motion on a different path based on the position of the foot plants [58].

An animation sequence is generated by taking a sequence of frames from a clip and then adding another frame sequence from another clip continuously until the character is in the desired position. The number of graph nodes and edges can be very large, therefore an efficient algorithm is needed to perform the search. This can be done by using various graph search methods to find the best motion between the starting pose and the ending pose.

Lee [55] uses reinforcement learning to find the best path between all various starting and target poses. Reinforcement learning uses rewards and discounted rewards to determine which path to take in each graph node. Once the best action for each graph node is already met, it is a trivial to create animation that encompasses various nodes.

This concept of path finding for motion has been developed further to create dynamic motion controllers. Dynamic motion controllers allow the creation of motions based on states, where each state consists of the configuration, speed, torque, and environmental factors [59], [60]. Each state includes the current pose, motion, and even the environments. The controller chooses the next state based on the current state, dynamic constraints, and the environments. Use of a controller allows some measure of variability in the motions created even in unpredictable environments [61].

Interpolation is performed on the newly connected frames to create a smooth transition. Interpolation creates k number of frames where each frame is a pose between the two existing frames. Linear interpolation, the most basic interpolation method, uses the starting angle and the ending angle (of each joint) to create an equal size interval which is used to calculate the angle at

every frame (Equation (2-13)). A better method is to first convert the Euler angles to quaternion and perform a linear interpolation on the quaternion [62].

$$\alpha_t = \alpha_0 + t \times \left(\frac{\alpha_k - \alpha_0}{k} \right) \tag{2-13}$$

A spline interpolation can also be used to connect two frames together. The difference of angles between two frames in spline interpolation is not fixed. The difference of angles between frames takes speed of motions into account. However, spline interpolation typically requires more than 2 sample points in order to get the best interpolation fit.

Motion capture data has also be used for basis of change. Yamane [63] and Pan [64] uses RRT, a randomized tree search to find possible paths between the starting pose and the ending pose. The points found represent collision free poses. Motions between the poses are created based on motion capture data, by finding motion clips that are meets the constraints.

2.4.2 Motion Synthesis

This method of blending motion is adequate when the range of movement in an application is limited. A prime example of this is sport games. In sport games (such as American football, or soccer) the main interaction is between a human and a ball. By capturing various interaction between person and ball, some which are basic, and a few specialized movements, various combination can be created that will allow natural movement over a whole game.

However if the range of movement is larger, or the terrain is unpredictable, motion blending will not do the trick. There is a need to create new motion based on existing captured data, a method sometimes referred to as *motion warping*.

Abe et al. [32] used a base frame of animation data captured from actors to create a family of similar frames. Each generated frame has each character in the same position as the

base frame but translated (in the x or z axis) or rotated (using the y axis). Each new animation preserves the physics characteristics of the base animation.

New motion can be created by warping or changing existing motion capture data to meet different constraints. Van Basten uses a greedy search to find feet motion that is similar to a new set of stepping motion [65]. Warping is then performed on the resulting steps.

Different body types have different correlation between joints [66] and thus different movements caused by body type. A lot of work on motion warping has focused on adapting motion from the captured data to other characters that may have different builds, shapes and characteristics, a process known as motion retargeting. Different body types have different correlation between joints and thus different movements caused by body type. Meredith and Maddock [36] change the motion capture data using weighted inverse kinematics. By changing the weights of various joints, they are able to create personalized movement for different types of people. The weighting of various parts was also performed by Popovic [28] to create different movement based on personality.

3 Natural Spaces

Multi dimensional data points such as the joint angles for motion capture data frames can be placed in an n-dimensional space. Each point represents one pose, and each value in a dimension represents the angle of the degree of freedom. This space is commonly known as the configuration space.

A theoretical natural space N-space can be thought of as a sub-space in the configuration space where human poses are natural. The dimension of N-space can be smaller or equal to the dimension of the configuration space. Search for poses and motions that meet constraints using N-space only returns poses and motions that are natural. Therefore there is no need to make the results natural.

One way to create N-space is to have it consist only all the points that are found from the motion capture data. Unfortunately, this severely limits movements, as poses that were not found on the original motion capture data will not be used even though it may be natural.

Another way to see this problem is by viewing this problem as a classification and learning problem. Given a set of learned data (motion capture data), classify any points into either natural or not natural. In other words, divide the space into N-space and N'-space. The next chapter discusses various ways to create N-space from the learning of motion capture data.

3.1 Creating Natural Spaces

The animation generation algorithm will start with a point in N-space and find a new point which meets all the criteria. This is done by testing various points in that N-space. In order to do this faster the created N-space must have the following characteristics:

1. There is an easy way of determining whether a position is in N-space or in N'-space. Every time a candidate point is found, the point must first be tested on whether it is in N-space or not. If the new point is not in N-space, this point is discarded.
2. A reasonable estimate can be made of how close a new position in N-space is to satisfying the various constraints of the animation based on the information provided by the space and also the resulting pose of other known positions. This allows the search algorithm to intelligently choose points that will take it reasonably closer to the solution point.

One idea to create N-space is to use model boundaries in the configuration space separating natural space from unnatural space. Voronoi cells or Parzen Windows [31] allow the creation of such borders. However in order to create these boundaries, both negative and positive samples (that is natural and unnatural poses) had to be available. Motion capture data only provides positive samples of human motion. Another problem with using this method is the complexity of the algorithm to create the borders as well as to detect the borders is O(n log n) where n is the number of data points [67].

To create N-space, various methods were considered. One popular classification method is the use of clustering. However the data obtained is not a good match for this method. There are 2 groups for this classification, a pose is either in N-space or in N'-space. Once again there is a need for negative or unnatural samples which are not available. Therefore any classification method that requires negative examples such as clustering can not be considered.

Statistical modeling allows the creation of models that describes the motion capture data. Synthesis of new motions is possible using these models. One of the earliest works of statistical modeling on motion capture data is done by Pullen and Bregler [68]. They used wavelet

decomposition and a Gaussian kernel to model a Wallaby's motion. Realistic motions was be created by calculating the conditional probability of each frames. Stylistic Hidden Markov Models have also been created from motion capture data. New motions were created by doing random walk on the resulting HMM state machines [69].

A multivariate distribution function can also be generated from the motion capture data [31]. This distribution function can then be used to determine the likelihood that a pose is a natural pose. N-space can be defined as any point that has a natural pose likelihood score of over P. The distribution function allows calculation of new points which have similar or higher likelihood score. However the distribution function by itself will not help find points that are closer to a solution. The search function needs information about the relation of points in the N-space to the solution to determine the best motion path.

Component analysis methods are able to summarize the relationships between various dimensions of the original data. Typically component analysis methods such as Principal Component Analysis, Independent Component Analysis, and Nonlinear Component Analysis allow the transformation from one space to another space that can better describe the data [31]. Component analysis methods are also typically used for creating a lower dimensional space of the data. Other lower dimensional methods such as MDS and Isomaps have also been used to model motion data [70]. The problem with MDS and Isomaps is that they typically need a large amount of space as they need to store the distance between all possible frames.

This thesis uses Principal Component Analysis (PCA) as the main method for creating natural poses and motions. PCA allows a straightforward summary and generalization of the motion capture data. The number of PCA dimensions can be reduced to limit the search space,

which is compatible with the aim of creating a natural search subspace. Synthesizing poses and motions from PCA space does not need for significant amount of space and computation.

3.2 Principal Component Analysis

PCA is a statistical technique that uses the variance of data to allow data points to be transformed into points in a reduced dimension space. An extended primer on PCA is given in Joliffe [71]. The points in the reduced dimension space retain characteristics of the original data set that contribute to its variance [31], [46]. Due to the smaller number of dimensions, data is easier to analyze.

To calculate the PCA of a data, first find the mean and the variance for each dimension. The covariance matrix C can then be calculated from the mean and variance. Given the covariance matrix C, a set of eigenvalues λ are calculated according to Equation (3-1). Once the eigenvalues are calculated, the eigenvectors e are calculated according to Equation (3-2).

$$|C - \lambda_i I| = 0 \tag{3-1}$$

$$Ce_i = \lambda_i e_i \tag{3-2}$$

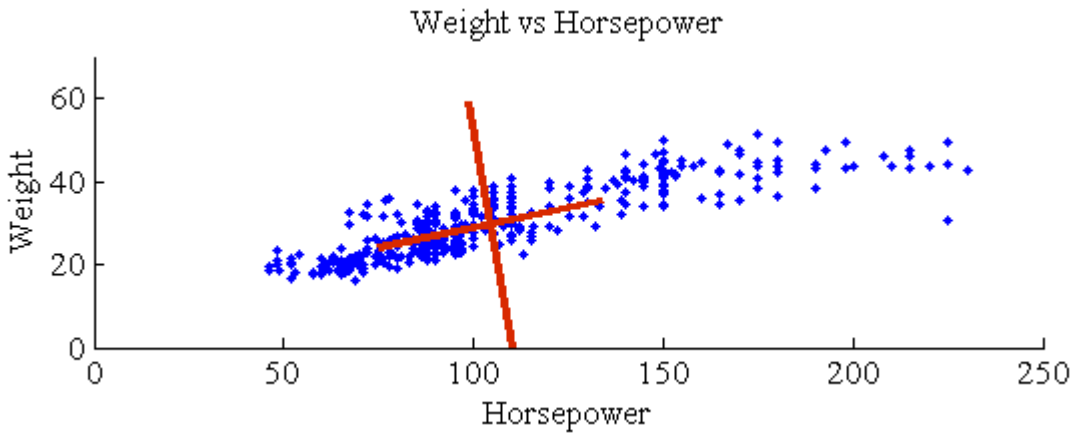

Figure 3-1 Weight vs. Horsepower data and the PCA axis. For interpretation of the references to color in this and all other figures, the reader is referred to the electronic version of this dissertation.

The set of eigenvalues is combined together to create the transform matrix T with a dimension of m x n where m is the number of dimensions in the original data and n is the number of dimensions in the reduced space. T is used to transform any data point in the original space into a point in the reduced dimension space. The value of any dimension in the reduced dimension space is actually a weighted combination of values from the original space.

To clarify the concepts here a chart was created to show the weight vs. horsepower data for various cars (Figure 3-1). This data was part of the Auto MPG data set from the UCI machine learning repository [72]. The lines in the figure show the scaled eigenvectors which form an orthogonal axis. Figure 3-2 shows the result of transforming the original data with the PCA transform matrix created from the eigenvectors.

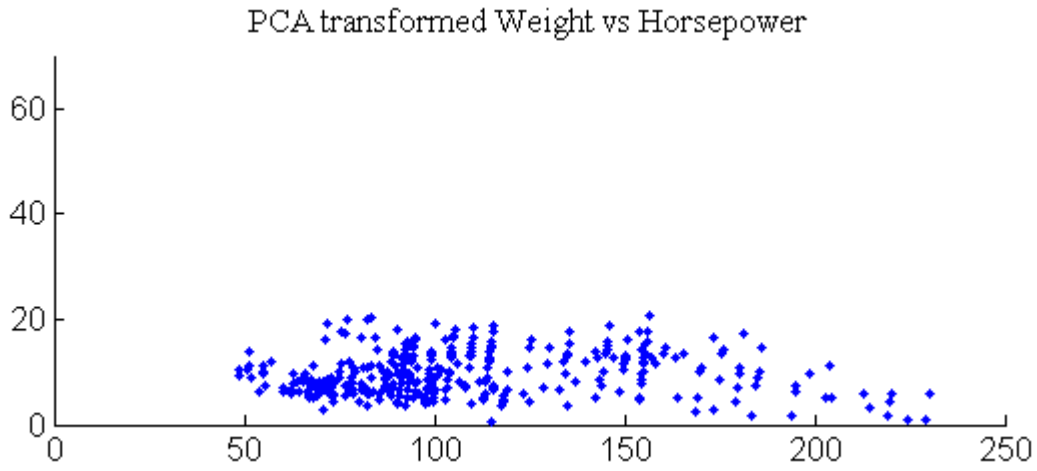

Figure 3-2 Transformed data to PCA space

If the dimension is reduced to one dimension, all the data will be on a single line (Figure 3-3). This reduced dimension space limits the data that can be synthesized. New data when transformed back to the original space will be on the main axis line of the PCA.

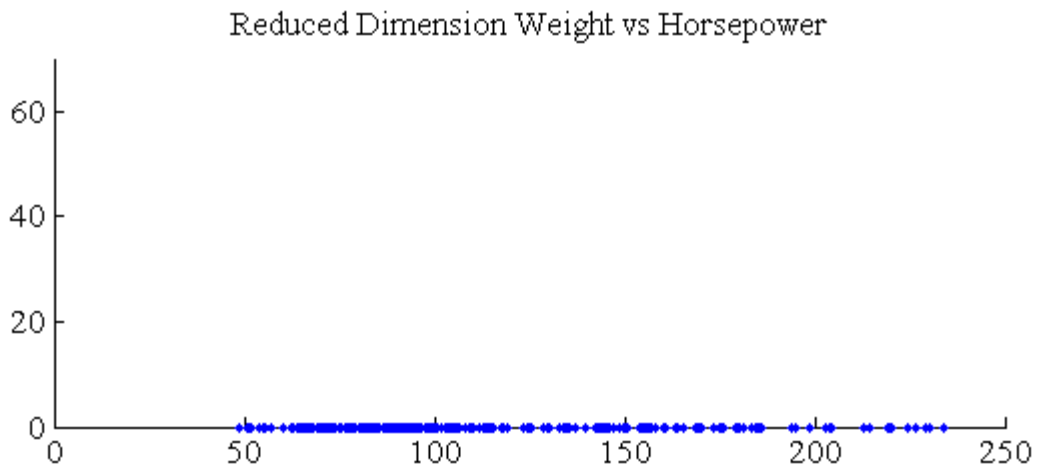

Figure 3-3 PCA data reduced to just one dimension

The transformation matrix to calculate the position of a point in the reduced dimension space is not a square matrix (because the number of dimensions is different). Although no inverse matrix is available for this non-square matrix, a pseudo inverse matrix can be created using the Singular Value Decomposition method [73].

Any point in the reduced dimension space can be inverted back to the original space using the pseudo inverse of the matrix. These points will have the same covariance between points as any of the original motion capture data that is in the original space. The next few sections show that synthesized points in the natural space (within some boundary) is natural. Therefore the reduced dimension space created by PCA is a good candidate for N-space.

3.2.1 Use of PCA in human animation

In PCA, the eigenvectors can be ordered by importance. The first k set of eigenvectors can be used to represent a certain percentage of the variance of the data. This property has been used to create compression algorithms on human motions using PCA [74], [75]. The quality of the motion reconstruction depends on the number of dimensions used in such a scheme.

PCA is often used to summarize motion capture data and then to generate new poses. Li shows that PCA can be used to create general unconstrained motions [76]. Some, like Tilmanne, uses PCA to create walking motions that are based on different emotion expressions [77].

Johnson's PhD dissertation focuses on using PCA on motion capture data based on quaternion [78]. The aim of his dissertation was similar to this work, which is to use the PCA space as an "expressive" sub space in order to synthesize poses and motions. His work focuses on using a single PCA space for the whole body, while this work focuses on multi PCA spaces. In his work, he also mentioned an inverse kinematics solution using the sub space, but never actually implemented.

Most of the work on PCA as a means of representing motion has been based on motion categorized into specific tasks [79], [80]. This thesis described a method that takes general motion data and performs PCA on body parts segments to better control the motion to meet the user's constraints. The naturalness of the resulting PCA generated motion is often just visually inspected. In Chapter 4.4.2, naturalness is shown by statistical comparison of the various motions.

3.2.2 Creating a Pose Space from Motion Capture Data

This chapter describes the creation of a natural pose space (P-Space) from motion capture data. The term pose space is a specialized version of a natural space (N-Space) that only contains information about pose configurations. Data used for this method is motion capture data which primarily deals with angles for each degree of freedom. Other information can also be added or deduced from the existing data. Additional data may include goals, emotions, and speed of movement. Adding information typically increases the number of dimensions, therefore the data used for learning need to also be increased.

To generate the P-space, frames are read from motion capture data. Each frame is considered to be a singular natural pose. A vector $X=(x_1, x_2, \ldots, x_n)$ is created from the joint rotation angles for each pose. To normalize the data, the rotation and translation angles of the root bone are ignored. The covariance matrix C is calculated, and the PCA transformation matrix T is created based on the eigenvectors calculated with Equation (3-2). Each pose X can be transformed to a corresponding point in N-Space $\alpha=(\alpha_1, \alpha_2, \ldots, \alpha_n)$ using Equation (3-3).

$$\alpha = T \times X \tag{3-3}$$

This point is then transformed to an actual pose by using the inverse of the PCA transformation matrix. The original transformation matrix T is orthonormal, and therefore the inverse of this matrix is simply its transpose. The dimension of the pose space is less than the original dimension. Therefore, only the first n columns of the inverse of the transformation matrix are used. A vector X in the original space can be created from a point in P-space by using Equation (3-4).

$$X = T^T \times \alpha \tag{3-4}$$

Multiple points in the original space may map to the same point in the reduced space. Inverting the point in the reduced space returns a point with the least mean squared-error to all possible points. It is not uncommon that the degrees of freedom may slightly exceed the specified DOF limits. If any of the angles of X is outside the bounds of the joint ($AMin_i...AMax_i$) the angle is adjusted according to the limit of the angles (Equation (3-5)). That is if $X_i > AMax_i$ then $X_i= AMax_i$ and if $X_i< AMin_i$ then $X_i= AMin_i$. The angle error is the difference between the limits and the calculated joint angles.

$$AErr_i = \begin{cases} x_i < AMin_i & AMin_i - X_i \\ x_i > AMax_i & X_i - AMax_i \\ otherwise & 0 \end{cases} \quad (3\text{-}5)$$

3.2.3 Characteristic of Synthetic Data in Natural Space

One important focus of our work is whether the P-space can be utilized to create new natural poses. First, a P-space is created for the right hand motion. The bones involved consists of right hand, right wrist, right humerus, right clavicle, thorax, upper back, lower back, and root. For this data, 98% of the arm variability can be described by only 7 dimensions. A transform matrix T is created from the eigenvectors that transforms the 18 angle vector to a 7 dimensional space. A sample point α consists of 7 values $\alpha=(\alpha_1, \alpha_2, \ldots, \alpha_7)$, each corresponding to one vector element.

The sample poses used for learning the PCA are transformed into their respective P-space coordinates. The minimum and maximum value of each dimensions are retained from these sampled poses. For each dimensions, a set of interval ranging from the minimum value α_{min} to the maximum value α_{max} is created.

A set of new poses are created by selecting points at an interval in P-space. For each dimension, 4 evenly spaced points between the minimum value and the maximum value is chosen. Points contain a combination of values from each interval.

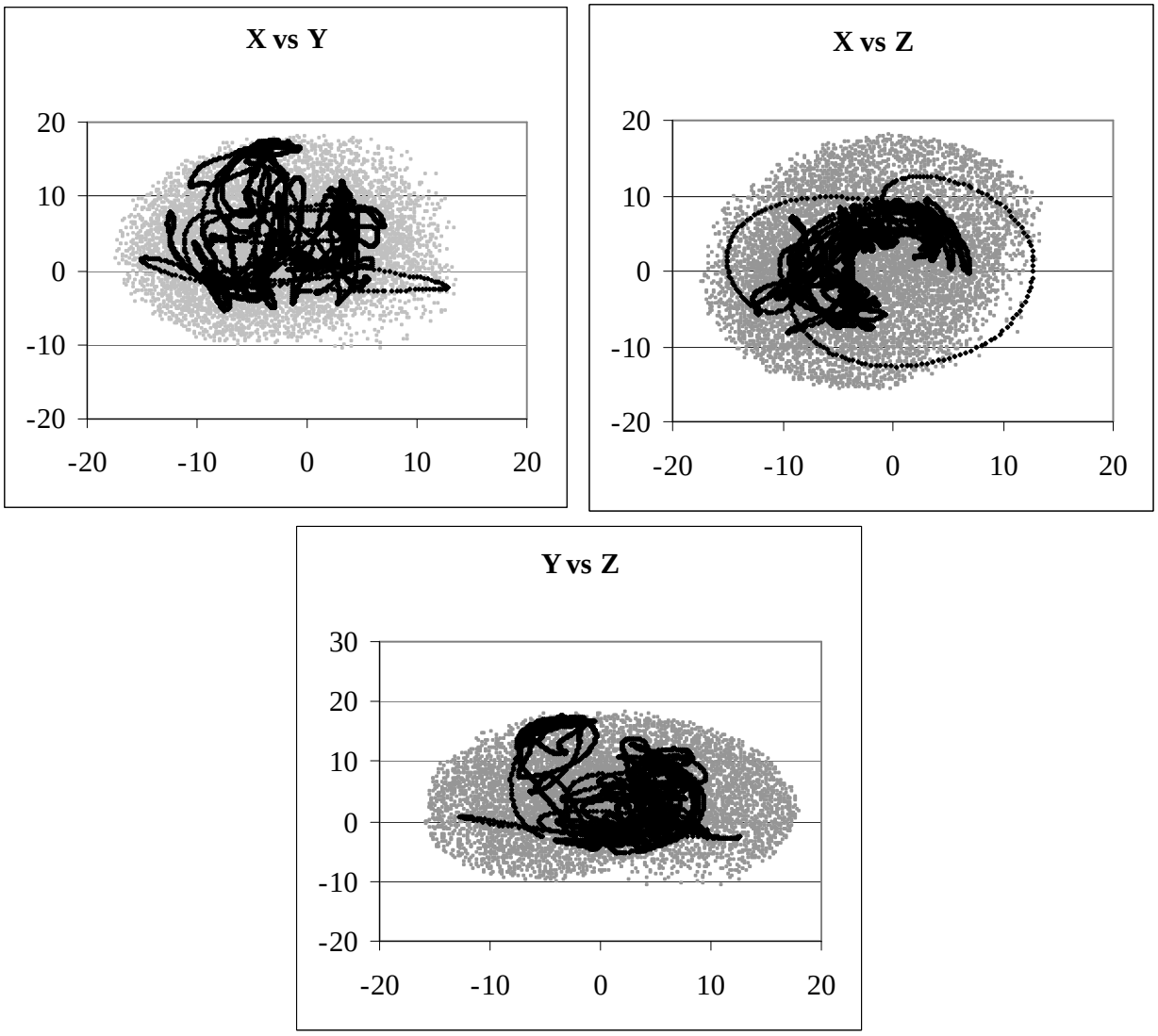

Figure 3-4 The coordinates of the right hand from both the original data (black) and the natural pose samples (grey). The top-left picture is the cutaway x-y coordinate view, the top-right, x-z and the bottom z-y

The created points in P-space correspond to new poses that were not in the original sample. These points are also quite different than the data learned. Figure 3-4 shows the right hand effector coordinates from both the original sample poses as well as the new starting poses. The grey dots represent the right hand coordinates of the new poses, while the black dots

represent the original sample poses. It is apparent that the synthesized poses have a larger range of reach than the original ones.

The naturalness score for each unconstrained point was calculated by calculating the similarity to an existing pose in the database. Comparison was done on a bone by bone basis. Given two poses, one a randomly selected sample point, and a pose from the MOCAP database, the quaternion dot product between each bone angles was computed. The dot product can be used to measure the angle needed to rotate from one angle to another. As the angle becomes more similar, the rotation needed decreases, and the dot product approaches a value of 1. Equation (3-6) shows the similarity measure between two poses, each having k bones,

$$\delta = \sum_{k} (1 - (\alpha_{1k} \cdot \alpha_{2k})) \tag{3-6}$$

We compare this method with two random methods for generating poses. The first method simply selects a set of random angles (within the DOF limits) to generate a pose. The second method creates random numbers using the distribution of the angles in the training data. The method is also compared to actual natural pose taken from the MOCAP database. A random pose is taken from the MOCAP database, the similarity distance is calculated over all poses not in the same motion. The reason for this is that poses in the same motion tend to be close together (especially a pose which comes before or after the reference pose in a motion).

500 poses were generated for each method. The 90^{th} percentile of the distance is calculated. The reason only the 90^{th} percentile of the distance is calculated is that in some poses, the difference of the most different bones significantly dwarfs the values of the other differences. The data was resampled (bootstrapping) 1000 times. The mean angle in radians of the 90^{th}

percentile for each degree of freedom is shown in Figure 3-5. ANOVA was used to calculate that the means are different at a .95 significance.

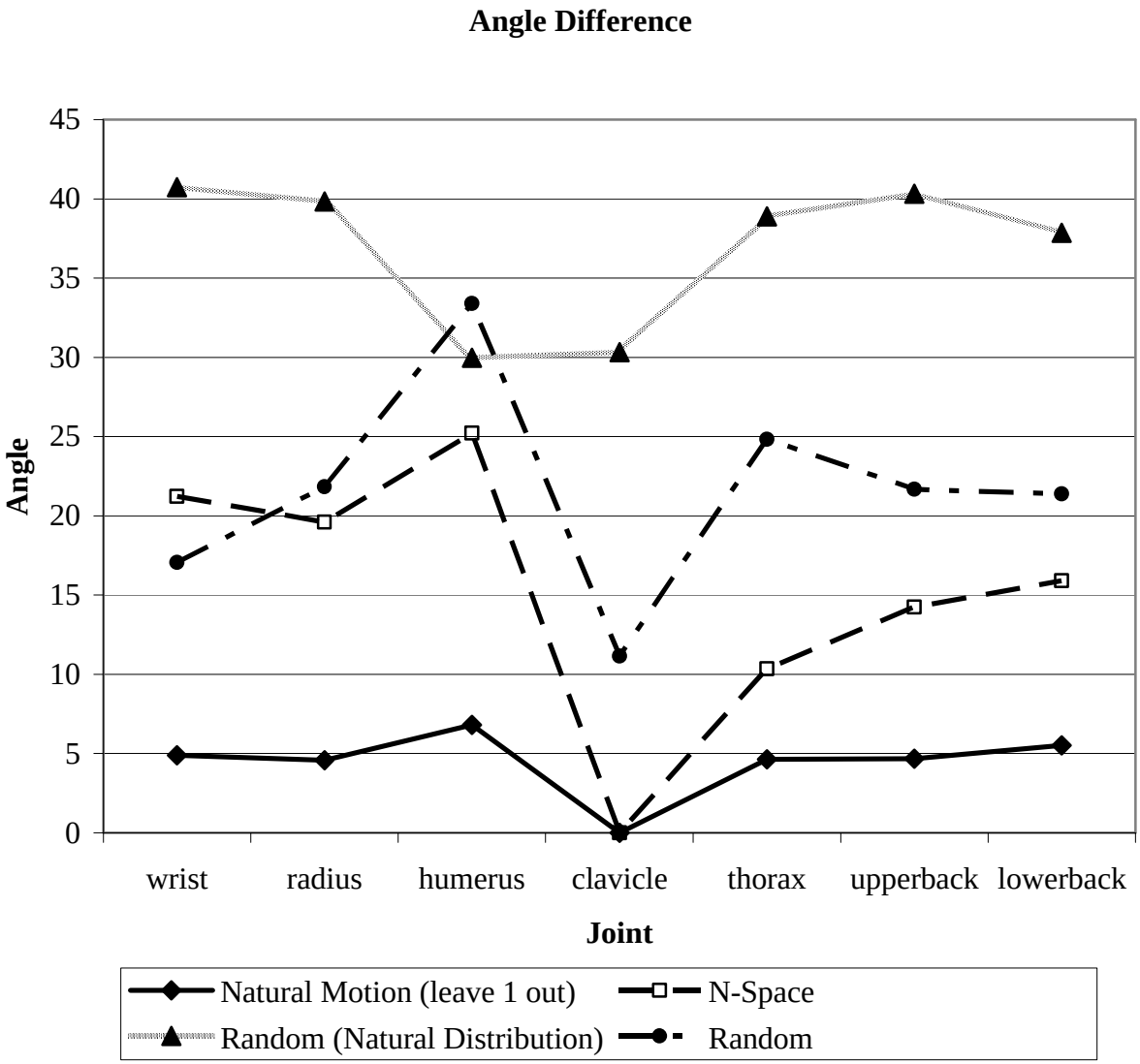

Figure 3-5 Naturalness of random poses generated by the P-space, random (with distribution), random, and from the motion capture data

From the graph above, it is apparent that poses in pose space are more natural compared to the completely random algorithm. Algorithms that are using the P-space have a potential of creating more natural poses than algorithms that use regular space.

3.3 Multiple Natural Spaces

Another factor of concern is the creation of an N-space that is too limited or too specific. The number of data and motion may not be enough to generate an N-space capable of all natural movements. The correlation between body parts that are not near each other may be small, and therefore. Furthermore, the work of Grudzinski concludes that PCA is often better for small sets of joints [81]. Therefore it is a good idea to create multiple N-spaces (one for different parts of the body) based on different parts of the data and then joining them in the animation generation phase. Using a single PCA for the whole body also lead to over generalization of the data. This over generalization reduces the number of possible poses inside the space.

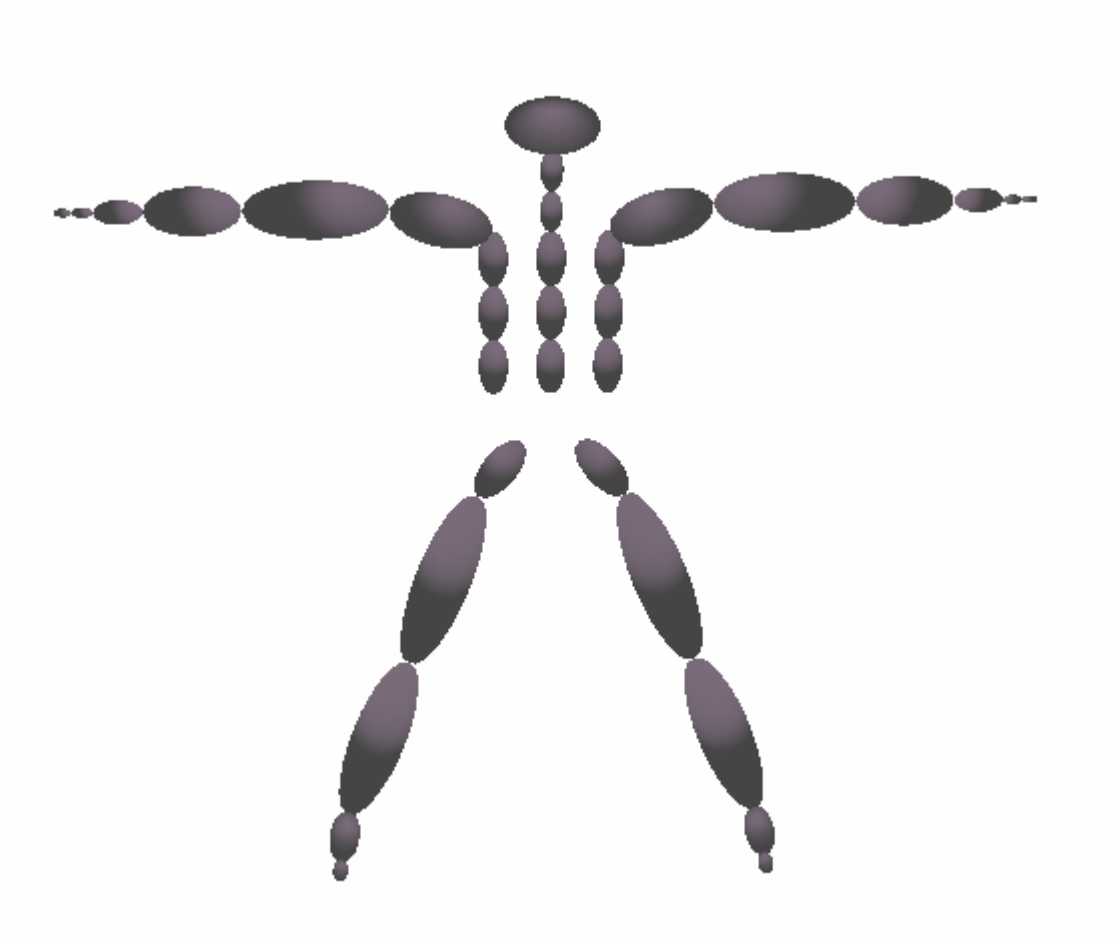

Figure 3-6 Division of the skeleton as shown in Figure 1-1 into multiple segments

In order to alleviate these problems, there is a need to create multiple natural spaces each one corresponding to one particular group of body parts. This gains importance due to the fact that there are multiple constraints for each target pose or motion. The algorithm proposed will combine these various natural spaces in generating the correct path to the pose.

There have been various attempts to divide the human body into parts in order to simplify kinematics and dynamics calculation [57], [64], [82], [83]. The segments created in previous work are free of overlaps. As such, there is a loss of information between the various segments. In order not to lose all the relation between body segments, an overlapping segment scheme is used. In this scheme, some of the joints (such as the lower back, upper back, and thorax) are used in multiple spaces. The upper body part (thorax and back) is important to both the left hand and right hand, and therefore it is impossible to be placed in only one space. Our method divides the body into 5 subsets which have overlapping joints (Figure 3-6, Table 3-1).

Table 3-1 List of bones for each segment

Region	Bones	Number of Dimensions
Right Hand to root	Right hand, right wrist, right radius, right humerus, right clavicle, thorax, upper back, lower back	18
Left Hand to root	Left hand, left wrist, left radius, left humerus, left clavicle, thorax, upper back, lower back	18
Right Foot to root	Right foot, right tibia, right femur, right hip joint	6
Left Foot to root	Left foot, left tibia, left femur, left hip joint	6
Head	Head, upper Neck, lower Neck, thorax, upper back, lower back	18

3.3.1 Effector Space

A problem with segmenting the body into multiple parts as shown above is that the correlation between bones in different part is lost. Another, more general problem is that, in many inverse kinematics problems, the constraint only applies to some body parts while

coordinates of other body parts are not specified. When the right hand is constrained to be placed at a desk for example, where should the other parts of the body lay?

To solve this problem, a space for the coordinates of various body parts is created. This method was also used by Ishigaki et al to compare similarity of a user pose to an example motion for control of avatars [84]. The coordinates of the hands, feet, elbow, neck and head are stored as shown in Figure 3-7. PCA transforms the data into a lower dimensional space. This new space is called the effector space and acts as a guide for positioning the whole body.

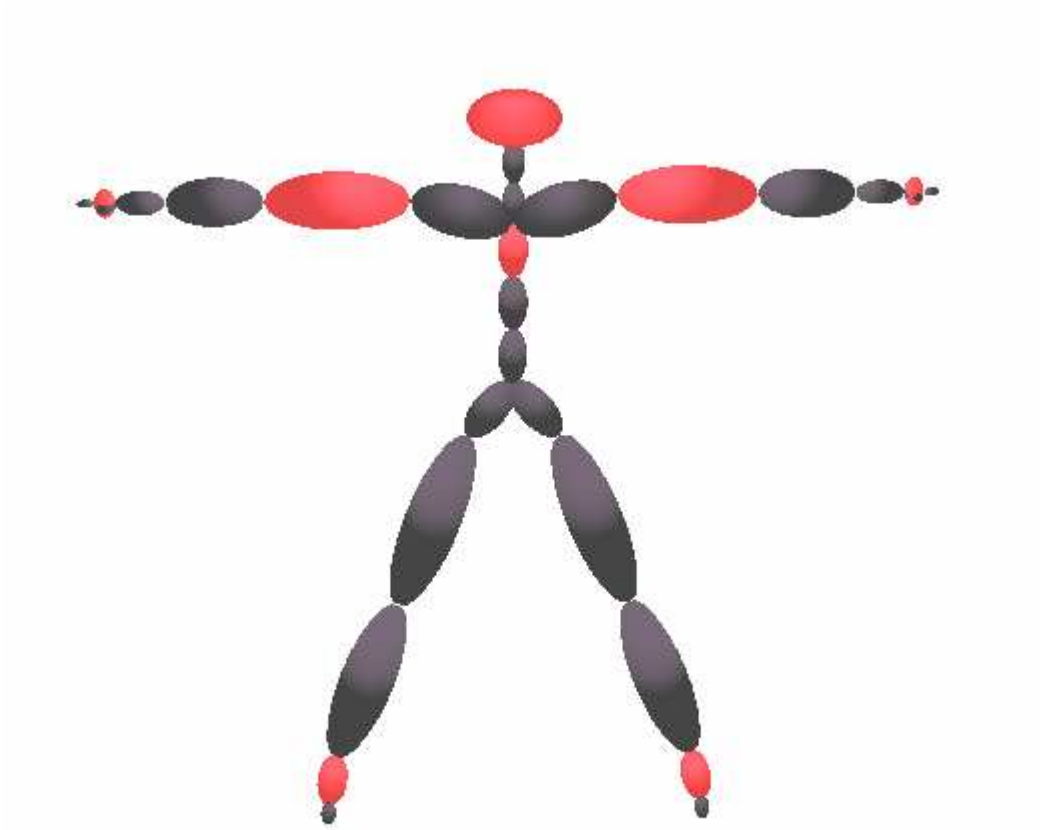

Figure 3-7 Location of effectors for effector space. The coordinates of joints that are colored in red are stored used to create a lower dimensional space.

4 Poses

Poses are the basic building block of motion. One way of thinking about motions is a continuous set of poses, and therefore poses serve as a good starting point for animation. By calculating a pose that meets a certain constraint, one can create motion by interpolating from starting pose to end pose. The constraints of a pose can be coordinates/position of the end of various bones, orientation of bones, or specific areas where the bones must not go through (collision detection). The natural poses that are of interest are those poses that meet the constraints, as well as being as close as possible to the starting pose.

The algorithms described here generate poses from pose spaces as described in the previous chapter. In order to generate natural poses, two strategies are used: search for closest natural pose and inverse kinematics in the pose spaces.

4.1 Lookup strategy

A search on motion capture data can be performed in order to find poses meeting certain constraints. In order to do this reliably, the motion data must be complete, which will likely mean it must be very large. Instead of finding exact matches, it is also possible to find poses that are near to meeting the constraints and then performing minor modifications on the resulting pose.

In order to efficiently perform the search, poses from motion capture data are placed into search optimized data structures such as the oct-tree or Rapidly-expanding Random Tree (RRT) [85]. RRT has been often been used in a planning stage, to find multiple poses (from starting to ending pose) that meet the constraints set out by the environment [63], [64].

The algorithm described here do not use the actual motion capture data. Instead this method first generates a set of poses that is deemed to be representative of the whole possible set of natural poses, referred to as unconstrained poses. Unconstrained poses are simply poses created by selecting points in the pose space. This point is then transformed to an actual pose by using the inverse of the PCA transformation matrix.

The motion capture data utilizes skeletons consist of 29 joints and 59 degrees of freedom. In this work we limit the method to finding poses for the right hand. As we are only concerned about arm pose location, we only utilize the angles from bones that connect the root bone to the right hand bone. The right hand bone acts as the end effector. There are 9 bones and 18 degrees of freedom between the right hand bone and the root bone. In our experiments, the goal is to place the end effector (hand) at a particular, specified, position. The starting pose of the virtual human is all the same. Bones from the hand to the root bone (lower back) are considered; other bones are ignored. The algorithm proposed is called the Constrained Pose Generator (CPG) algorithm.

Sample points were predetermined by choosing points on a grid in P-space. For this data, 98% of the arm data variability can be described by only 7 dimensions. A P-Space was created using 7 dimensions. For each of the seven dimensions, points are sampled on the grid, starting from the minimal value to the maximum value. The lower dimension of PCA captures more variability than the higher dimensions; therefore the lower dimensions were sampled at a higher rate. The total number of points used as a seed for this method is 53000. These points are stored in database, indexed by the end effector position to facilitate fast searching.

The similarity calculation in Equation (4-1) was performed on a more complete motion database. The motion database contained over 3 million poses. For each sampled pose the similarity score is the minimum value calculated using the above formula.

Given a starting pose P_0, our method seeks to determine a set of DOFs that place the end effector (the right hand) at a desired target position T(x, y, and z). We find a sample point R_i having the end effector position S in the pose space which is the best match for that pose. The criteria to find R_i is based on the distance of the end effector with the addition of a weighted naturalness score δ.

$$\arg\min_i \|T_i - S_i\|^2 + w\delta \tag{4-1}$$

Once the algorithm determines a candidate pose that is nearest to achieving the desired constraint, either Coordinate Cyclical Descent (CCD) [35] or the iterative Jacobian [25] is used to refine the pose so as to accurately meet the constraints.

4.1.1 Results

To test the result of this algorithm, 1000 random single constraint problems (on the position of the right hand) was generated to determine accuracy of algorithm as well as naturalness of results. The algorithms tested are the CPG using CCD, CPG using the Jacobian, and Iterative Jacobian algorithm.

Table 4-1 shows how accurate the various algorithms were at finding a solution. Out of the 1000 constraints given, 104 constraints were never found by any of the algorithm. This could mean that the constraints were out of reach range of the virtual human. The two scores in Table 4-1 show the accuracy for all constraints, and accuracy for only the reachable constraints (with a confidence interval of 95%). Based on this table it is clear that in terms of accuracy in finding the

correct pose for a given constraint, usage of P-space is an improvement to using iterative Jacobian.

In some of the poses, the inverse Jacobian algorithm creates poses where the body is twisted and awkward. There are of course exceptions. One of the main problems with the Jacobian is that all joint angles are changed, even though in natural human motion not all joint angles change to achieve a pose.

Table 4-1 Accuracy comparison of various pose generation algorithm

Method	**Accuracy**	**Accuracy for Reachable Constraints**
CPG (CCD)	0.7280 ± 0.0276	0.8125 ± 0.0256
CPG (Jacobian)	0.8090 ± 0.0244	0.9029 ± 0.0194
Jacobian	0.6030 ± 0.0303	0.6730 ± 0.0307

To compare the naturalness of each algorithm, the mean angle difference between each bone in the generated pose and the closest natural pose is calculated. The 90^{th} percentile of the distance is calculated. The reason only the 90^{th} percentile of the distance is calculated is that in some poses, the difference of the most different bones significantly dwarfs the values of the other differences. Therefore we take the biggest 10% difference out of the data. The data was re-sampled (bootstrapping) 1000 times, and ANOVA was used on the data. From ANOVA we find that the mean were different with 95% confidence. The mean angle in radians of the 90^{th} percentile for each degree of freedom is shown in Figure 4-1.

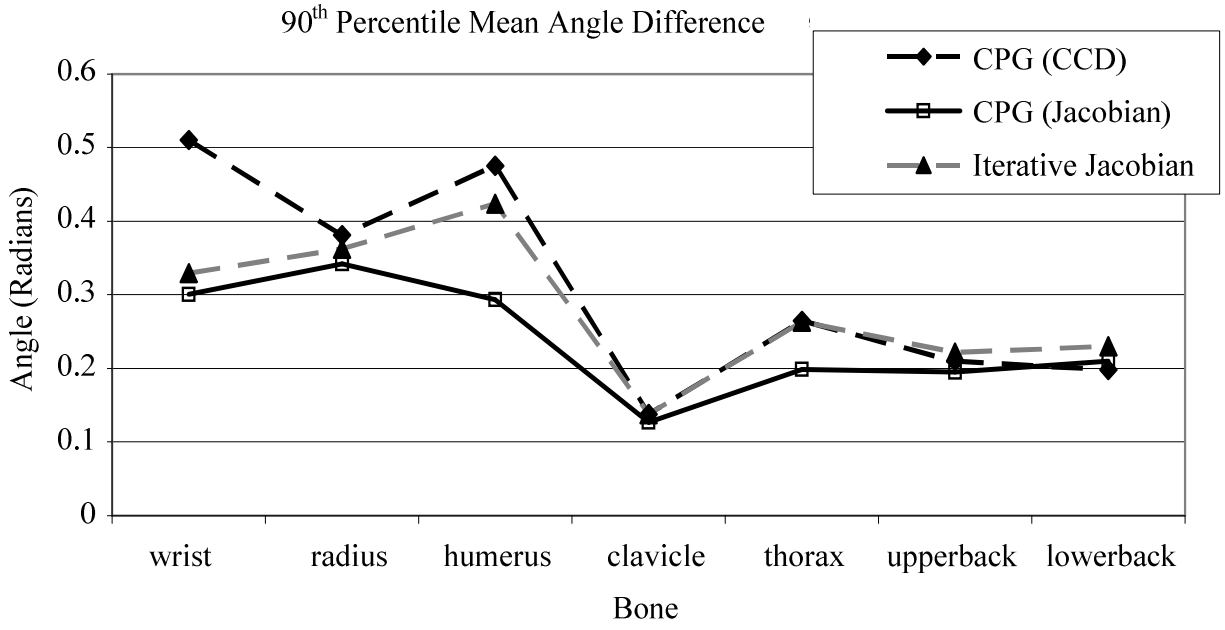

Figure 4-1 Mean angle difference of poses created by CPG (both Jacobian and CCD) and iterative Jacobian method

Based on this graph, we can see that the CPG algorithm created more natural (as shown by a smaller difference to the sampled motion. Because the CPG algorithm moves the starting point, the Jacobian method is able to find poses which are more natural. The resulting poses for the different algorithm on 6 different targets are shown in Figure 4-2.

One of the major disadvantages of this method is that only the generated pose is natural. The modifications made using CCD and Jacobian do not necessarily keep the pose in a natural state. Another problem is that the generated P-Space pose is only based on the distance to meeting the constraints and not on the starting pose. The starting P-Space pose may actually be very far from the starting pose, and thus the ending pose is not always an optimal solution.

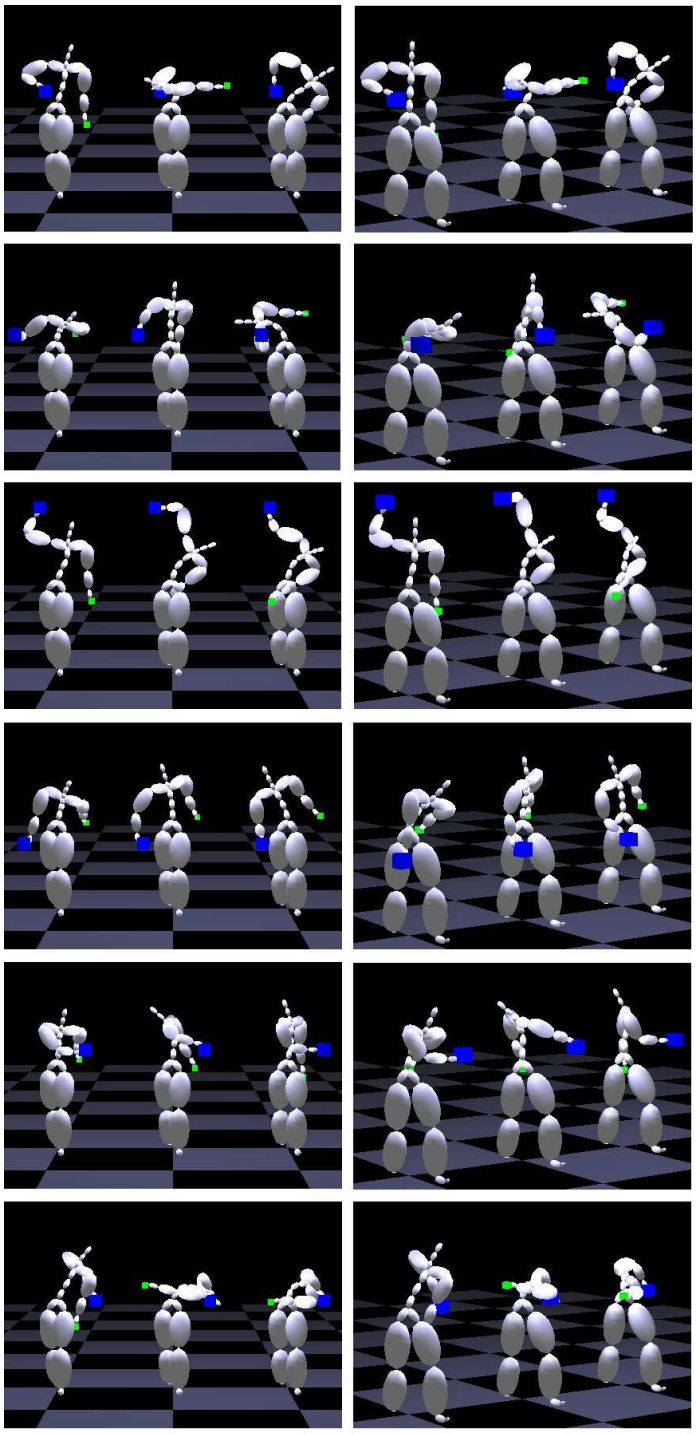

Figure 4-2. Each frame shows the result of running the CPG algorithm for each of 6 poses. The pictures on the left side are the front view of the pose; the pictures of the right side are is from an angled view from the right side of the animated humans. The leftmost pose in each frame was

created using the NSPA algorithm followed by the CCD algorithm, the middle pose was calculated using CCD, and the rightmost pose was calculated using the iterative Jacobian method.

4.2 Inverse Kinematics in Pose Space

In order to address these problems, a method utilizing iteration within the N-Space is used. Iteration through N-Space will also take into account the starting pose. The method proposed here uses the inverse Jacobian in N-space to find pose that meets the constraint. In the previous chapter, CCD was shown to better find natural position from unconstrained N-space poses. However the Jacobian method was preferred for the inverse kinematics because of two things:

1. It is difficult to do multiple constraints with CCD. With Jacobian you just need to add the new degree of freedom to the Jacobian.
2. Segmented needs synchronization between two N-spaces. It is also difficult to do this with CCD. In CCD you change one DOF/dimension at a time. With multiple spaces we may have to change a degree, and then have it changed by the synchronization process.

The first step is to calculate the Jacobian. The Jacobian determines changes in angle effector coordinate(s). Instead of the actual angle however, P-space is used. Therefore the Jacobian calculates the changes in P-space dimension to the effector coordinates. The Jacobian depends on the model used for the skeleton. Our model used a skeleton in which the forward kinematics formula is a series of rotation matrix (R(x)) and translation matrix (L). This corresponds to the bone having a rotation around the x, y, and z axis, followed by a translation (Equation (4-2)). P is a vector (P_x, P_y, P_z) containing the coordinates of the vector.

$$P = \prod_i R_x(x_{i1}) R_y(x_{i2}) R_z(x_{i3}) L \tag{4-2}$$

However because we are using forward kinematics in N-space, this formula changes slightly. To find the value for the n^{th} degree of freedom, Equation (4-3) is used. The angle for the n^{th} degree of freedom is simply the n^{th} row of T^T multiplied by the current point in P-space (Equation (4-4)).

$$X_n = T_n^T \times \alpha \tag{4-3}$$

$$X_n = (T_{n,1}^T \times \alpha_1) + (T_{n,2}^T \times \alpha_2) + \cdots + (T_{n,k}^T \times \alpha_k) \tag{4-4}$$

The forward kinematics in N-space can then be calculated as

$$P = \prod_i R_x(T_{i1}^T \alpha) R_y(T_{i2}^T \alpha) R_z(T_{i3}^T \alpha) L \tag{4-5}$$

The nth value of X is a weighted sum of α according to the n^{th} row of T^T. The derivation of the nth angle of X with regards to the i^{th} value of α is therefore the i^{th} weight of the value, or simply the i^{th} value of the n^{th} row from T^T (Equation (4-6)).

$$\frac{dX_n}{d\alpha_i} = T_{n,i}^T \tag{4-6}$$

$$d(ab) = a'b + ab' \tag{4-7}$$

Using the product rule for derivation (Equation (4-7)), the Jacobian for the i^{th} dimension in P-space and the j^{th} degree of freedom can be calculated. To simplify the derivative equation, let Ψ represent each rotation and translation equation in the forward kinematics calculation of P (Equation (4-8)) such that P is simply the product of all Ψ (4-9). For all rotation factors of P, the derivation of Ψ with regards to α is shown in Equation (4-10). The translation factors of P are constant and therefore the derivations of such factors are 0.

$$\Psi_i = \begin{cases} R(T_i^T \alpha) \\ L_i \end{cases} \tag{4-8}$$

$$P = \prod_i \Psi_i \tag{4-9}$$

$$\frac{d\Psi_i'}{d\alpha_j} = T_{i,j}^T R'(T_i^T \alpha) \tag{4-10}$$

Based on the above equation and the product rule, the partial derivative of the i^{th} coordinate in the pose P with regards to the j^{th} value of α is:

$$\frac{dP_i}{d\alpha_j} = \sum_n \left((\prod_{k=1}^{n-1} \Psi_k) \Psi_n' (\prod_{k=n+1}^{n} \Psi_k) \right) \tag{4-11}$$

The iterative Jacobian method of solving inverse kinematics is to iteratively calculate ΔX that moves the point in P-space to another point that is closer to meeting the target. At each step of the iteration, ΔP, the vector between the current coordinates P_t and the target constraints P_c is calculated (Equation (4-12)).

$$\Delta P = P_c - P_t \tag{4-12}$$

The Jacobian J in P-space is similar to the Jacobian in normal space. The difference is that each element is a partial derivative of P with regards to changes in α. This is shown in Equation (4-13)

$$J = \begin{bmatrix} dP_1/d\alpha_1 & dP_1/d\alpha_2 & \cdots & dP_1/d\alpha_n \\ dP_2/d\alpha_1 & dP_2/d\alpha_2 & & dP_2/d\alpha_n \\ \vdots & & \ddots & \vdots \\ dP_m/d\alpha_1 & dP_m/d\alpha_2 & \cdots & dP_m/d\alpha_n \end{bmatrix} \tag{4-13}$$

The aim is to find Δ α, which is the changes to α that moves the skeleton to the target pose. Δ α can be calculated by using Equation (4-14)

$$J\Delta\alpha = \Delta P \tag{4-14}$$

The Jacobian is not a square matrix, and therefore not invertible. There is no guarantee that the problem is under-constrained or over constrained. To solve the problem, we find Δα that minimizes the distance to the following equation.

$$\|J\Delta\alpha - \Delta P\|^2 \tag{4-15}$$

Furthermore, the angles of the joints should move in small interval. A large Δα may cause the problem to overshoot, or become unnatural. A damping factor is introduced in Equation (4-16). This damping factor tries to minimize the distance of Δα.

$$\left\|T^T \Delta\alpha\right\|^2 \tag{4-16}$$

A simple inverse Kinematics solution for N-Space can therefore be calculated by finding the minimum of the following equation

$$\|J\Delta\alpha - \Delta P\|^2 + \left\|T^T \Delta\alpha\right\|^2 \tag{4-17}$$

If this equation is calculated using Euclidian distance, the minimum distance is equivalent to finding the minimum value Equation (4-18). Because T is orthonormal, $T*T^T$ results in the identity matrix

$$\begin{aligned}&(J\Delta\alpha - \Delta P)^T (J\Delta\alpha - \Delta P) + \Delta\alpha^T TT^T \Delta\alpha \\ &= (J\Delta\alpha)^T J\Delta\alpha - (J\Delta\alpha)^T \Delta P - \Delta P^T J\Delta\alpha + \Delta P^T \Delta P + \Delta\alpha^T \Delta\alpha \\ &= \Delta\alpha^T J^T J\Delta\alpha - \Delta\alpha^T J^T \Delta P - \Delta P^T J\Delta\alpha + \Delta P^T \Delta P + \Delta\alpha^T \Delta\alpha\end{aligned} \tag{4-18}$$

The minimal value of Δα can be calculated by deriving the formula above. The minimal value is found when the derivation of the formula is equals to 0. This results in Equation (4-21) that can be used to find Δα through either the inverse of the left matrix or using factorization such as LU/QR algorithm [86], and therefore solve the inverse kinematics in P-Space. This method is also commonly known as the damped least square method.

$$\begin{aligned}&df/d\Delta\alpha = J^T J\Delta\alpha + (J^T J)^T \Delta\alpha - J^T \Delta P - (\Delta P^T J)^T + 2\Delta\alpha \\ &= 2J^T J\Delta\alpha - 2J^T \Delta P + 2\Delta\alpha\end{aligned} \tag{4-19}$$

$$df/d\Delta\alpha = 0 \tag{4-20}$$

$$2J^T J\Delta\alpha - 2J^T \Delta P + 2\Delta\alpha = 0$$

$$J^T J\Delta\alpha + \Delta\alpha = J^T \Delta P$$

$$(J^T J + I)\Delta\alpha = J^T \Delta P \tag{4-21}$$

4.2.1 Conquering joint limits

One of the problems of this method is that it does not take into account the joint limits. Problems occur when the closest path to a target is via poses that have angles outside the limits. To counter this problem, a perturbation factor that checks for limit breaking angles, and forces these angles to move to the other direction (usually to the middle of the joint limits). If a single iteration contains more than one joint outside the limits, not all are perturbed at the same time. The idea is to perturb one joint and allow other joints to adjust to the perturbed angle and change accordingly.

When the i^{th} joint is outside the limit, this method checks whether perturbation should be performed. Perturbation on the i^{th} joint is performed when the number of steps, and i meets the condition in Equation (4-22). When the modulo of these two numbers to a constant k is equal, then perturbation starts. Once a perturbation starts, it will remain active for w iterations (w is a window size constant).

$$\mathrm{mod}(n\_step, k) == \mathrm{mod}(i, k) \tag{4-22}$$

Equation (4-23) describes this perturbation factor. Δm is the target joint angles, calculated by the difference between the mid-point of the joint limits and the current joint angle. This vector only contains joints currently subject to perturbation. The M factor is a d_p x d_α (P

rows times α columns) transformation matrix that simply deletes all unnecessary factors in the PCA space if the factor does not effect the angle being perturbed.

$$MT^T \Delta\alpha = \Delta m \tag{4-23}$$

To calculate the change in angles/point in P-space, each step of the iteration tries to find a change to α that brings the body closer to the constraints, and at the same time make sure that the intersecting bones are consistent. To solve this, we find a solution for Δα that minimizes the following equation:

$$\lambda_1 \left\| J\Delta\alpha - \Delta P \right\|^2 + \lambda_2 \left\| T^T \Delta\alpha \right\|^2 + \lambda_3 \left\| MT^T \Delta\alpha - \Delta m \right\|^2 \tag{4-24}$$

The equation consists of a weighted sum of three distances. The first distance is based on the difference between the current body pose to the target constraint. The second term is the damping factor, which simply measures the distance between the current pose and the ending pose based on Δα. A damping factor limits the size of Δα, making it change only in small incremental steps. The third factor is the perturbation factor described above. The weights λ is set beforehand to calculate the importance of each factor.

Similar to the simple inverse kinematics equation, the minimum value of this equation can be calculated by finding the derivative and setting it to 0. The solution for Δα can be calculated by solving the following equation:

$$\begin{aligned} &(\lambda_1 J^T J + \lambda_2 I + \lambda_3 T M^T M T^T)\Delta\alpha \\ &= J^T \Delta P + T M^T \Delta m \end{aligned} \tag{4-25}$$

4.2.2 Naturalness

An important part of this research is determining the naturalness of the resulting poses. The most notable method of calculating naturalness from motion capture data has been performed by Ren et al. [82]. They measured naturalness using various methods (Mixture of

Gaussians, Hidden Markov Models, Naïve Bayes, and Switching Linear Dynamic System) and compared the results to human scored naturalness.

Due to time constraints, this thesis does not implement those methods. In this thesis, naturalness is measured by calculating the Gaussian probability density function based on a testing sample. The testing sample consists of 96000 poses taken from the motion capture data. The testing model is assumed to have a Gaussian multivariate distribution with a mean and co-variance N(μ,Σ). The naturalness score for an algorithm is calculated by determining the likelihood that a set of poses were generated by the sample distribution given in Equation (4-26).

$$L(N(\mu,\Sigma)\mid X) = f(X \mid N(\mu,\Sigma)) = \prod_i f(x_i \mid N(\mu,\Sigma)) \tag{4-26}$$

Where f(x| N(μ,Σ)) is simply the Gaussian multivariate probability distribution function, with k being the number of dimensions (Equation (4-27)).

$$f(x_i \mid N(\mu,\Sigma)) = \frac{1}{\sqrt{\pi^k|\Sigma|}} e^{-\frac{1}{2}(x_i-\mu)'\Sigma^{-1}(x_i-\mu)} \tag{4-27}$$

Due to the large size of k, it is common to calculate the average log likelihood in order to compare the various algorithms (Equation (4-28) and Equation (4-29)).

$$\ln(L(N(\mu,\Sigma)\mid X)) = \sum_i \ln(f(x_i \mid N(\mu,\Sigma))) \tag{4-28}$$

$$l = \frac{1}{n}\sum_i \ln(f(x_i \mid N(\mu,\Sigma))) \tag{4-29}$$

The average log likelihood of each algorithm corresponds to the degree of naturalness of the algorithm. An algorithm with a higher average log likelihood score is deemed to be more natural than an algorithm with a lower score. Matlab was used to calculate the log probability distribution function and log likelihood. With Matlab, the minimum positive number is approximately 1x10-300, any value lower than that is considered to be 0. As ln(0) is undefined, a

constant value of -750 was used as the result of ln(0) in order to avoid working with undefined number.

4.2.3 Number of Dimensions

For PCA methods the number of dimensions used has a significant effect on the resulting poses. To find out the effect of the number of dimensions on accuracy and naturalness, we run the single-space PCA algorithm on 100 randomly generated constraints for one constraint problem (the right hand). The original data has 56 dimensions, and testing was performed on 1, 7, 14, 21, 28, 35, 42, 49 and 55 dimensions. The results of the accuracy and naturalness is shown in Figure 4-3 and Table 4-2.

In general, an increase of the number of dimensions increases the accuracy. Accuracy is the percentage of poses that meets all constraints. As the number of dimension increase, there is more freedom in the motion allowing the algorithm to find poses that meet the constraints. Naturalness on the other hand, peaks when the number of dimensions is approximately 50% of the original dimension. Allowing more freedom of movement is detrimental to naturalness, as it allows poses that are not natural to be used.

Table 4-2. The effect of different number of dimensions to naturalness and accuracy

Number of Dimensions	**Average Log Likelihood**	**Accuracy**
1	-750	0
7	-738.8863	0.71
14	-529.2594	0.9
21	-387.8207	0.94
28	-326.3475	0.94
35	-502.4741	0.97
42	-502.2705	0.99
49	-678.8402	1
55	-673.3091	0.99

For the algorithm comparison it is important that the algorithms have a high degree of accuracy and naturalness. We use a threshold value of 95% for the accuracy with as high as possible score for naturalness. This translates to 30 dimensions (from 56 original dimensions) for the single PCA. For the multiple-space PCA, 12 dimensions were used for the right hand, left hand and head space, and 5 dimensions for the right and left foot space. The same number of PCA dimensions is used for all constraint groups.

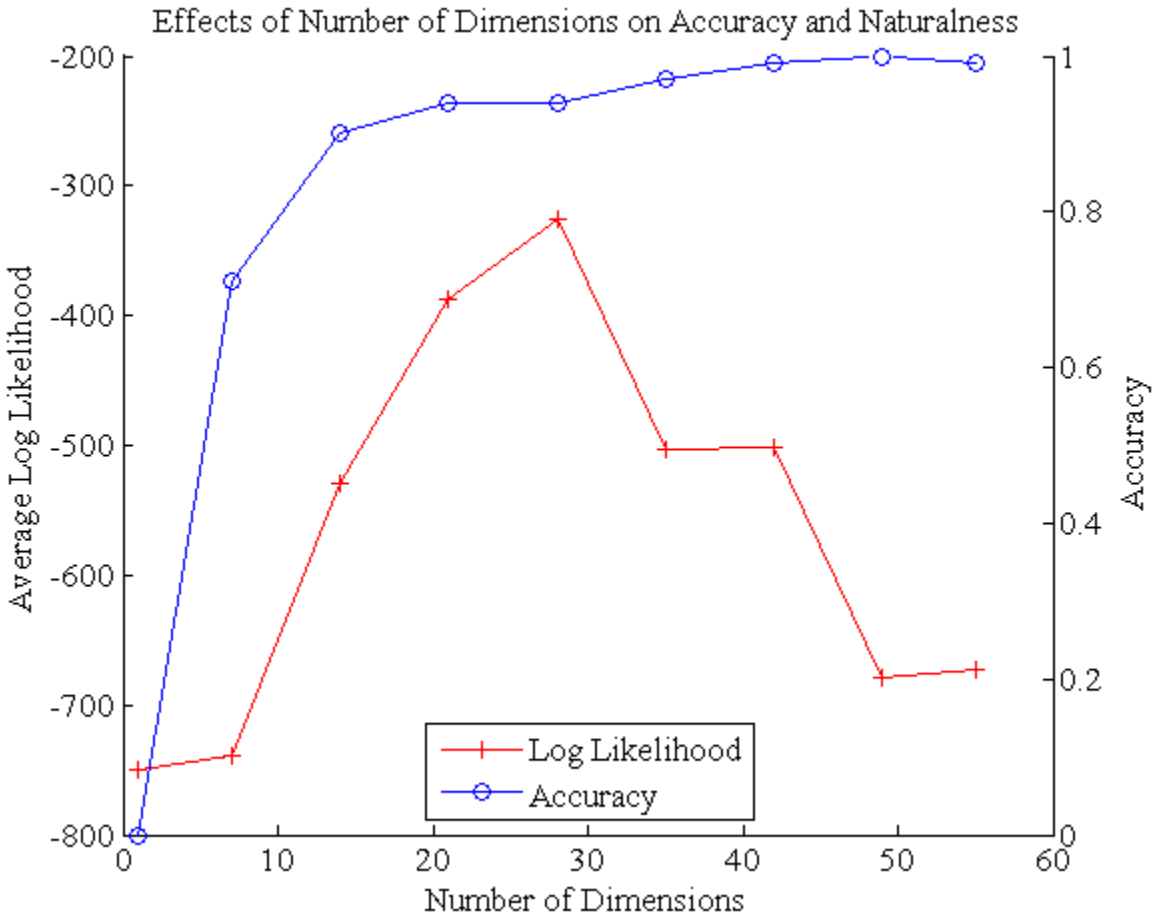

Figure 4-3. Graph of number of dimensions vs accuracy and naturalness

4.3 Multiple PCA Space Inverse Kinematics

It is now a matter of solving a multi-constrained inverse kinematics problem using multiple spaces. There are two ways of solving multi-constrained inverse kinematics problem, using priority [51], [52], or using weights [53]. In this proposed algorithm, a dynamic weighting system is used. In essence, the importance of the estimated effector space constraints decreases at every step while the importance of the external constraints remains the same. This allows the algorithm to satisfy the external constraints in cases where both constraints cannot be satisfied at the same time.

For each kinematics step the current target position ΔP is a matrix that consists of the external constraints P_x and the effector space constraints P_e. Section 4.3.1 describes how to calculate both the external constraints and the effector space constraints. The external constraints are capped with a distance of m_d (Equation (4-30)) while the effector constraints are capped with a variable distance that depends on m_d and the step number n (Equation (4-31)). If the resulting calculated distance for the effectors space is lower than a constant k_d then that constraint is ignored. This reduces the number of constraints that the calculation must meet and helps in finding the solution faster. This also has the effect that as the number of steps increases, the inverse kinematics solution prefers to stay at its current position rather than trying to move to the exact position of the effectors constraint

$$d_{tx}(P) = \min\left(\frac{m_d}{\|P\|^2}, 1\right) \tag{4-30}$$

$$d_{te}(P) = \min\left(\frac{m_d}{\sqrt{n}\|P\|^2}, 1\right) \tag{4-31}$$

$$\Delta P = \begin{bmatrix} d_{tx}(P_{tx})P_{tx} \\ \forall_{P_{te}, d_{te}(P_{te})>k_d} d_{te}(P_{te})P_{te} \end{bmatrix} \quad (4\text{-}32)$$

4.3.1 Estimated Effectors Space

The effectors space described in chapter 3.3.1 is implemented and utilized in this method. If P is a vector of coordinates of the various body parts, T_E is a PCA transformation matrix to a lower dimension based on learning the coordinates of the 8 effectors positions from the motion capture data. β is the resulting point in the reduced dimensional space according to Equation (4-33). The reduced space is notated as CC-space (constraint coordinate space)

$$\beta = T_E P \quad (4\text{-}33)$$

There are two groups of constraints, the external constraints C_x which are specified by the user or by the actions, and the new set of constraints, called the estimated constraints C_e which constraint the rest of the body parts. The estimated constraint C_e, is calculated based on the current position as well as the target constraint.

Given that β_0 is a point in CC-space corresponding to the current pose, Δβ is the difference in CC-space that corresponds to a point that meets the external constraints C_x. Γ is a matrix that finds the external constraint elements from a pose. To calculate Δβ, the following equation is minimized:

$$\left\| \Gamma T_E^T \Delta\beta - (\Gamma P_{t+1} - \Gamma P_t) \right\|^2 + \left\| \Delta\beta \right\|^2 \quad (4\text{-}34)$$

Equation (4-34) is minimized by solving the following equation:

$$(T_E \Gamma^T \Gamma T_E^T + I)\Delta\beta = T_E \Gamma^T (\Gamma P_{t+1} - \Gamma P_t) \quad (4\text{-}35)$$

The target position P_t, consisting of both the external and the estimated constraints is calculated using Equation (4-36). This value is used in the calculation for finding $\Delta\alpha$ such as in Equation (4-25).

$$T_E^T(\Delta\beta + \beta_0) = P_t \tag{4-36}$$

4.3.2 Matching overlapping angles

The biggest problem with creating segments that overlap is that some of the angles are in multiple segments. When calculating the inverse kinematics solution for each segment, angles that are in multiple segments must result in the same value.

The first attempt to solve this involves creating additional factors to control the angle difference. One angle may appear in two segments k and l. This angle appears as the i^{th} angle of the k^{th} segment and the j^{th} angle of the l^{th} segment (Equation (4-37)).

$$x_{i,k} = x_{j,l} \tag{4-37}$$

The angles from the above equation can be calculated from the points in P-space through:

$$\begin{aligned} T_{i,k}^T\alpha_k &= T_{j,l}^T\alpha_l \\ T_{i,k}^T\alpha_k - T_{j,l}^T\alpha_l &= 0 \\ \begin{bmatrix} T_{i,k} & -T_{j,l} \end{bmatrix} &\begin{bmatrix} \alpha_k \\ \alpha_l \end{bmatrix} = 0 \end{aligned} \tag{4-38}$$

For multiple angles, a matrix can be created for all angles that must be matched. A new matrix Γ combines the subtraction formula for multiple angles, one formula per row. The above equation is equivalent to:

$$\Gamma\begin{bmatrix} \alpha_k \\ \alpha_l \end{bmatrix} = 0 \tag{4-39}$$

The above equation can be added as a factor to the Equation (4-24). However, experiments show that this formulation decreases the accuracy of the solution. This significantly adds to the number of constraints in the system. The iterations using this formula often end up at a local minimum that does not sufficiently address the main constraints.

4.3.3 Weighting Overlapping Angles

A second attempt to solve the overlapping problem is through weighting after the calculation. Once the change in α is calculated, the overlapping angles is consolidated. Consolidation is done through calculating the weighted sum of all angle spaces. Equation (4-40) describes the calculation for the k^{th} joint angle, based on the weight of the i^{th} space (w_i), and the k^{th} angle calculated by the i^{th} space.

$$\theta_k = \frac{\sum_i w_i \theta_{i,k}}{\sum_i w_i} \qquad (4\text{-}40)$$

The weights used are based on whether the i^{th} space corresponds to an active constraint or an effector space constraint. If the space affects an active constraint, it is given a priority weight of 1.0 or a non-priority weight 0.8. Only one space can have a priority weight of 1.0 at one time, therefore the space that has the priority weighting is changed every n steps (we use 10 steps). Passive constraints or effector space constraints is given a weight of 0.4. All the constants here were found using empirical methods. This weighting scheme gives more importance to the active constraint spaces and allows the solution to be found much quicker.

4.4 Results

Comparisons were performed on a different number of constraints. There are five groups of constraints that are used. All of the constraints were the Cartesian coordinates of the various body parts. For each group, 100 randomly generated set of constraints were created. The groups are:

1. Right hand constraint only
2. Right foot constraint only
3. Right and left hand
4. Right and left foot
5. Right hand, left hand, right foot, and left foot

4.4.1 Accuracy

The first measure for the algorithm is the accuracy. The accuracy is simply the number of poses found that meets all constraints. The algorithm must find a pose that meet all constraints in 250 iterations.

Table 4-3 and Figure 4-4 shows the accuracy of the various algorithms in finding poses that meet the constraints. An increase in the number of constraints tends to lower the accuracy of the various algorithms. This may be because it is impossible to achieve a pose that meets all constraints. Another explanation is that some of the algorithms simply iterates on a local minima during the Jacobian iteration. When the iteration count is higher than the maximum allowed, the application stops the algorithm and marks it as unable to fulfill the constraints.

Table 4-3. Accuracy comparison of all algorithms

	R. Hand Constraints	**R. Foot Constraints**	**R + L hand constraints**	**R + L foot constraints**	**R+L Hand + R+L Foot constraints**
Single PCA	0.95	0.68	0.79	0.61	0.32
Single PCA+	0.96	0.96	0.63	0.91	0.37
Regular Jacobian	0.91	0.98	0.28	0.82	0.14
Multiple space PCA	0.98	1.00	0.69	1.00	0.65

The foot constraints problem causes a lot of difficulty. A look at the right foot constraints problem shows that the single PCA algorithm is unable to find the goal because in trying to find the shortest path to a goal, typically tries to bend the knee outward (past the bounds of the angles).

The Multiple space PCA method has a very high accuracy compared to the other methods. The only exception to this is for the right and left hand constraint group where the Multiple spaces PCA does worse than the Single PCA method. A possible reason for this is that the right and left hand constraints cause a conflict in the overlapping joints of the spaces. As such, the algorithm finds a pose where the error is minimized, which is to say the total distance from the end effector position to the constraints is the smallest. However this pose does not actually meet any of the constraints.

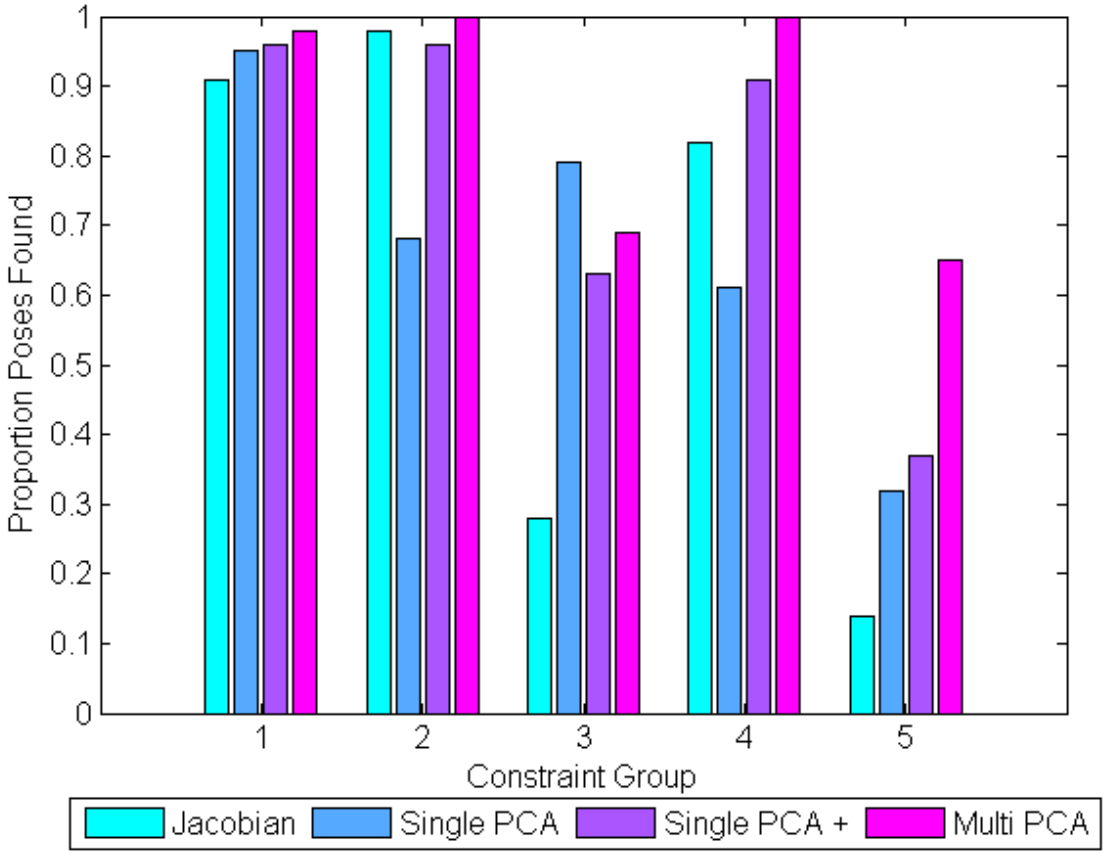

Figure 4-4 . Graph of accuracy of all algorithms

4.4.2 Naturalness

Table 4-4. Average log likelihood of all algorithms

	R. Hand Constraints	**R. Foot Constraints**	**R + L hand constraints**	**R + L foot constraints**	**R+L Hand + R+L Foot constraints**
Regular Jacobian	-712.4789	-141.356	-721.299	-219.605	-696.894
Single PCA	-339.3964	-169.534	-577.728	-298.049	-580.832
Single PCA+	-315.1096	-234.453	-445.176	-363.788	-474.114
Multiple space PCA	-264.8083	-133.141	-305.194	-139.147	-348.928

The naturalness of each pose created by the algorithm is measured by Equation (4-29). To analyze this data further, bootstrapping was used to find the mean and the variance. 1000

pose groups were created for each algorithm. Each pose group consists of n poses, where n is the number of correct poses found by the algorithm. The poses in the pose group is found by randomly sampling with substitution from the correct poses. We calculate the average log likelihood for each group and then calculate the mean and standard deviation (Table 4-4 and Figure 4-5).

The regular Jacobian method on average results in less natural poses for the hand constraints and the hands and feet constraints. However the poses for the feet constraints are more natural than the resulting poses from the single PCA algorithms. For the feet constraint groups we find that the regular Jacobian method does not move the upper body at all. Because the upper body remains in the starting pose (which is a natural pose), and given that the number of non moving joints is much higher, the probability of this pose being a natural pose is also higher.

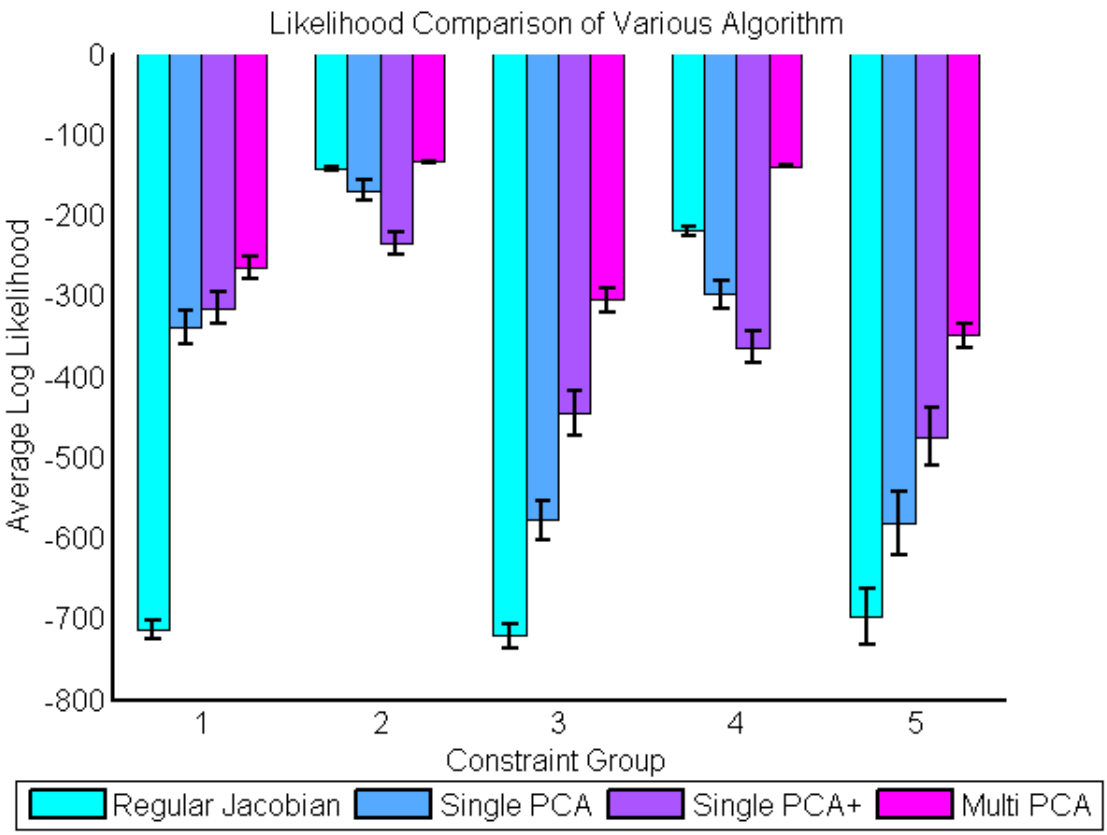

Figure 4-5. Average log likelihood of all algorithms graph

For all constraint groups, we can see that the multiple space PCA performs better than the single PCA or even the single PCA with effector space and perturbation. One reason why single space PCA creates unnatural poses is that all joints are correlated in this algorithm, therefore moving one part a certain angle, also moves another part that is deemed to be related to it. Moving the arms is correlated to movement in the legs, and the more the arm joint moves, the more the leg joints move. Figure 4-6 shows how the legs in figures created using the single space PCA method is unnatural due to the fact that they are moved in relation to the hands movement. In these figures, only the position of the right hand is a hard constraint.

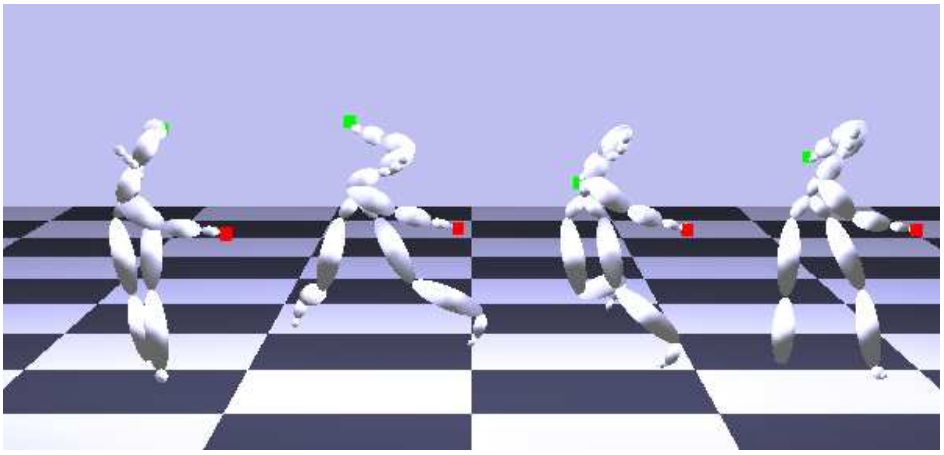

Figure 4-6. Unnaturalness is caused by extreme leg movement in single space PCA. The figures were created by (from left to right) regular Jacobian, single space PCA, single space PCA +, multiple space PCA

The second reason for unnatural poses is the proclivity to bend to a target when the natural thing is to turn the body towards a target. Bending is preferred because it is usually the closer solution in the pose space. However, as we can see in Figure 4-7, bending the body to a particular target sometimes results in unnatural poses.

Because the correct poses created by each algorithm differ, the naturalness score may be skewed. To compare the algorithms more fairly, pair wise T-test were performed on pairs of algorithms. The list of tests consisted of the single space (Single PCA) versus single space + effector space + perturbation algorithm (Single PCA+), single PCA vs. multiple spaces, single PCA + vs. multiple space, and Jacobian vs. multiple spaces. Only figures that meet constraints for both algorithms were included in this calculation. The paired log likelihood score for these algorithms are shown in Table 4-5, Table 4-6, Table 4-7, Table 4-8, and Figure 4-8. Again, bootstrapping was used on the paired data in order to retrieve the best estimate for mean and standard deviation. T-test results suggest that the difference between all of the pairings are significant ($p<0.01$)

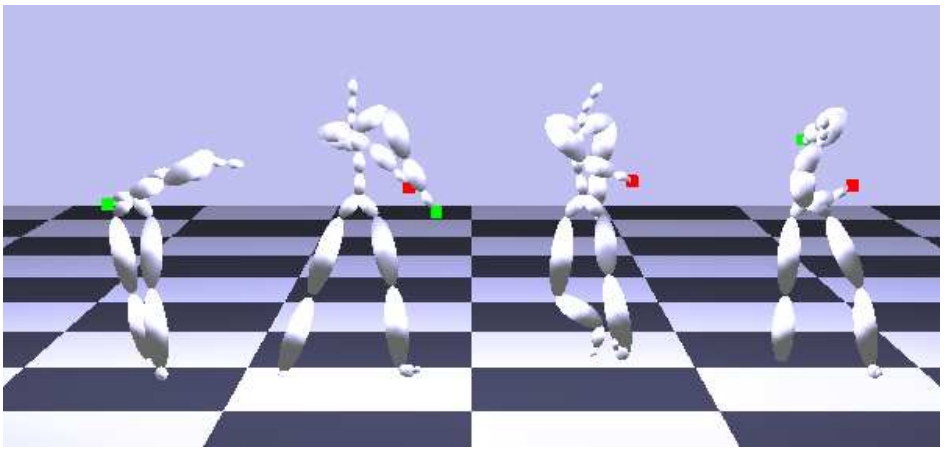

Figure 4-7. Unnaturalness is caused by extreme leg movement in single space PCA. The figures were created by (from left to right) regular Jacobian, single space PCA, single space PCA +, and multiple space PCA

Table 4-5. Average log likelihood comparison of the Inverse Jacobian and Multi PCA algorithm

	R. Hand Constraints	**R. Foot Constraints**	**R + L hand constraints**	**R + L foot constraints**	**R+L Hand + R+L Foot constraints**
Jacobian	-711.811	-719.07	-689.783	-717.499	-750.00
Multi PCA	-253.033	-269.747	-304.883	-273.159	-307.586

As expected, the regular Jacobian creates poses which are the most unnatural. For poses having the same constraint, the multiple-space PCA algorithm creates poses that are significantly more natural.

Table 4-6. Average log likelihood comparison of the Single PCA and Single PCA + algorithm

	R. Hand Constraints	**R. Foot Constraints**	**R + L hand constraints**	**R + L foot constraints**	**R+L Hand + R+L Foot constraints**
Single PCA	-323.0806	-340.911	-352.788	-369.725	-406.215
Single PCA+	-296.2438	-303.793	-333.184	-334.931	-388.236

The single space PCA and single space PCA+ algorithm's average likelihood do not differ as dramatically as other pairings. Adding the perturbation factor and a set of effector space constraints to the single space PCA algorithm helps control the location of all the non-constrained body parts, and thus the single space PCA+ algorithm creates more natural poses.

Table 4-7. Average log likelihood comparison of the Single PCA and Multi PCA algorithm

	R. Hand Constraints	**R. Foot Constraints**	**R + L hand constraints**	**R + L foot constraints**	**R+L Hand + R+L Foot constraints**
Single PCA	-330.6858	-339.53	-373.038	-369.51	-347.944
Multi PCA	-256.6454	-265.001	-287.241	-279.14	-260.00

Table 4-8. Average log likelihood comparison of the Single PCA+ and Multi PCA algorithm

	R. Hand Constraints	**R. Foot Constraints**	**R + L hand constraints**	**R + L foot constraints**	**R+L Hand + R+L Foot constraints**
Single PCA +	-307.2701	-325.293	-334.842	-332.925	-321.276
Multi PCA	-256.5242	-266.892	-280.256	-269.738	-263.291

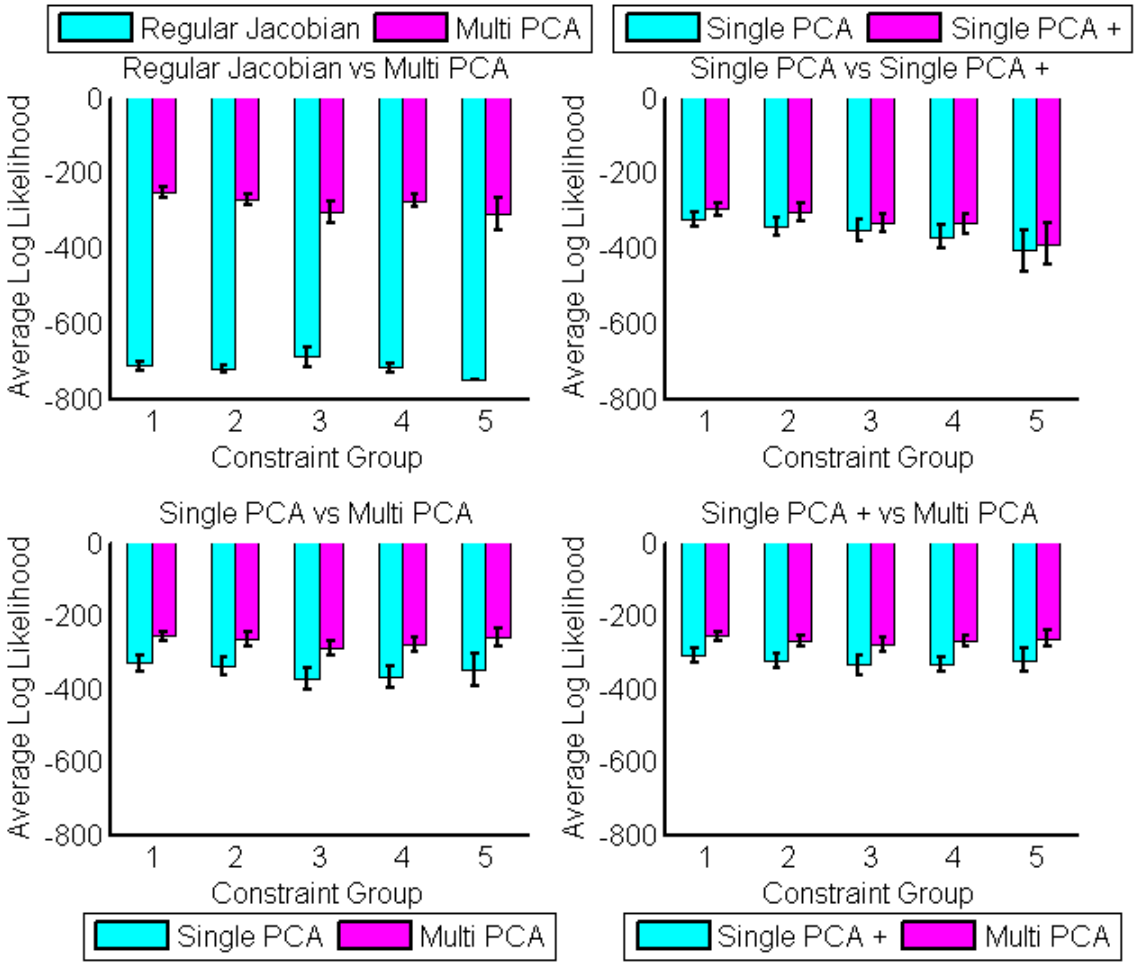

Figure 4-8. Graph of pairwise comparison between algorithms

Table 4-7 and Table 4-8 shows that the multiple space PCA method creates poses that are more natural than both the single space PCA and single space PCA+ algorithms. A graph of these values is shown in Figure 4-8.

Figure 4-9 shows a small sample of results of the algorithms for 2 constraint problems (hands). For each pose group the figures show the resulting pose calculated by the regular Jacobian, single space PCA, single space PCA+, and multiple space PCA method. A more detailed set of samples for different constraint groups can be seen in Appendix A.

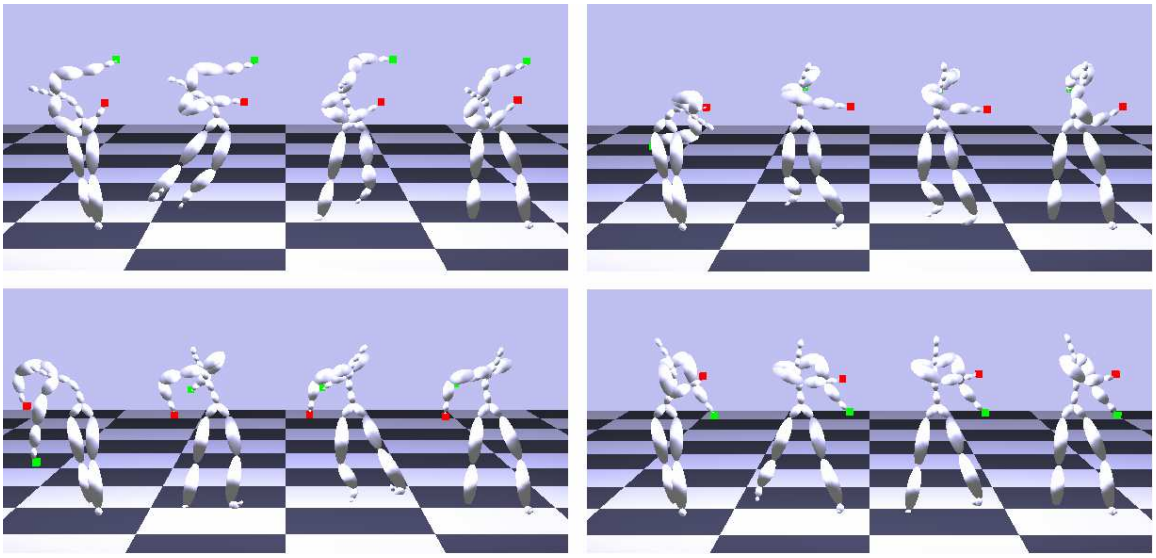

Figure 4-9 Sample result poses for 2 constraints (hands) problems

5 Motion

Motion is a set of frames presented sequentially a such a rate of speed that the illusion of smooth movement is created. To create motion automatically, an algorithm creates a number of frames containing the poses, from the starting pose to an ending pose that meets the target.

The algorithm described here finds a motion in motion space (M-space) to move a virtual human from the starting pose to the ending pose that meets constraints. The M-space is a specialized version of a natural space that contains data regarding the phase, which is the pose configuration and angular speed for each frame. The ending pose must be predefined through other algorithms (e.g. the P-space inverse kinematics algorithm described in Chapter 4.3). The M-space algorithm uses the damped least squares method [87], [88] to find a set of frames that allows a natural movement between the starting pose and the ending pose.

5.1 Characteristic of natural motion

The simplest method for creating motion between P_0 and P_t is simple linear interpolation. Linear interpolation assumes the velocity at each time frame is the same, and that the path from P_0 to P_n is a straight path. To perform this type of interpolation, the first thing to do is calculate the interpolation step ΔP, which is simply the vector from P_0 to P_n divided by the number of frames (Equation (5-1)).

Each step of the motion can be calculated by using the pose for the previous frame and adding the interpolation step (Equation (5-2)). This step is repeated until the final pose is reached.

$$\Delta P = \frac{P_n - P_0}{n} \qquad (5\text{-}1)$$

$$P_{t+1} = P_t + \Delta P \tag{5-2}$$

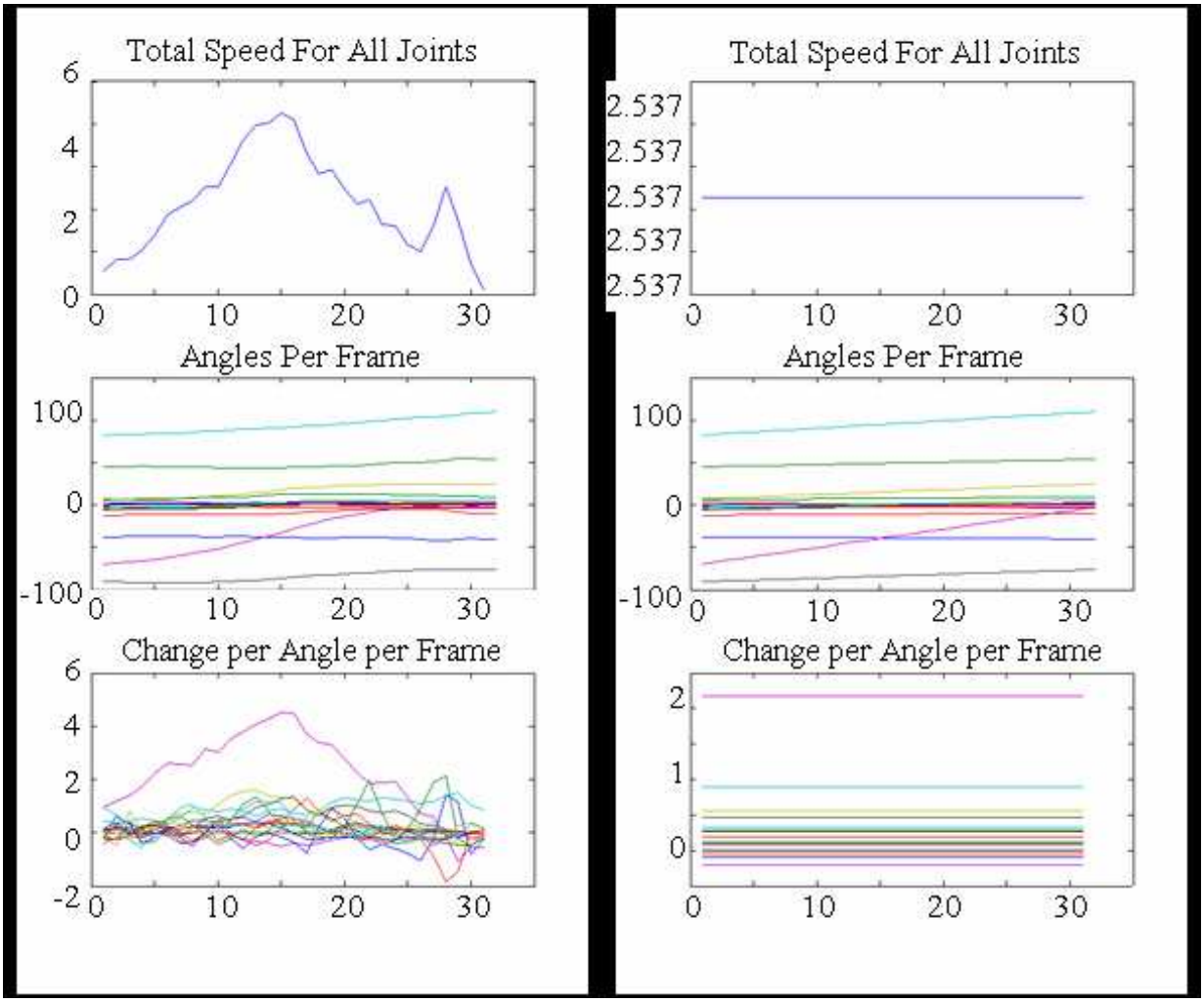

Figure 5-1 The speed and angles of natural motion and a linear interpolated motion

However, observation of captured motion data reveals that natural motion do not look like the results of linear interpolation. Natural motions do not have a constant speed for all angles. The angles for each joint do not always move with the same speed, some angles may start slow and speed up in the middle before slowing down again near the end. Joint angles sometimes deviates from the straight path between the start and end angles.

An example of the speed of joint angles in a natural motion created by the right hand segment, with a length of 32 frames is shown in Figure 5-1. The left part of the figure shows the

natural speed, the actual angles in degrees for each degree of freedom, and the speed of each degree of freedom. The right part shows the same attributes created by linear interpolation between the starting pose and the ending pose. The same non linear motion characteristics have also been observed in faces deformation [89].

5.2 Utilizing a Motion Space

Chapter 2 discusses the various techniques used to create motion. This thesis focuses on how to use existing motion capture data to generate motions. There are two approaches to creating human motion assuming a constrained target. The first is to generate a motion that finds a pose at the end that meets the constraints. The other method finds a set of key frames that meet the constraints first, and then interpolate the rest of the motion. The second approach is the approach used in this thesis.

The creation of synthesized motions from motion data, have been proposed by various authors. Abe et al. creates a limited amount of poses which have the same physical requirements using transformation (by rotation or translation) of motion capture data [32]. Grochow et al [90] compute poses via kinematics based on data learnt from captured motion. They calculate the likelihood of a particular pose using a probabilistic model. New poses are synthesized using an optimization algorithm in which the objective function depends on the learnt poses. Similarly, in Yamane [63], inverse kinematics is calculated using a constrained optimization algorithm. Captured data is stored and used as soft constraints. Motion is created by the smoothing of various results of IK computations over multiple positions.

The algorithm proposed by Pan [64] uses RRT to plan out the key frames necessary. For each set of key frames, a motion is calculated using interpolation, and depending on how natural it is (and whether it is in a constrained environment or not), it is replaced by existing motion

capture data. They first find a motion capture data segment that resembles the motion they have, and then uses that. This still requires a large database of motion capture data to be able to create good results.

PCA have been used on whole motions (normalized skeletal configuration across multiple frames) [79], [80]. The created PCA space is then used to create new motions with different constraints. In order to do this, only similar motions, or motions which performs one task can be summarized using the PCA, for example golf motions.

Instead of a using each motion as one data point, the algorithm described here uses the phase (a combination of pose configuration and angular velocity) to create a reduced motion space (M-space). Given the starting pose P_0, the ending pose P_n, and the number of frames n, the proposed method attempts to find a natural motion (through generation of the interleaving set of poses) that moves the human from the start pose to the end pose. The poses are found by finding the points in a lower dimension motion space that meets motion constraints.

The algorithm focuses on finding the shortest path between the starting pose and ending pose. Realistic physics may sometimes be overlooked. For example, if position A was a standing position and position B was a kicking position (where the leg meets the ball), it is unlikely that the algorithm will try to move the leg back to gain momentum for kicking. Instead the end results will probably be a straight path between the starting positions to the kick position. This can be remedied by having a constraint that the motion includes a backward movement to create momentum.

5.2.1 The Motion Space

To create a motion space, information about the current angle as well as the angular velocity for each DOF is used. To calculate the angular velocity ω_t, the average difference between the previous frame and the next frame is used in order to smooth out the result (Equation (5-3)).

$$\omega_t = \frac{P_{t+1} - P_{t-1}}{2} \tag{5-3}$$

The vector H, represents a phase of motion contains both the current angle configuration as well as the angle velocity at each frame (Equation (5-4)). The numbers of dimensions for H is double that of the original configuration space. PCA is used to create a lower dimension space (the motion space or M-space). The number of data used for creating the PCA space was increased to 45000 samples.

$$H_t = [P_t, \omega_t] \tag{5-4}$$

N is the point in M-space corresponding to H. T is the transformation matrix from the original phase space to the M-space that transforms H to N (Equation (5-5)). T is orthornormal, and therefore the transpose of T transforms from N to H (Equation (5-6)).

$$N_t = TH_t \tag{5-5}$$

$$H_t = T^T N_t \tag{5-6}$$

Each point in N-space corresponds to a pose and a velocity. T^T_p and T^T_v are transformation matrices that transforms from the point in M-space to the pose (Equation (5-7)) and velocity (Equation (5-8)) respectively.

$$P = T_P^T N \tag{5-7}$$

$$\omega = T_v^T N \tag{5-8}$$

5.2.2 Calculating Motion

Motion is calculated in the M-space. N_0 and N_n is the point in M-space corresponding of the phases H_0 and H_n. We attempt to find the motion M which is a contiguous set of points in M-space corresponding to a set of poses in the original space. Figure 5-2 illustrates the M vector that includes the data for every frame.

$M = [N_0\ N_1\ N_2 \ldots\ N_n]$

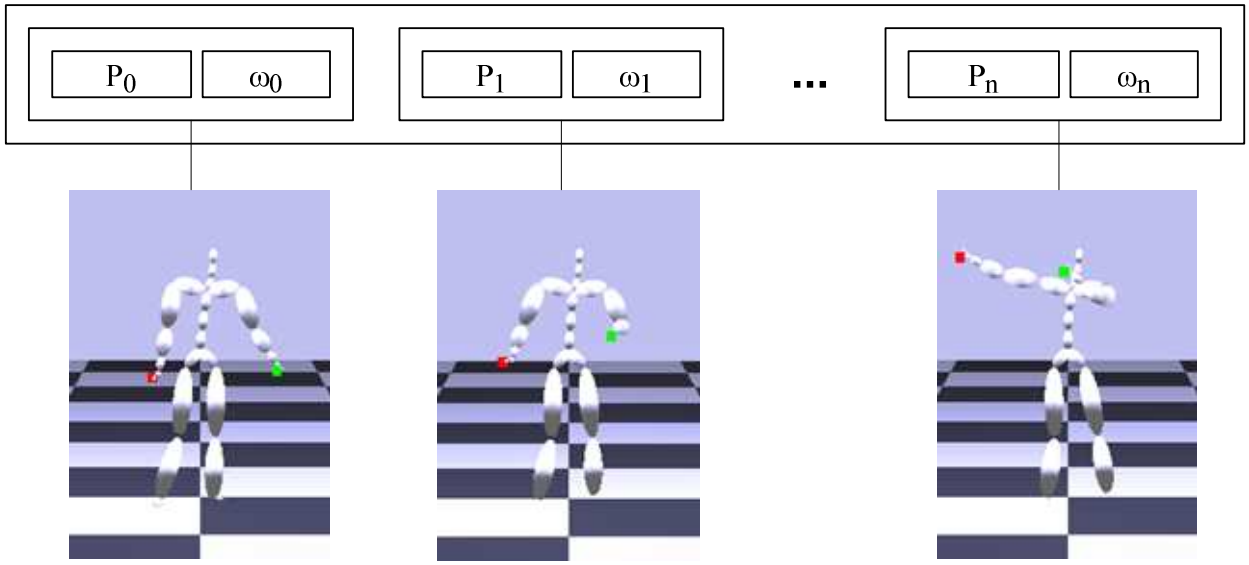

Figure 5-2 Motion vector representation used in this thesis

To calculate M using N-Space, these constraints are used:

1. $N_0 = [P_0\ \omega_0]$. The first element of M is a PCA transformation of P_0 and the initial speed
2. $N_n = [P_n\ \omega_n]$. The n^{th} frame element of M is a PCA transformation of P_n and the final angular speed
3. $P_{t+1} = P_t + k\ \omega_t$. The poses in the motion must be contiguous. The momentum added to each pose must match the following pose (possibly scaled by a constant k).

4. ω_0 and ω_{n-1} are minimized. The assumption here is that the motion starts and end on a still pose. Therefore the angular velocity at the first frame and at the last frame is constrained to be as small as possible.

In order to find the best motion between a starting pose P_0 and an ending pose P_n, as well as meet all those constraints, the constraints are transformed to equations that can be minimized. Constraint number 1 and 2 corresponds to the minimization Equation (5-9) and Equation (5-10). This equation attempts to minimize the difference between the actual start and ending pose and the corresponding poses from the starting and ending points in M-space. Constraint number 3 corresponds to Equation (5-11). This equation tries to force the next pose in the motion to be as close as possible to the current pose plus the angular velocity. Constraint number four corresponds to Equations (5-12) and (5-13). These two equations attempt to find a point in N-space with the smallest possible angular velocity. Finally a damping factor is introduced in Equation (5-14). This damping factor ensures that each pose remains as close as possible to the subsequent pose.

$$\left\| P_0 - T_P^T N_0 \right\|^2 \tag{5-9}$$

$$\left\| P_n - T_P^T N_n \right\|^2 \tag{5-10}$$

$$\forall_{t=1..n-1} \left\| T_P^T N_{t+1} - (T_P^T N_t + T_v^T N_t) \right\|^2 \tag{5-11}$$

$$\left\| T_v^T N_0 \right\|^2 \tag{5-12}$$

$$\left\| T_v^T N_{n-1} \right\|^2 \tag{5-13}$$

$$\forall_{t=1..n-1} \left\| N_{t+1} - N_t \right\|^2 \tag{5-14}$$

To get the individual M-space point corresponding to a single frame within a motion, the L matrix can be used. The L matrix's value depends on the frame number.

$$N_k = L_k M \tag{5-15}$$

Therefore the Equations (5-9)-(5-14) can be written as

$$\left\| P_0 - T_P^T L_0 M \right\|^2 \tag{5-16}$$

$$\left\| P_n - T_P^T L_n M \right\|^2 \tag{5-17}$$

$$\forall_{t=1..n-1} \left\| T_P^T L_{t+1} M - (T_P^T L_t M + T_v^T L_t M) \right\|^2 \tag{5-18}$$

$$\left\| T_v^T L_0 M \right\|^2 \tag{5-19}$$

$$\left\| T_v^T L_{n-1} M \right\|^2 \tag{5-20}$$

$$\forall_{t=1..n-1} \left\| L_{t+1} M - L_t M \right\|^2 \tag{5-21}$$

The motion solution M is the motion that minimizes all those factors according to Equation (5-22). Each factor in this equation is weighted by the constant lambda that denotes the importance of each part. The first four factors have the same weight because they are similar in purpose (matching the pose or the velocity). Furthermore having one weight reduces the number of parameters that must be controlled.

$$\begin{aligned} &\lambda_1 \left\| P_0 - T_P^T L_0 M \right\|^2 + \lambda_1 \left\| P_n - T_P^T L_n M \right\|^2 \\ &+ \lambda_1 \left\| T_v^T L_0 M \right\|^2 + \lambda_1 \left\| T_v^T L_{n-1} M \right\|^2 \\ &+ \lambda_2 \sum_{t=1}^{n-1} \left\| T_P^T L_{t+1} M - (T_P^T L_t M + T_v^T L_t M) \right\|^2 \\ &+ \lambda_3 \sum_{t=1}^{n-1} \left\| L_{t+1} M - L_t M \right\|^2 \end{aligned} \tag{5-22}$$

To calculate the value of M that minimizes the above equation, the derivation for each factor with respect to M (df/dM) is calculated as shown in Equations (5-23)-(5-28)

$$2(L_0^T T_P T_P^T L_0)M - 2(L_0^T T_P P_0) \tag{5-23}$$

$$2(L_n^T T_P T_P^T L_n)M - 2(L_n^T T_P P_n) \tag{5-24}$$

$$2\sum_{t=1}^{n-1}(T_P^T L_{t+1} - T_P^T L_t + T_v^T L_t)^T (T_P^T L_{t+1} - T_P^T L_t + T_v^T L_t)M \tag{5-25}$$

$$2(L_0^T T_v T_v^T L_0)M \tag{5-26}$$

$$2(L_{n-1}^T T_v T_v^T L_{n-1})M \tag{5-27}$$

$$2\sum_{t=1}^{n-1}(L_{t+1} - L_t)^T (L_{t+1} - L_t)M \tag{5-28}$$

Finding the weighted sum of all the derivation of Equation (5-22) and setting it to 0, gives a formula that allows a matrix factorization to find M:

$$\begin{pmatrix} \lambda_1(L_0^T T_P T_P^T L_0) + \lambda_1(L_n^T T_P T_P^T L_n) \\ + \lambda_1(L_0^T T_v T_v^T L_0) + \lambda_1(L_{n-1}^T T_v T_v^T L_{n-1}) \\ + \lambda_2 \sum_{t=1}^{n-1}(T_P^T L_{t+1} - T_P^T L_t + T_v^T L_t)^T (T_P^T L_{t+1} - T_P^T L_t + T_v^T L_t) \\ + \lambda_3 \sum_{t=1}^{n-1}(L_{t+1} - L_t)^T (L_{t+1} - L_t) \end{pmatrix} M \tag{5-29}$$

$$= \lambda_1 L_0^T T_P P_0 + \lambda_1 L_n^T T_P P_n$$

The result of using this calculation to generate motion is shown in Figure 5-3. The original motion characteristic is shown on the left. The starting pose and ending pose of the original motion is used as input to the algorithm described in this chapter. The resulting motion shows variability in the total speed and angular speed.

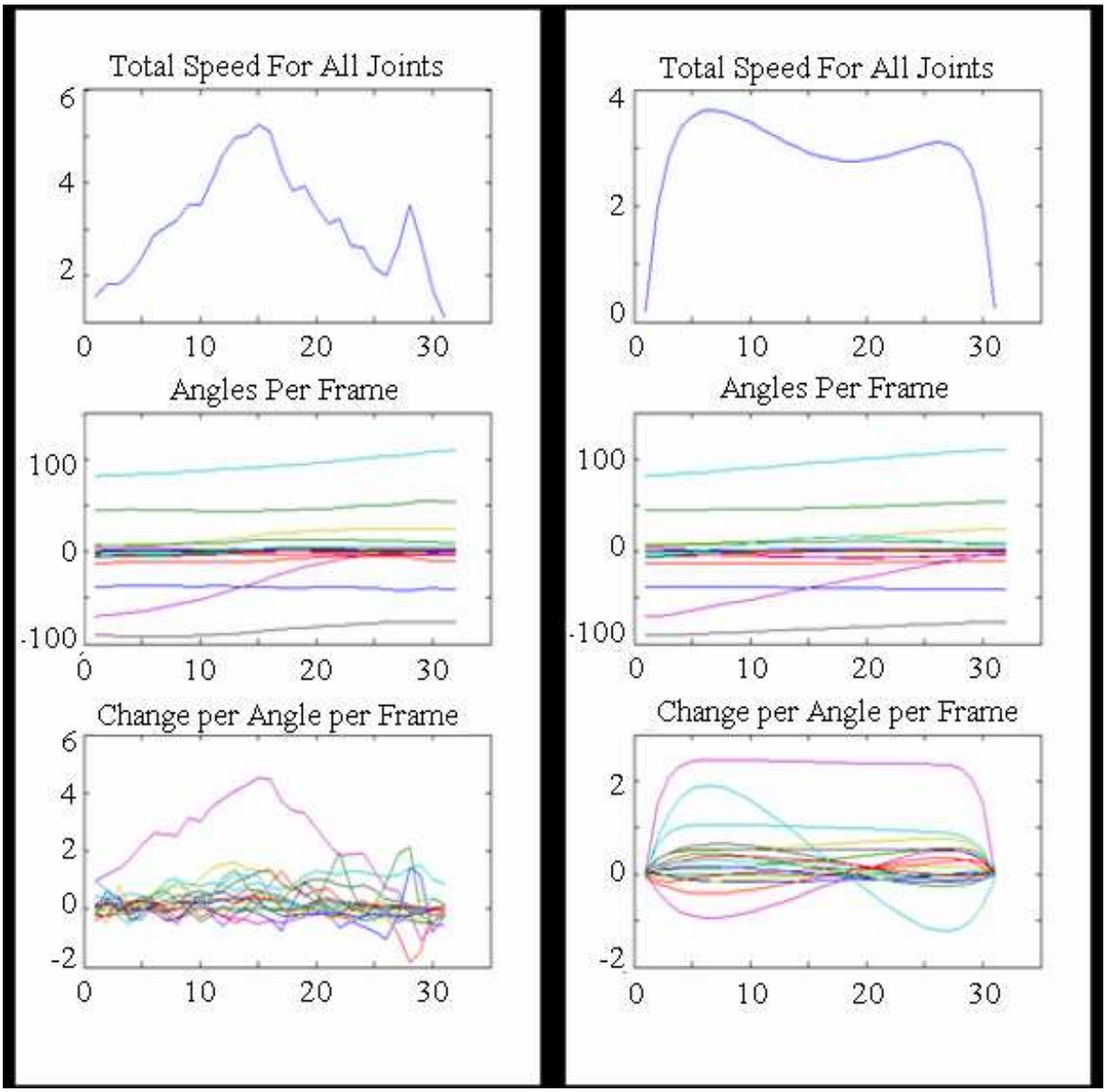

Figure 5-3 Total speed, angles, and angular velocity of motion calculated using M-space.

5.2.3 Characteristic of Motion Speed

The algorithm described so far minimizes the total speed at the beginning and the end. It does not constraint the speed of motion in the body of the motion. Different motions have different speed trajectories, and different motions can be created by changing the speed of the angles. An example of a speed trajectory based on a function is a quadratic function speed trajectory, which peaks at the middle of the motion.

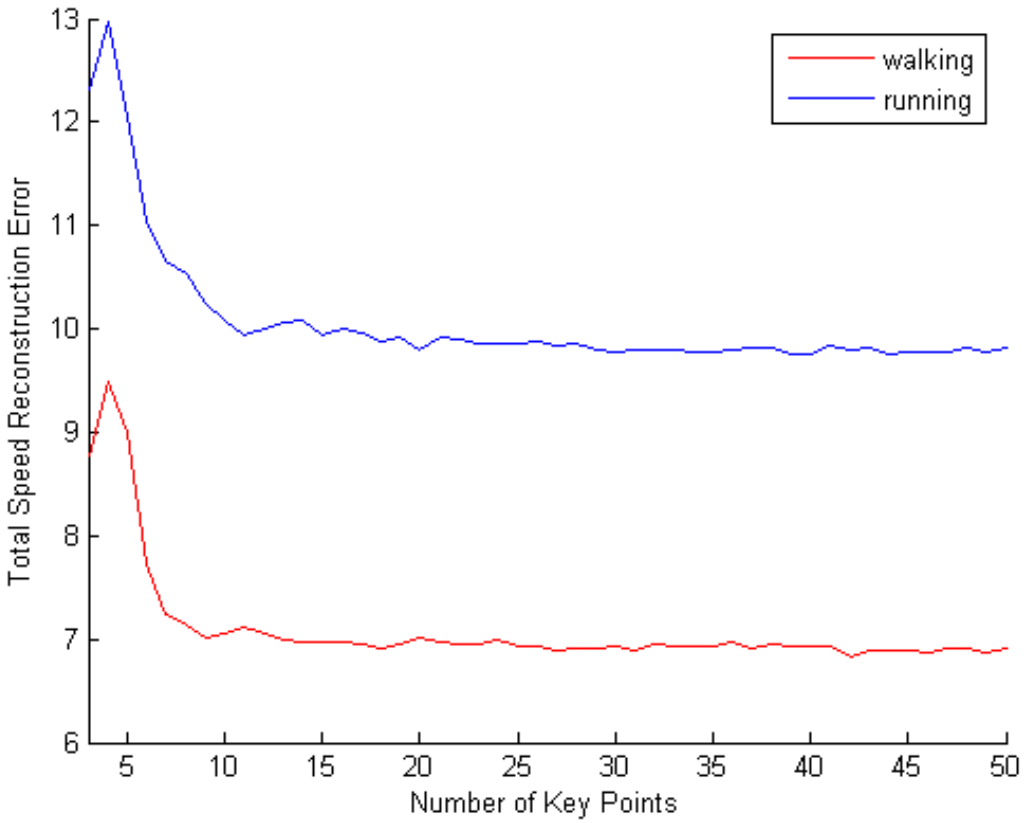

Figure 5-4 Mean error of speed reconstruction on original samples

Twelve walking motions and twelve running motions were selected from the motion capture database, each with different subjects. The motion of the right foot during the performance of a single step is used for speed analysis. The speed for each frame was calculated by taking the difference between subsequent frames. Every motion has different distances traveled and different number of time frames. Therefore the motion must be normalized. A set of regularly spaced key points, including the first and the last speed were selected. The speed at each key-point was then calculated. From this method 24 normalized speed trajectories from the motion samples were created. The prototypical speed for a motion is the average of those normalized speed trajectories.

To reconstruct the speed for a motion of n frames, the interval centers were matched to the frame index. A cubic spline algorithm [91] was used to create a curve that would best fit all the key points. This method was used to reconstruct the speeds of the original samples. The mean of the error of the reconstructed speed with respect to the original sample speed for key points between 3 and 50 was calculated and is shown in Figure 5-4. The figure shows that the error rate does not significantly decrease after 13 key points.

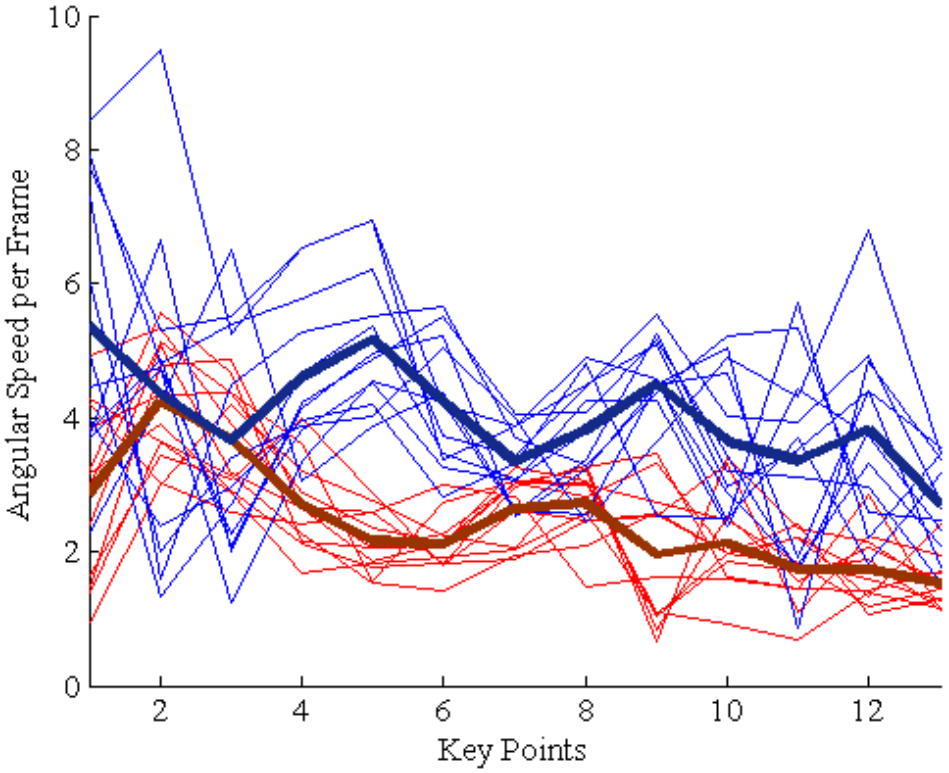

Figure 5-5 Mean speed trajectory of running motion and walking motion

Figure 5-5 shows the normalized mean speed trajectory for running and walking motions in 13 key points. The blue lines represent the running motion, while the red lines represent the walking motion. The thick lines represent the mean value or the prototype speed. As expected, different motions require different speed characteristics, and therefore this motion trajectory can be used to generate different motions. Hotelling's T squared method was used to calculate the

statistical significance of the multivariate differences of means [92]. This test shows that the means of the two types of motions differ significantly ($p<0.01$, $F_{13,10}=17.2947$)

5.2.4 Speed Trajectory Matching

Given a prototype speed, the next task is to use that speed to create a motion. The total speed Ω can be calculated from ω_x as shown in (5-30). To match the velocity to a set of speed constraint $S = \{s_0, s_1 \ldots s_n\}$ the following constraint could be used.

$$\Omega_x = \sqrt{\sum_i \omega_{x,i}^2} \tag{5-30}$$

$$\left\| \Omega_x - s_x \right\|^2 \tag{5-31}$$

To simplify this problem however, we use the squared total velocity in (5-32) and minimize

$$\Theta_x = \Omega_x^2 = \sum_i \omega_{x,i}^2 \tag{5-32}$$

$$\left\| \Theta_x - s_x^2 \right\|^2 \tag{5-33}$$

Finding the derivation of this formula and then using it to calculate the motion with Equation (5-33) turns out to be complex. The reason for the difficulty comes from the w variable being raised to the power of 4. Instead of trying to derive this formula, we calculate the Jacobian with respect to each individual ω Equation (5-34). Based on Equation (5-30), it can be inferred that each partial derivative is equal to the value 2ω.

$$\begin{aligned} J &= \begin{bmatrix} d\Theta_x / d\omega_{x,1} & d\Theta_x / d\omega_{x,2} & \cdots & d\Theta_x / d\omega_{x,n} \end{bmatrix} \\ &= \begin{bmatrix} 2\omega_{x,1} & 2\omega_{x,2} & \cdots & 2\omega_{x,n} \end{bmatrix} \end{aligned} \tag{5-34}$$

In order to match the speed we can add the constraints shown in Equation (5-35) and Equation (5-36). The term $\Delta\omega$ represents the moment velocity. Based on this moment velocity we can find the moment for motion ΔM

$$\forall_{t=1..n}\left\| J\Delta\omega_t - (v_t^2 - \Theta_t)\right\|^2 \tag{5-35}$$

$$\forall_{t=1..n}\left\| JT_v^T L_t \Delta M - (v_t^2 - \Theta_t)\right\|^2 \tag{5-36}$$

The derivation of this with respect to ΔM is:

$$2(L_t^T T_v J^T J T_v^T L_t)\Delta M - 2(L_t^T T_v J^T (v_t^2 - \Theta_t)) \tag{5-37}$$

The solution to M can be found through iteration. After every iteration, ΔM is added to M as shown in Equation (5-38). In order to use this iteration, Equation (5-22) must be changed to use both M and ΔM.

$$M_{t+1} = M_t + \Delta M \tag{5-38}$$

To find ΔM for each iteration, Equation (5-39) must be minimized. Equation (5-39) is similar to the original M-space equation shown in Equation (5-22), with an extra speed trajectory matching factor added. By calculating the derivation of Equation (5-39) and setting it to 0, the most optimal value of ΔM can be calculated using Equation (5-40).

$$
\begin{aligned}
&\lambda_1 \left\| X_0 - T_P^T L_0 (M + \Delta M) \right\|^2 + \lambda_1 \left\| X_n - T_P^T L_n (M + \Delta M) \right\|^2 \\
&+ \lambda_1 \left\| T_v^T L_0 (M + \Delta M) \right\|^2 + \lambda_1 \left\| T_v^T L_{n-1} (M + \Delta M) \right\|^2 \\
&+ \lambda_2 \sum_t \left\| (T_P^T L_{t+1} - T_P^T L_t + T_v^T L_t)(M + \Delta M) \right\|^2 \\
&+ \lambda_3 \sum_t \left\| L_{t+1}(M + \Delta M) - L_t (M + \Delta M) \right\|^2 \\
&+ \lambda_4 \sum_t \left\| J T_v^T L_t \Delta M - (v_t^2 - \Theta_t) \right\|^2
\end{aligned}
\tag{5-39}
$$

$$
\begin{pmatrix}
\lambda_1 (L_0^T T_P T_P^T L_0) + \lambda_1 (L_n^T T_P T_P^T L_n) \\
+ \lambda_1 (L_0^T T_v T_v^T L_0) + \lambda_1 (L_{n-1}^T T_v T_v^T L_{n-1}) \\
+ \lambda_2 \sum_{t=1}^{n-1} (T_P^T L_{t+1} - T_P^T L_t + T_v^T L_t)^T (T_P^T L_{t+1} - T_P^T L_t + T_v^T L_t) \\
+ \lambda_3 \sum_{t=1}^{n-1} (L_{t+1} - L_t)^T (L_{t+1} - L_t) \\
+ \lambda_4 \sum_{t=1}^{n-1} (L_t^T T_v J^T J T_v^T L_t) - (L_t^T T_v J^T (v_t^2 - \Theta_t)
\end{pmatrix} \Delta M
\tag{5-40}
$$

$$
\begin{aligned}
&= \lambda_1 L_0^T T_P X_0 - \lambda_1 L_0^T T_P M + \lambda_1 L_n^T T_P X_n - \lambda_1 L_n^T T_P M \\
&- \lambda_1 L_0^T T_v M - \lambda_1 L_{n-1}^T T_v M \\
&- \lambda_2 \sum_{t=1}^{n-1} (T_P^T L_{t+1} - T_P^T L_t + T_v^T L_t)^T M \\
&- \lambda_3 \sum_{t=1}^{n-1} (L_{t+1} - L_t)^T M + \lambda_4 L_t^T T_v J^T (v_t^2 - \Theta_t)
\end{aligned}
$$

The result of using this calculation to generate motion is shown in Figure 5-6. The original motion characteristic is shown on the left. The starting pose and ending pose of the original motion is used as input to the algorithm described in this chapter. The right data shows the motion trying to match quadratic function speed. The resulting motion shows variability in the total speed and angular speed.

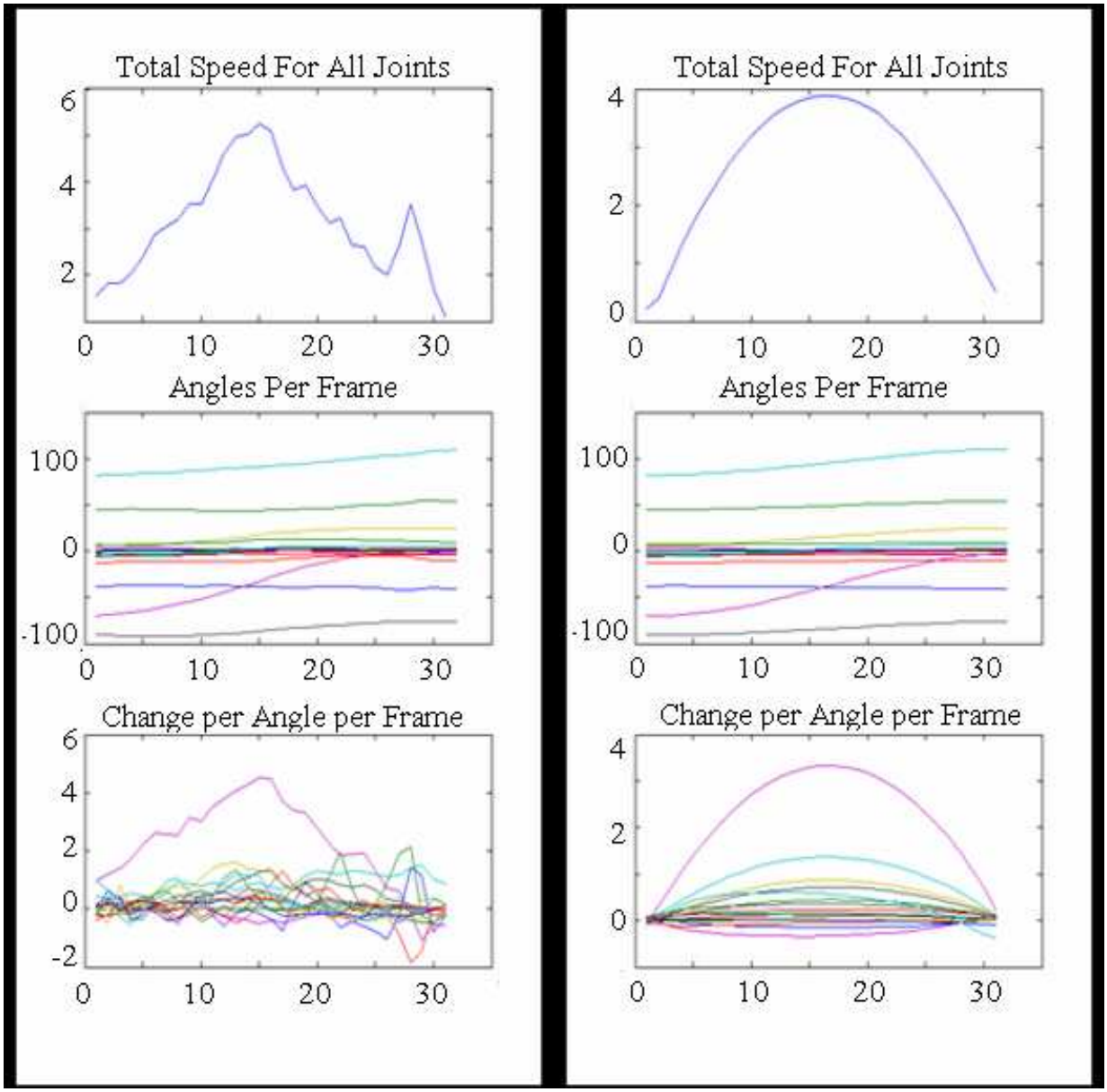

Figure 5-6 Total speed, angles, and angular velocity of motion calculated using Speed Trajectory Matching

5.2.5 Multiple Spaces

The body space is divided into 5 segments as described in Chapter 3.3. To calculate the motion for the whole body, each segment is calculated separately. The angle of joints that are in multiple segments is simply the average of the angle values for that joint in all segments. Equation (5-41) describes the calculation for the k^{th} joint angle, which is an average of the angles in all spaces that contains the k joint angle.

$$\theta_k = \frac{\sum_i \theta_{i,k}}{n} \tag{5-41}$$

5.3 Results

Two types of evaluation are performed on this algorithm. The first evaluation is aimed at measuring the similarity between a motion generated by this algorithm and a motion taken from the motion capture data. To do this, the angle difference at each frame is calculated. A visual observation of the motions is also performed. The second evaluation attempts to asses new motions synthesized with the M-space. Visual observation of the created motion is used in assessing the resulting motion.

Table 5-1 Number of dimensions per M-space segment

Body Segment	**Number of Dimensions**
Right hand	18
Left hand	18
Head	18
Right Foot	8
Left Foot	8

The body space is divided into 5 segments. By using the minimum distance between actual motion and motion created using M-space, the number of dimensions used are shown in Table 5-1.

5.3.1 Motion Reconstruction

The first evaluation compares a motion created in M-space and one existing in the motion capture data. Four clips were selected from either the right hand or the right foot segments. The clips chosen had a speed trajectory that was similar to a quadratic function. The clips start off slow, peak near the middle, and slow down again near the end. The start pose and the end pose was given as input to the various algorithms tested.

The results of three different motion generating algorithms were compared. The first algorithm is a simple linear interpolation between the first pose and the end pose. The second motion algorithm uses the M-space algorithm described in chapter 5.2.2. The third algorithm is the speed trajectory matching algorithm of chapter 5.2.4. The trajectory of the speed was a quadratic function with a peak of 5 degrees/frame.

Table 5-2 Euclidian distance for the right hand segment motions

Clip	**Number of Frames**	**Interpolation**	**M-Space**	**Speed Trajectory**
Clip 1	41	13.9794 (7.4394)	13.9380 (7.9521)	14.2215 (8.6094)
Clip 2	32	10.2899 (6.0120)	12.3045 (7.2574)	9.0544 (5.3202)
Clip 3	13	14.3558 (9.0049)	14.6976 (8.6656)	13.7084 (7.9130)
Clip 4	47	32.4070 (15.5398)	33.0236 (16.9043)	31.0832 (14.9078)
Total	133	19.6405 (14.4471)	20.3638 (15.0579)	18.8868 (14.0731)

The total Euclidian distance between the joint angles was calculated for each frame. The farther the distance of the natural motion and the generated motion, the less natural the motion is deemed to be.

The mean and standard deviation (in brackets) for each algorithm is shown in Table 5-2. Based on the data found here, all three algorithms create motions which are similar in terms of naturalness. Paired T-tests shows that the differences were not statistically significant (t score were -0.3997 for methods 1 and 2, 0.4310 for methods 1 and 3, and 0.8265 for methods 2 and 3).

Similarly, Table 5-3 shows the mean and standard deviation of distances for the right foot. It is clear here that the speed trajectory algorithm performs better for foot motion than the other algorithms. Paired T-tests shows that the differences are statistically significant for $p<0.005$ (t score were –8.0657 for methods 1 and 2, 3.9803 for methods 1 and 3, and 10.4810 for methods 2 and 3).

Table 5-3 Euclidian distance for the right foot segment motions

Clip	**Number of Frames**	**Interpolation**	**M-Space**	**Speed Trajectory**
Clip 1	37	15.7634 (7.3665)	26.1878 (14.8549)	8.7737 (4.5666)
Clip 2	31	14.1270 (9.0573)	31.2707 (18.2234)	11.3426 (7.7485)
Clip 3	21	8.7946 (4.5610)	44.1516 (30.6322)	4.6237 (2.8625)
Clip 4	41	13.0720 (9.7789)	25.6661 (19.2689)	11.0984 (7.0715)
Total	130	13.7276 (8.7391)	28.7152 (19.5668)	9.9472 (6.6033)

A second evaluation was performed on this data. A walking motion was taken from the motion capture database. The original walking motion was of 60 frame length. To generate the motion, the first and the last frame of the walking motion were given to the interpolation and the

M-space algorithms. The speed matching algorithm with a quadratic function approximation was used. The maximum sum of angle speed per frame was 3 degrees for the arms and left leg segments, 2 for the head segment, and 5 for the right leg segment.

Figure 5-7 compares various walking motion created by regular interpolation and using motion space. The original walking motion is shown in the top row. The second row shows the result of linear interpolation. The third row shows the result from the M-space with speed matching.

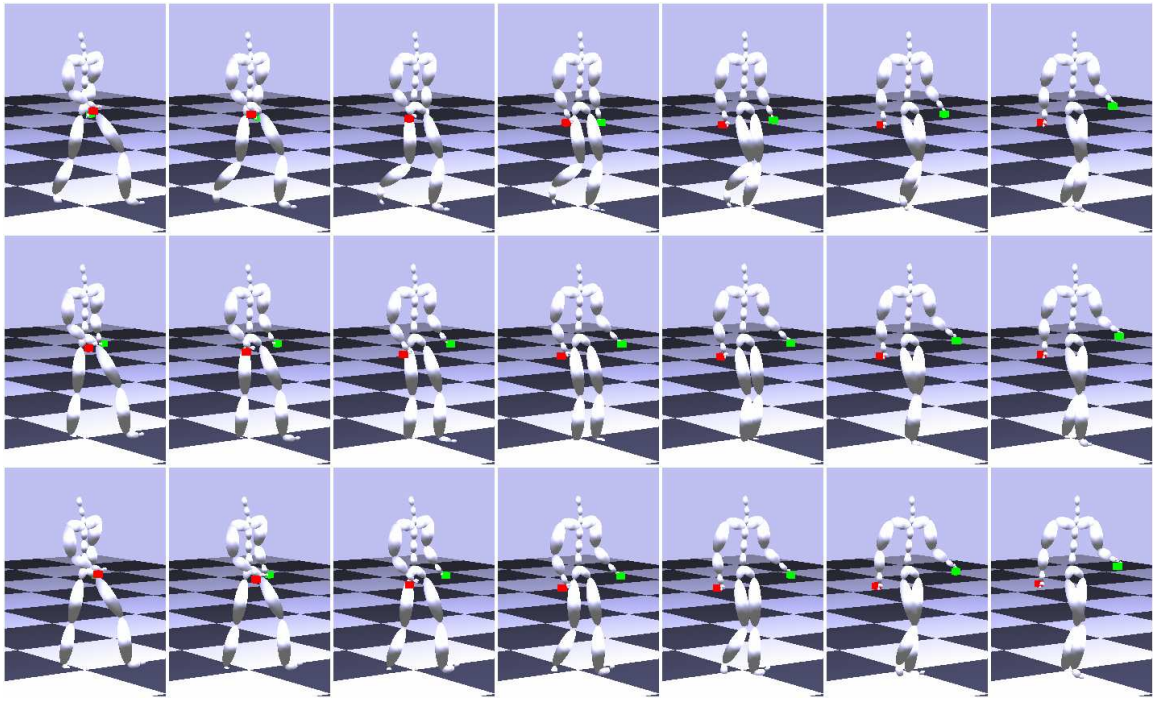

Figure 5-7 Comparison of walking motion

The problem with linear interpolation can be seen in the right leg segment of the motion. In a normal walking motion shown in the first row, a person bends his knees to lift the leg forward. This is not seen in the motion created by interpolation. The motion drags the feet from the back to the front. The M-space motion created on the other hand shows some knee bending and as a result a small leg lift occurred, albeit not as high as the lift in the normal walking motion.

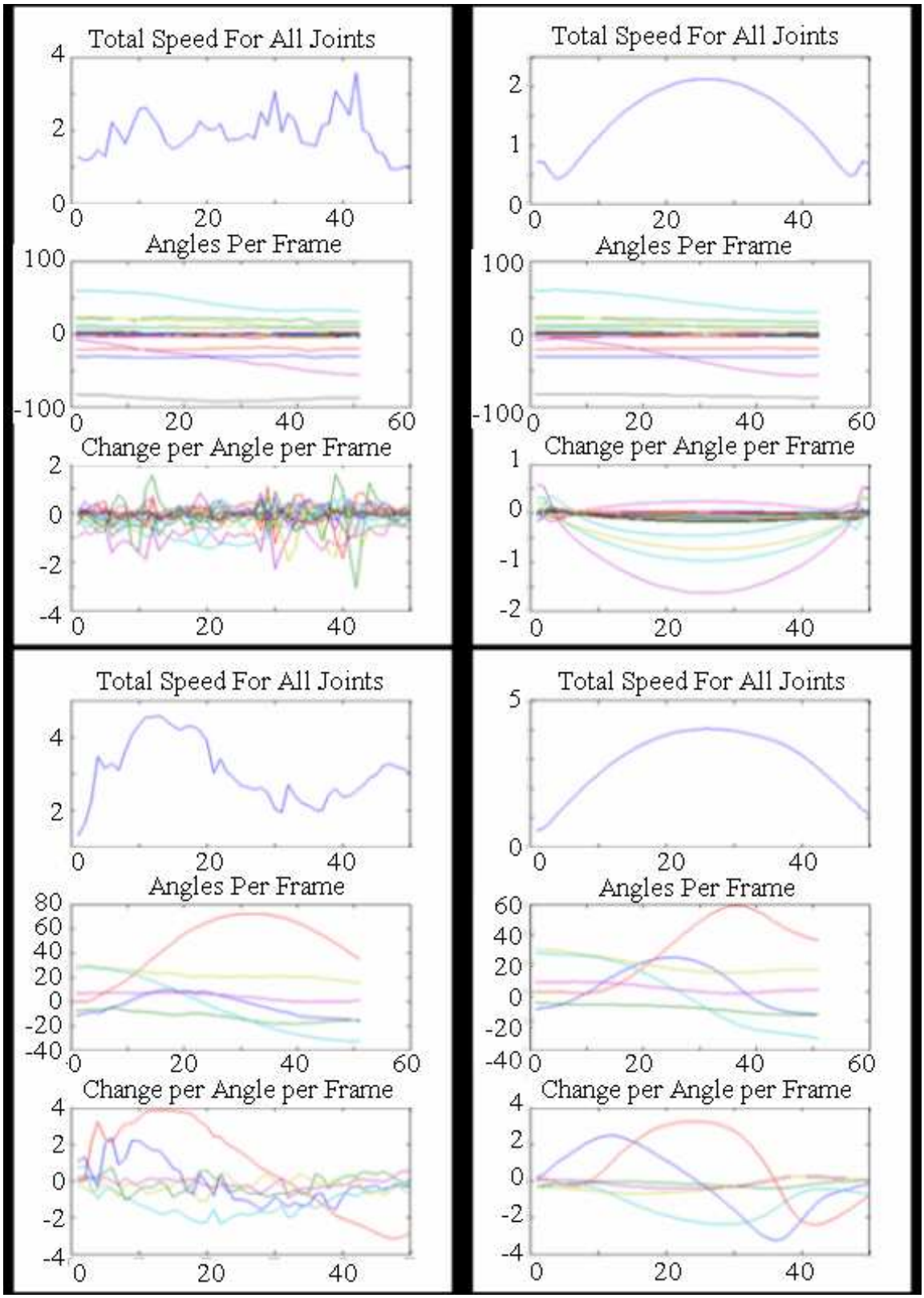

Figure 5-8 Speed and angle comparison of right hand and right foot

Figure 5-8 shows the speed and angles for both the right hand (top) and the right foot (bottom) segments. The left graphs show the original speed and angle per frames. The right graphs show the speed and angle per frames of the motion created by the speed trajectory matching algorithm. The motion generated by the speed trajectory matching algorithm follows a path that in most joints mimics the path in the original motion.

5.3.2 Pose Based Motion Synthesis

In most applications of human animation, there is a need to synthesize new motions based on existing data. The method described in this thesis allows different ways of creating new motion to meet an application's need.

Motions are generated based on the start and end poses. By changing the end pose, new natural motions must be generated. The top figures of Figure 5-9 shows an animated human pointing upwards. New motions are generated by changing the direction that the human points to. Two new motions were generated, one pointing to the front and one pointing to the side.

The first pose for each motion was taken from the original motion capture data. The multi space pose generator described in Chapter 4.3 was used to generate the end poses. The speed trajectory matching algorithm with a quadratic function motion trajectory with a peak of 5 was used for creating both motions. The algorithm was able to create a novel motion that meets both speed and position constraints.

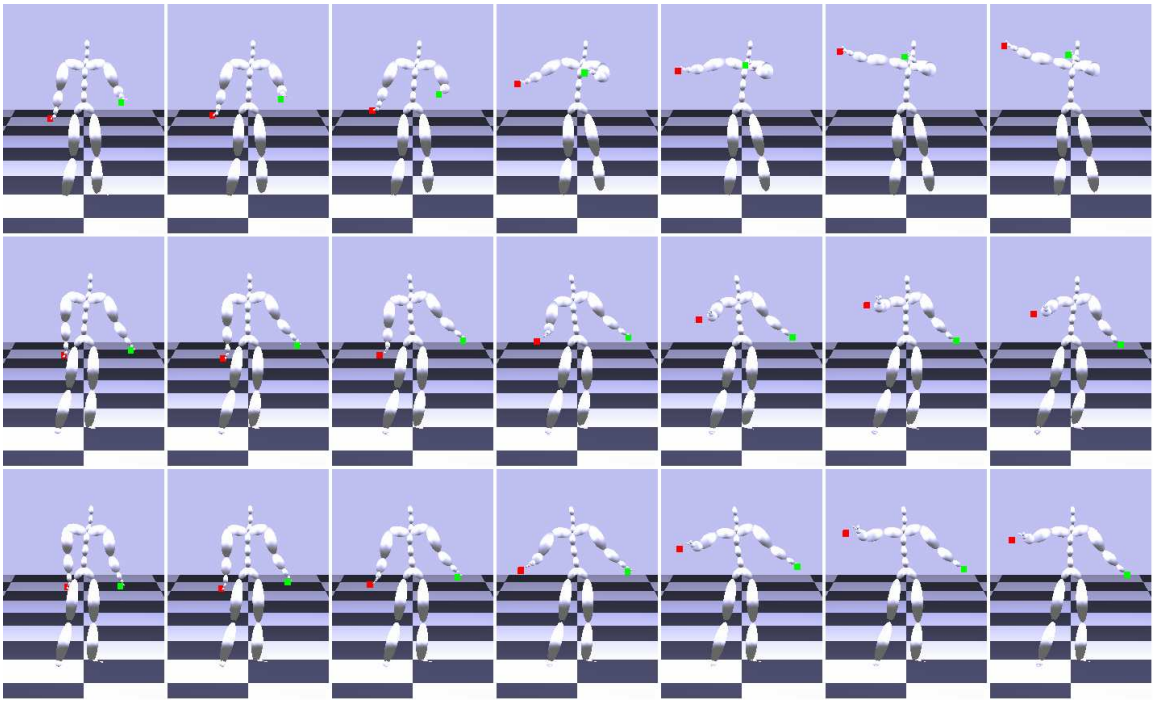

Figure 5-9 Creating new pointing motions using natural spaces

5.3.3 Speed Based Motion Synthesis

An alternate way of generating new motions is to change the speed of various parts of the body. However one must be very careful that the motion remains natural, even when the speed changes. This method allows a user to set the total speed for a segment. This allows the user to control which segment needs to change, while at the same time not worry about changing individual joint angles and how they relate to one another.

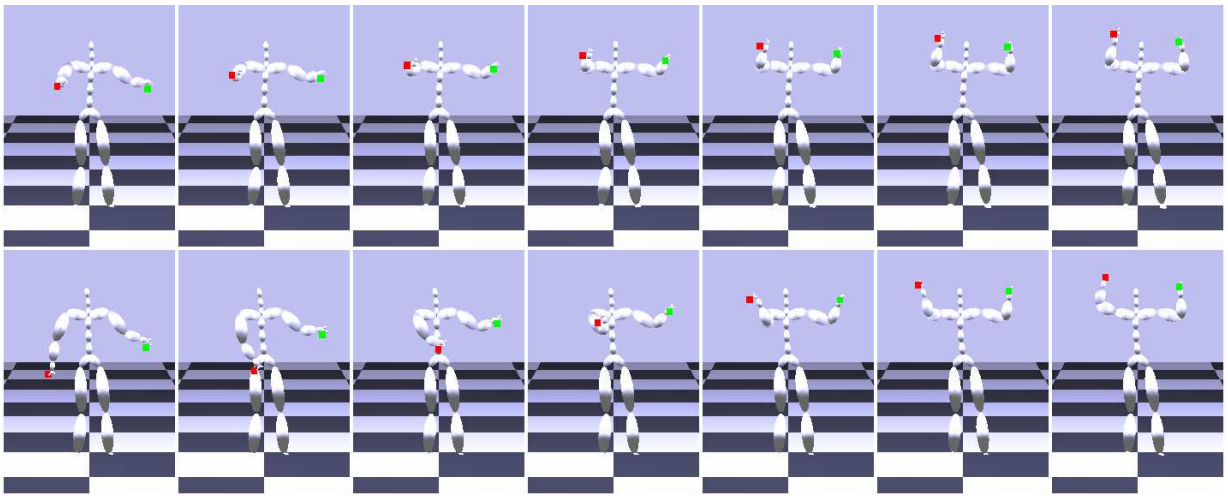

Figure 5-10 Effect of a different speed trajectory to motion

The animated figure in Figure 5-10 is performing traffic control actions (waving cars through an intersection). The figures on top show the original motion. The speed of the right hand segment was changed to have a peak of 8. However the starting and ending poses remain the same. The figures on the bottom show the resulting motion after the speed was changed. Due to the change of speed, the character had to move the hands in a curved trajectory in order to meet both positional and speed constraints. As a result of this change of speed, the end motion is significantly different from the original motion. Figure 5-11 shows the speed and angle per frame of the right hand segment. The left graphs show the original motion capture data while the right graphs show the data after the change of speed.

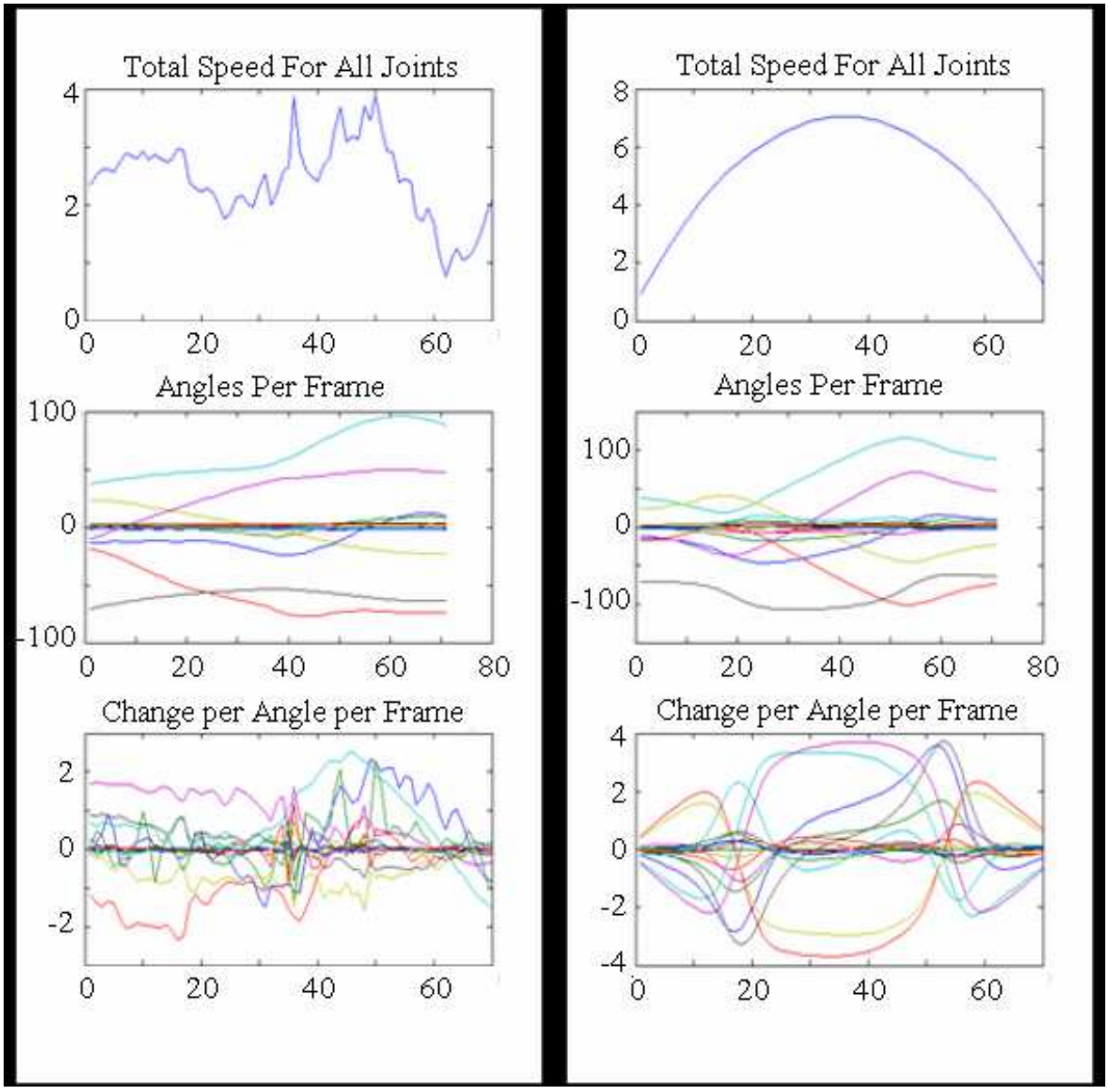

Figure 5-11 Speed and angle per frame after fitting with new trajectory

A prototype speed trajectory (as shown in chapter 5.2.3) can also be used to generate motions. A walking motion was taken from the motion capture database. The original walking motion was of 86 frame length. A 60 frame motion was created to speed up calculation. To generate the motion, the first and the last frame of the walking motion were given to the interpolation and the M-space algorithms. Instead of matching the speed at all frames, speed were matched every 2 frames. In essence, this allows the algorithm to choose a more natural

speed for some frames and not make unnecessary movements just to meet the pre arranged speed constraints.

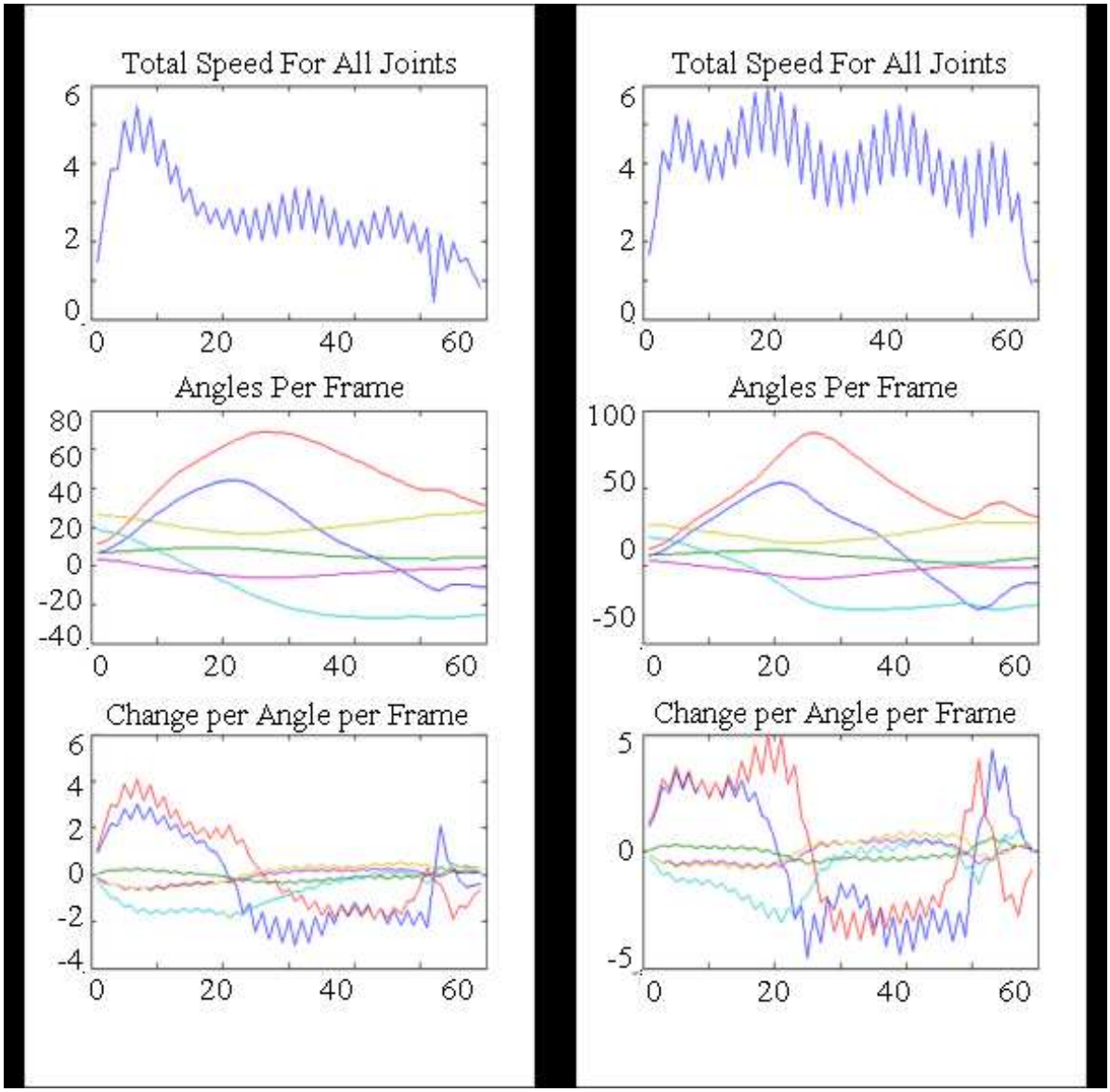

Figure 5-12 Speed trajectories for walking and running motion

Figure 5-12 shows the speed and angle per frames generated by this algorithm for two prototype speed trajectories, a walking and a running motion that starts and ends with the same pose. Figure 5-13 shows the results of synthesizing a motion having those speed trajectories. The top images show the original motion taken from the motion capture data. The middle images

show the synthesized walking motion while the bottom images show the synthesized running motion.

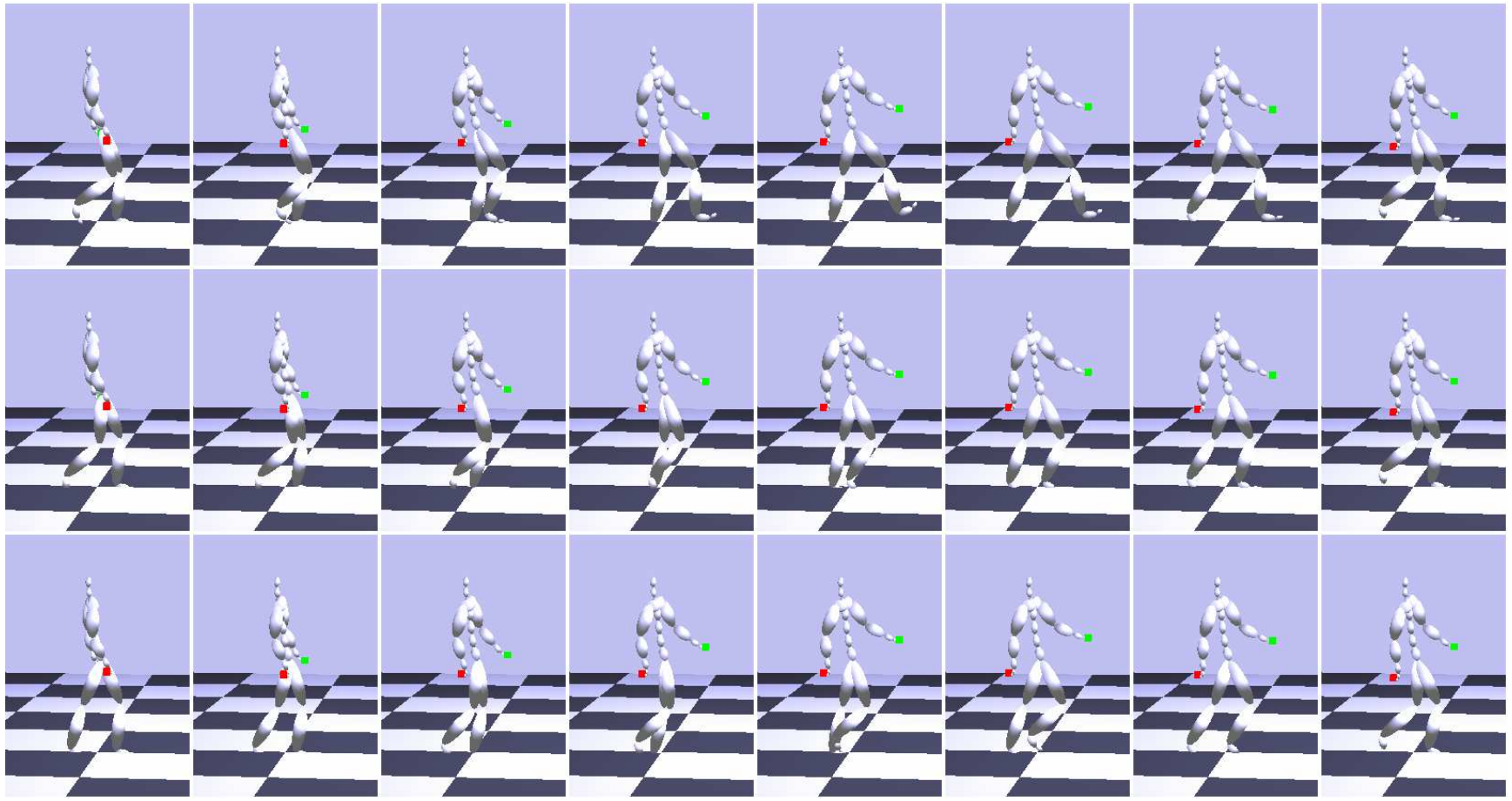

Figure 5-13 Walking and running motions generated from prototype speed trajectories. The top figures show the original motion. The figures on the second row shows generated walking motion, while the figures on the third row shows generated running motion

6 Conclusion and future work

6.1 Conclusion

This thesis describes a set of algorithms that creates natural spaces, subspaces of all possible poses and motions, and how to utilize these subspaces to create more natural poses and movement.

The skeletal model is divided into multiple overlapping segments. PCA is performed for the motion capture data in each segment. The resulting space is called the natural space. Inverse kinematics on the PCA space is used to create constrained poses. Furthermore to retain the relation between the various segments, an effector space is created consisting of the coordinates of 8 end effectors / bones.

Naturalness was measured by calculating as log likelihood of a set of poses created by an algorithm given a model of naturalness. The mean and variance of the motion capture data was calculated and used as the parameter for a normal distribution. This normal distribution of data is used as the model for comparing naturalness of poses. The likelihood is calculated by calculating the probability density function (pdf) of poses to the model.

Resulting poses are most natural when the number of PCA dimensions is around 50% of the original number of dimensions. A lower number of dimensions severely limits the poses that the algorithm is able to use, therefore, it is not likely to find one that meets the constraints. A high number of dimensions on the other hand give too much freedom on the resulting bone angles. The relationships between the bone angles are lost, resulting in unnatural poses.

The algorithm proposed here creates poses that are more natural compared to traditional iterative Jacobian and the single PCA space method. As predicted the traditional iterative Jacobian performs the worst, as it focuses on finding solutions without taking naturalness into account. The single PCA solution creates relations over body parts that may not really exist, and are only there due to insufficient learning data. By breaking the space into multiple segments, this algorithm is better place the bones into a pose that fits the constraints, while at the same time keeping it natural.

To create motion, a motion space was created from the phase space. The phase space includes the angle of the bones as well as the speed of each angle. Similar to the pose space, the skeletal model is also segmented into the same space.

Given a starting pose and an ending pose, the algorithm described in this thesis creates the inner poses in order to create a motion. A single vector represents each phase of the motion from the starting pose to the ending pose. The motion generated algorithm searches through the motion space to fill out the vector with motion phases that meets the constraint.

Traditional linear interpolation assumes changes in joint angles are always constant. The motion capture data however shows that this is not always true. With linear interpolation the angle of a joint at any time t is always a linear product of t and the angle speed. The speed of joint angles in natural motion changes constantly and cannot be easily replicated by linear interpolation.

The first algorithm proposed finds a solution by minimizing the beginning and ending speed. This algorithm is able to create motions that have variance in its speed, similar to natural motion. The second algorithm proposed goes even further by allowing a

user to control the speed of the motion at each step. Therefore the user can select a speed curve (such as a simple quadratic curve) which the algorithm tries to match. This algorithm is also able to create motions with variable speed for each joint, creating a more natural motion.

The work described here shows that the use of multiple reduced dimension space is a feasible alternative in generating poses and motions. By utilizing the statistical properties of motion capture data, these methods generate more natural looking poses and motions compared to traditional methods.

6.2 Future Work

There are various directions where this work can be expanded. Some of the areas for expansion include a more thorough study of data, optimization, methods merging, and evaluations.

Chai and Hodgins dismissed the use of global PCA in their work because it over-generalizes the data [93]. One reason for this is that they use global PCA instead of the multiple space method described here. Even the multiple space method can be broken down further. Some have used techniques that work on motion capture data for specific movement [79]. Preliminary studies by this author, shows that categorizing motion and using different category results for the motion capture data results in different motions. Further study can be performed to utilize fully the specific properties of each motion.

Different emotions elicit different types of movement [21]. PCA has been used to generate different walking movement based on emotions [77]. This work can be expanded by using emotion based motion capture and eliciting motions and poses that is typical of an emotion.

Quaternion representation of joints has some advantages over Euler angles [78]. Various PCA algorithms on motion have been used using this quaternion representation [78], [81]. A comparative study for multi space PCA using quaternion as compared to the current Euler angle is needed.

No optimization or algorithmic analysis is performed on the current algorithm. Therefore it is impossible to know whether the algorithm here is suitable for real time systems. The complexity of this algorithm comes from finding the inverse of matrices. As some of the matrices are sparse, it may be possible to speed up the inverse calculation.

The algorithm described here can be easily incorporated into other programs. The pose generating mechanism can be used in the RRT search algorithm proposed in Yamane [63] and Pan [64]. The motion generating algorithm does not necessarily need to take as input the results from the pose generating algorithm. All it needs is the skeletal configuration for the starting pose and the ending pose.

The assumption used in this work is that motion is transferable between subjects and body types. One of the findings of Pronost et al, is that different body will have different correlation between joints, and thus different type of movement [66]. Future research can weigh in on how to use the body information to transfer or change the natural space.

In this thesis, naturalness is measured objectively through the use of a probability distribution function. It would be interesting to see whether the poses and motions created here are natural according to direct observation by humans. To test the naturalness of the motion created, subjects may try to determine [94] which two motions were generated by a real human, and which were generated by this algorithm. If the subjects were not able to

discern which is which, it can be concluded that the algorithm proposed creates animation of natural motion.

APPENDICES

Appendix A. Multiple Space PCA Results

The following figures show selected comparison of running the iterative Jacobian, Single PCA, Single PCA +, and Multi PCA algorithms as described in Chapter 4.3 on various constraints. For all figures, the skeleton shows the results of running the respective algorithms in order from left to right.

In all of the following figures, the Multi PCA found a solution pose that meets the constraints set. The other algorithms may also possibly have found a solution pose to the constraints.

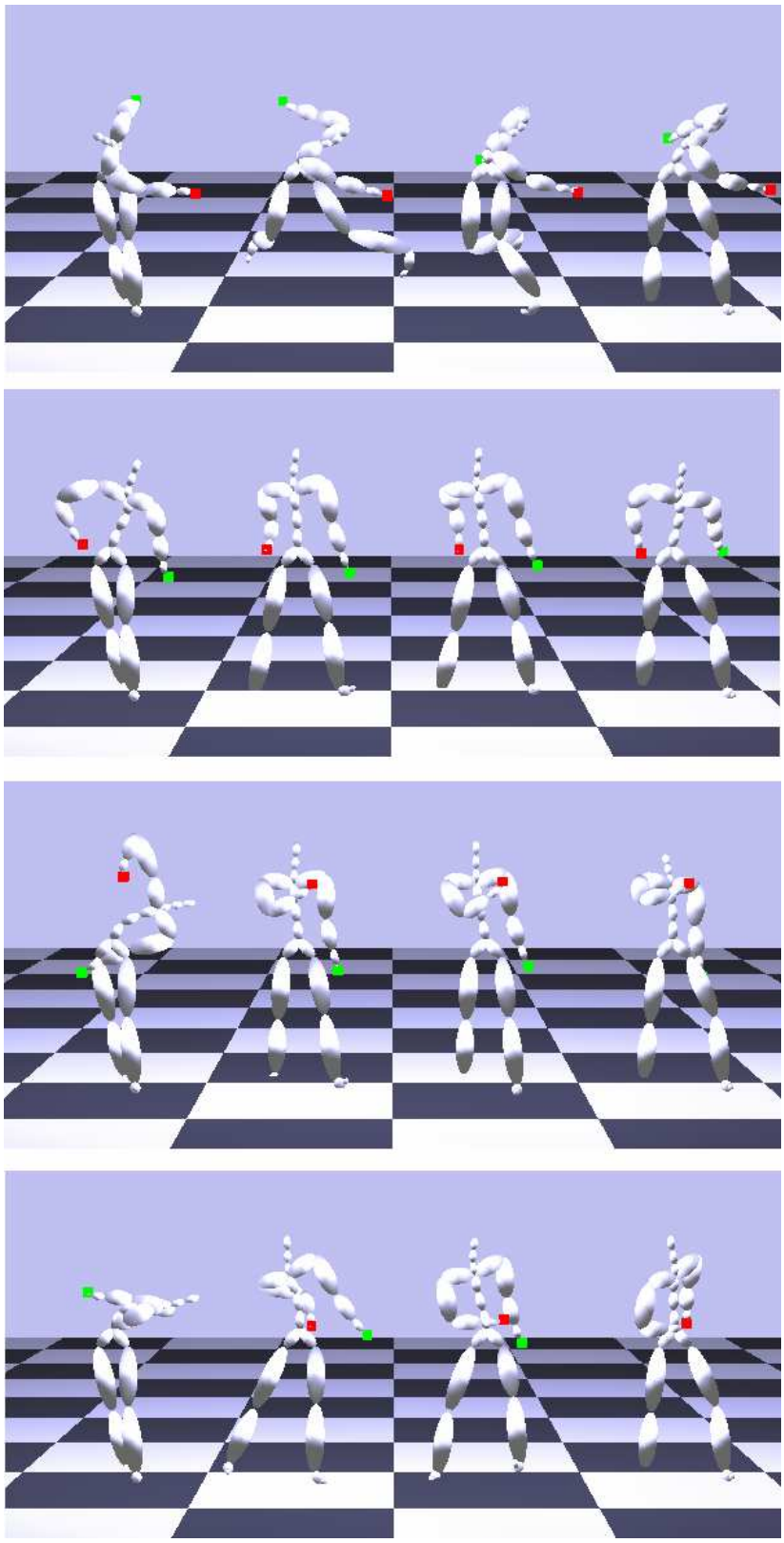

Figure 6-1 Results of running the pose generation algorithms on right hand constraints only

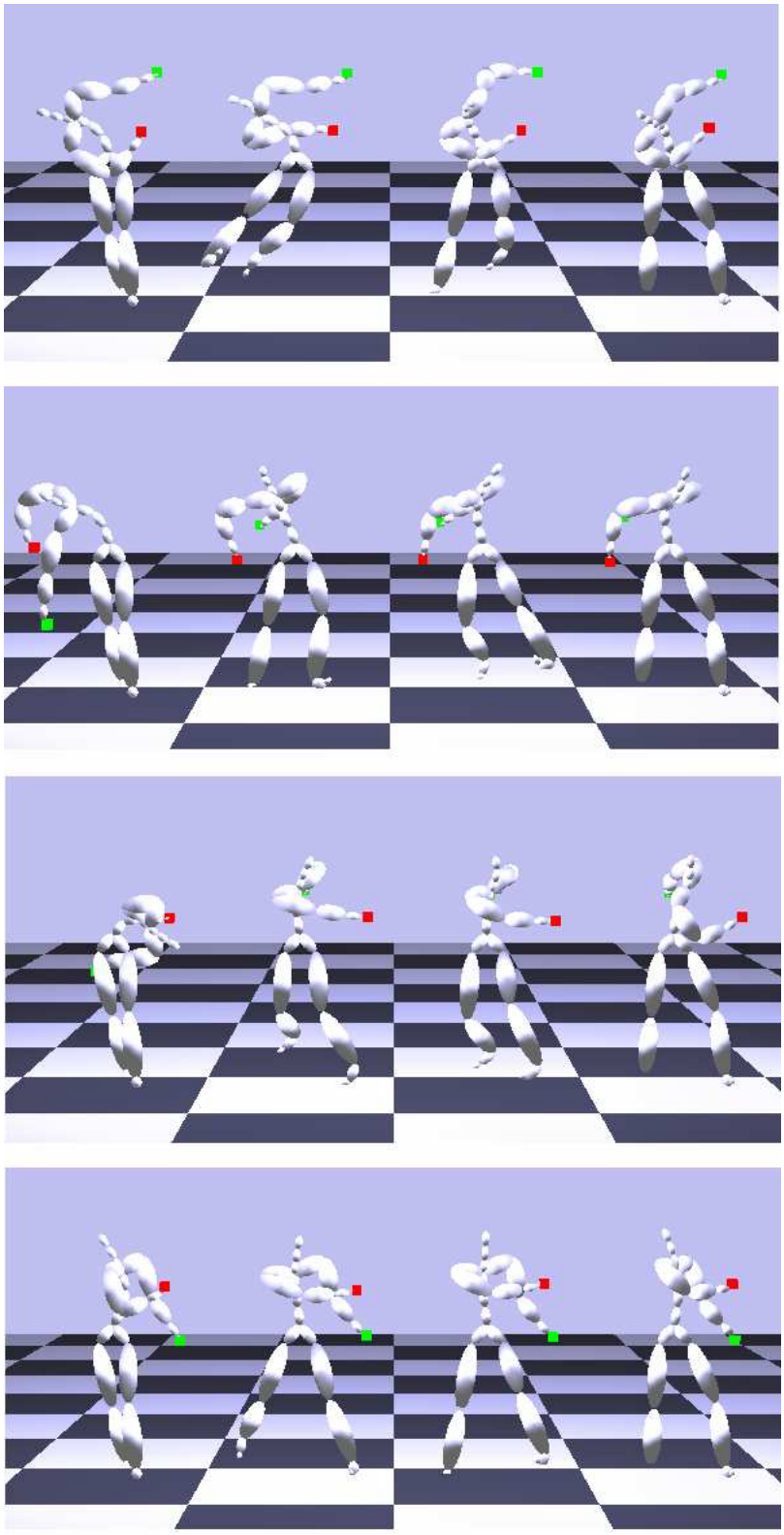

Figure A-2 Results of running the pose generation algorithms on right hand and left hand constraints

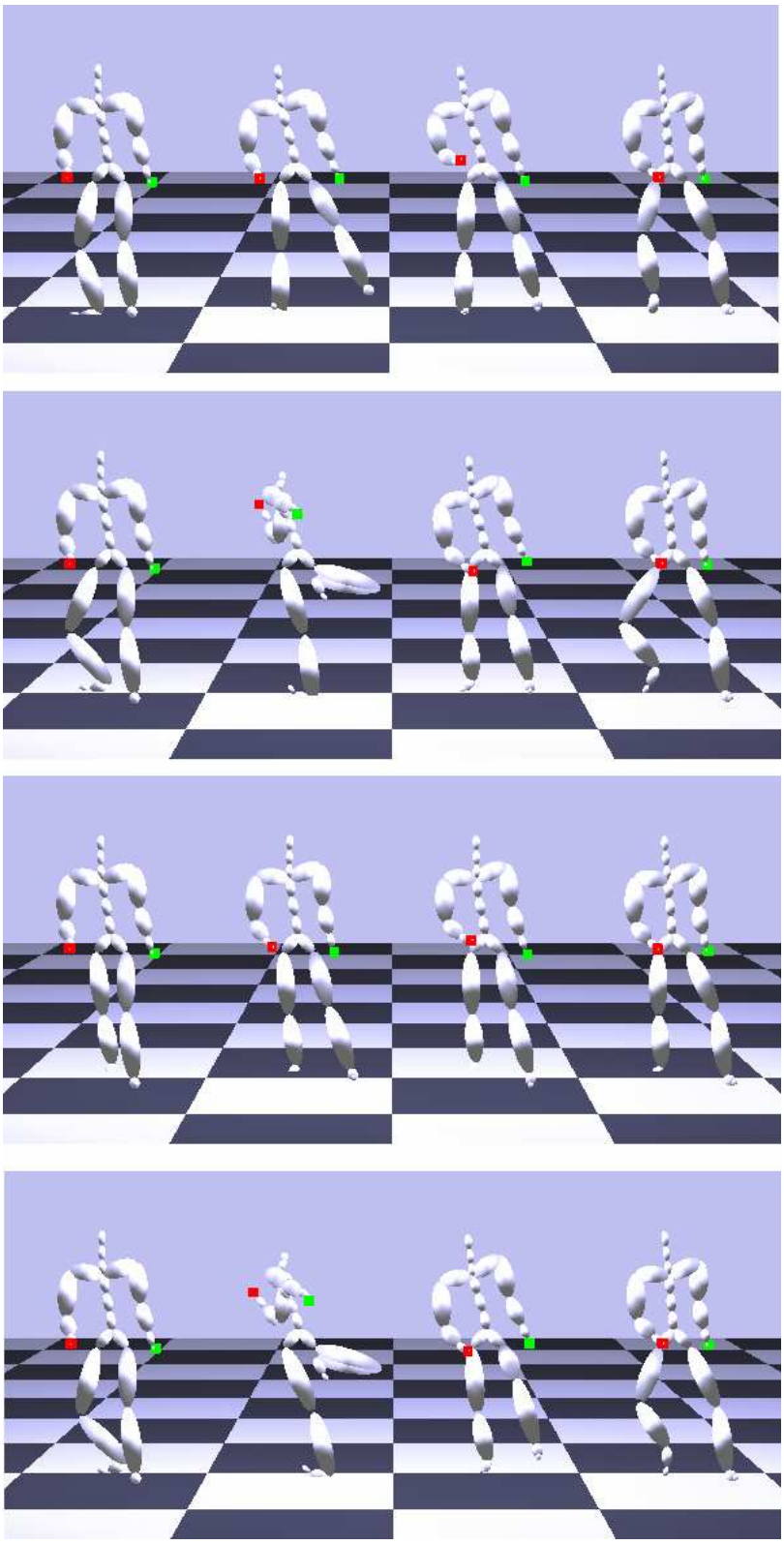

Figure A-3 Results of running the pose generation algorithms on right foot constraint only

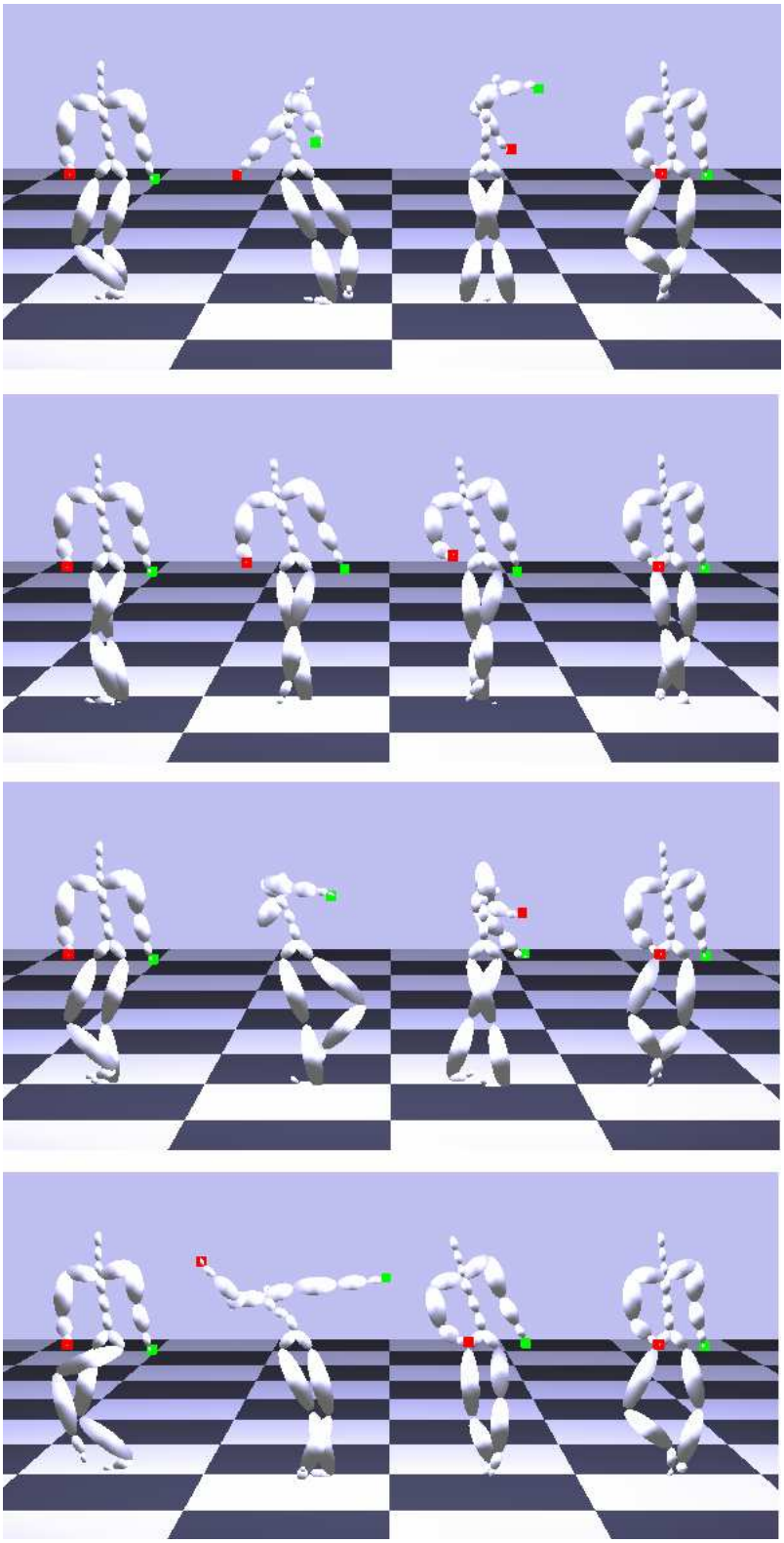

Figure A-4 Results of running the pose generation algorithms on right and left foot constraints

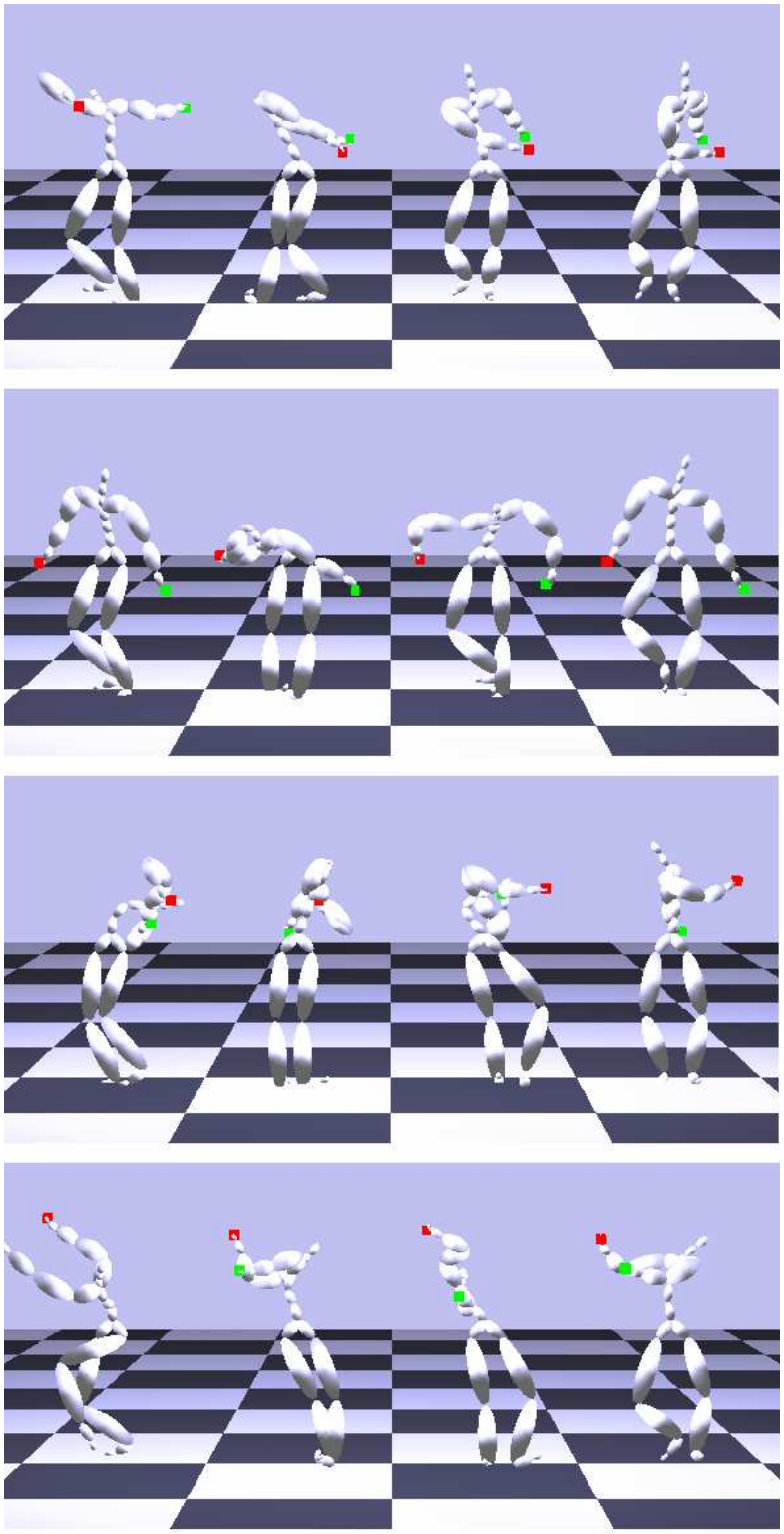

Figure A-5 Results of running the pose generation algorithms on hands and foot constraints

REFERENCES

References

[1] Nadia Magnenat-Thalmann and Daniel Thalmann, "An Overview of Virtual Humans," in *Handbook of Virtual Humans*, John Wiley & Sons, 2004.

[2] K. L. Nowak and F. Biocca, "The effect of the agency and anthropomorphism of users' sense of telepresence, copresence, and social presence in virtual environments," *Presence: Teleoperators and Virtual Environments*, vol. 12, p. 481–494, Oct. 2003.

[3] P. Goldstein, "Why is the $170-million 'Polar Express' getting derailed?," *Los Angeles Times*, 2004.

[4] F. Charles, M. Cavazza, S. J. Mead, O. Martin, A. Nandi, and X. Marichal, "Compelling experiences in mixed reality interactive storytelling," in *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*, Singapore: ACM Press, 2004, pp. 32-40.

[5] M. Mateas and A. Stern, "Towards Integrating Plot and Character for Interactive Drama," *Socially Intelligent Agents: The Human in the Loop. Papers from the 2000 AAAI Fall Symposium*, pp. 113-118, 2000.

[6] J. Gratch, "Émile: Marshalling passions in training and education," in *Proceedings of the fourth international conference on Autonomous agents*, Barcelona, Spain: ACM Press, 2000, pp. 325-332.

[7] J. Rickel and W. L. Johnson, "Integrating pedagogical capabilities in a virtual environment agent," in *Proceedings of the first international conference on Autonomous agents*, Marina del Rey, California, United States: ACM Press, 1997, pp. 30-38.

[8] J. Lester and B. Stone, "Increasing Believability in Animated Pedagogical Agents," in *The First International Conference on Autonomous Agents*, Marina del Rey, California, 1997, pp. 16-21.

[9] E. Andre, T. Rist, and J. Muller, "Integrating reactive and scripted behaviors in a life-like presentation agent," in *Proceedings of the second international conference on Autonomous agents*, Minneapolis, Minnesota, United States: ACM Press, 1998, pp. 261-268.

[10] J. Cassell et al., “Embodiment in conversational interfaces: Rea,” in *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, Pittsburgh, Pennsylvania, United States: ACM Press, 1999, pp. 520-527.

[11] R. McDonnell, F. Newell, and C. O’Sullivan, “Smooth movers: perceptually guided human motion simulation,” in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Aire-la-Ville, Switzerland, Switzerland, 2007, p. 259–269.

[12] P. S. A. Reitsma and N. S. Pollard, “Perceptual metrics for character animation: sensitivity to errors in ballistic motion,” *ACM Trans. Graph.*, vol. 22, no. 3, pp. 537-542, 2003.

[13] A. Hertzmann, C. O’Sullivan, and K. Perlin, “Realistic human body movement for emotional expressiveness,” in *ACM SIGGRAPH 2009 Courses*, New York, NY, USA, 2009, p. 20:1–20:27.

[14] N. Burtnyk and M. Wein, “Interactive skeleton techniques for enhancing motion dynamics in key frame animation,” *Commun. ACM*, vol. 19, no. 10, pp. 564-569, 1976.

[15] E. Catmull, “A system for computer generated movies,” in *Proceedings of the ACM annual conference - Volume 1*, Boston, Massachusetts, United States: ACM Press, 1972, pp. 422-431.

[16] ASF/AMC, “Acclaim ASF/AMC,” 1999. [Online]. Available: http://www.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/ASF-AMC.html.

[17] M. Garau, M. Slater, S. Bee, and M. A. Sasse, “The impact of eye gaze on communication using humanoid avatars,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*, Seattle, Washington, United States: ACM Press, 2001, pp. 309-316.

[18] P. Gebhard, “ALMA: a layered model of affect,” in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, The Netherlands: ACM Press, 2005, pp. 29-36.

[19] B. Tomlinson, “From linear to interactive animation: how autonomous characters change the process and product of animating,” *Comput. Entertain.*, vol. 3, no. 1, pp. 5-5, 2005.

[20] J. Lasseter, “Tricks to animating characters with a computer,” *SIGGRAPH Comput. Graph.*, vol. 35, no. 2, pp. 45-47, 2001.

[21] T. Porter and G. Susman, “On site: creating lifelike characters in Pixar movies,” *Commun. ACM*, vol. 43, no. 1, p. 25, 2000.

[22] K. Perlin and A. Goldberg, “Improv: a system for scripting interactive actors in virtual worlds,” in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, 1996, pp. 205-216.

[23] Z. Huang, A. Eliens, and C. Visser, “Implementation of a scripting language for VRML/X3D-based embodied agents,” in *Proceeding of the eighth international conference on 3D Web technology*, Saint Malo, France: ACM Press, 2003, pp. 91-100.

[24] N. I. Badler, M. S. Palmer, and R. Bindiganavale, “Animation control for real-time virtual humans,” *Commun. ACM*, vol. 42, no. 8, pp. 64-73, 1999.

[25] A. Watt and M. Watt, *Advanced Animation and Rendering Techniques Theory and Practice*. New York: Addison-Wesley, 1992.

[26] A. Bruderlin and T. W. Calvert, “Goal-directed, dynamic animation of human walking,” in *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, ACM Press, 1989, pp. 233-242.

[27] C. K. Liu and Z. Popovic;, “Synthesis of complex dynamic character motion from simple animations,” in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, San Antonio, Texas: ACM Press, 2002, pp. 408-416.

[28] Z. Popovic, “Controlling physics in realistic character animation,” *Commun. ACM*, vol. 43, no. 7, pp. 50-58, 2000.

[29] M. S. Geroch, “Motion capture for the rest of us,” *J. Comput. Small Coll.*, vol. 19, no. 3, pp. 157-164, 2004.

[30] B. Bodenheimer, C. Rose, S. Rosenthal, and J. Pella, “The Process of Motion Capture: Dealing with the Data,” *Computer Animation and Simulation ’97, Eurographics Animation Workshop*, no. 1997, 18. 1997.

[31] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. Wiley-Interscience, 2000.

[32] Y. Abe, C. K. Liu, and Z. Popovic;, “Momentum-based parameterization of dynamic character motion,” in *Proceedings of the 2004 ACM*

SIGGRAPH/Eurographics symposium on Computer animation, Grenoble, France: Eurographics Association, 2004, pp. 173-182.

[33] H-Anim, "H-ANIM specification for a standard humanoid," 2007. [Online]. Available: http://h-anim.org/Specifications/H-Anim1.1/.

[34] R. P. Paul, B. Shimano, and G. E. Mayer, "Kinematic Control Equations for Simple Manipulators," *IEEE Transactions on Systems, Man, and Cybernetics*, 1981.

[35] M. Fedor, "Application of inverse kinematics for skeleton manipulation in real-time," in *Proceedings of the 19th spring conference on Computer graphics*, Budmerice, Slovakia: ACM Press, 2003, pp. 203-212.

[36] M. Meredith and S. Maddock, "Adapting motion capture data using weighted real-time inverse kinematics," *Comput. Entertain.*, vol. 3, no. 1, pp. 5-5, 2005.

[37] R. Barzel and A. H. Barr, "A modeling system based on dynamic constraints," in *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, ACM Press, 1988, pp. 179-188.

[38] P. M. Isaacs and M. F. Cohen, "Controlling dynamic simulation with kinematic constraints," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM Press, 1987, pp. 215-224.

[39] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien, "Animating human athletics," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM Press, 1995, pp. 71-78.

[40] J. K. Hahn, "Realistic animation of rigid bodies," *SIGGRAPH Comput. Graph.*, vol. 22, no. 4, pp. 299-308, 1988.

[41] H. Zhiyong, N. M. Thalmann, and D. Thalmann, "Interactive human motion control using a closed-form of direct and inverse dynamics," in *The Second Pacific Conference on Computer Graphics and Applications, Pacific Graphics '94.*, Beijing, China., 1994.

[42] A. Witkin and M. Kass, "Spacetime constraints," in *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, ACM Press, 1988, pp. 159-168.

[43] H. Ko and N. I. Badler, "Animating human locomotion with inverse dynamics," *IEEE Computer Graphics and Animation*, vol. 16, no. 2, pp. 50-59, Mar. 1999.

[44] S.-H. Lee, E. Sifakis, and D. Terzopoulos, “Comprehensive biomechanical modeling and simulation of the upper body,” *ACM Transactions on Graphics (TOG)*, vol. 28, p. 99:1–99:17, Sep. 2009.

[45] C. K. Liu, “Dextrous manipulation from a grasping pose,” in *ACM Transactions on Graphics (TOG)*, New York, NY, USA, 2009, p. 59:1–59:6.

[46] A. Safonova, J. K. Hodgins, and N. S. Pollard, “Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces,” *ACM Trans. Graph.*, vol. 23, no. 3, pp. 514-521, 2004.

[47] P. Baerlocher and R. Boulic, “An Inverse Kinematic Architecture Enforcing an Arbitrary Number of Strict Priority Levels,” *The Visual Computer*, vol. 20, p. 402–417, 2004.

[48] A. Witkin, K. Fleischer, and A. Barr, “Energy constraints on parameterized models,” in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM Press, 1987, pp. 225-232.

[49] A. R. Conn and N. I. M. Gould, “Large-scale nonlinear constrained optimization: a current survey,” in *Algorithms for continuous optimization: the state of the art*, E. Spedicato, Ed. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1994, pp. 287-332.

[50] CMU, “CMU Graphics Lab Motion Capture Database,” 2008. .

[51] J. Barbic, A. Safonova, J.-Y. Pan, C. Faloutsos, J. K. Hodgins, and N. S. Pollard, “Segmenting motion capture data into distinct behaviors,” in *Proceedings of Graphics Interface 2004*, London, Ontario, Canada: Canadian Human-Computer Communications Society, 2004, pp. 185-194.

[52] L. Kovar and M. Gleicher, “Flexible automatic motion blending with registration curves,” in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, San Diego, California: Eurographics Association, 2003, pp. 214-224.

[53] M. Gleicher, H. J. Shin, L. Kovar, and A. Jepsen, “Snap-together motion: assembling run-time animations,” in *Proceedings of the 2003 symposium on Interactive 3D graphics*, Monterey, California: ACM Press, 2003, pp. 181-188.

[54] O. Arikan and D. A. Forsyth, “Interactive motion generation from examples,” in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, San Antonio, Texas: ACM Press, 2002, pp. 483-490.

[55] J. Lee and K. H. Lee, “Precomputing avatar behavior from human motion data,” in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Grenoble, France: ACM Press, 2004, pp. 79-87.

[56] H. P. H. Shum, T. Komura, and P. Yadav, “Angular momentum guided motion concatenation,” *Computer Animation and Virtual Worlds*, vol. 20, no. 2-3, pp. 385-394, Jun. 2009.

[57] R. Heck, L. Kovar, and M. Gleicher, “Splicing Upper-Body Actions with Locomotion,” *Computer Graphics Forum*, vol. 25, no. 3, pp. 459-466, Sep. 2006.

[58] S. Mutlu, “Motion Enriching Using Humanoide Captured Motions,” Master’s Thesis, Polytecnic University of Catalunya, 2010.

[59] M. da Silva, F. Durand, and J. Popović, “Linear Bellman combination for control of character animation,” in *ACM Transactions on Graphics (TOG)*, New York, NY, USA, 2009, p. 82:1–82:10.

[60] S. J. Lee and Z. Popović, “Learning behavior styles with inverse reinforcement learning,” in *ACM Transactions on Graphics (TOG)*, New York, NY, USA, 2010, vol. 29, p. 122:1–122:7.

[61] J. M. Wang, D. J. Fleet, and A. Hertzmann, “Optimizing walking controllers for uncertain inputs and environments,” in *ACM Transactions on Graphics (TOG)*, New York, NY, USA, 2010, p. 73:1–73:8.

[62] K. Shoemake, “Animating rotation with quaternion curves,” in *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, ACM, 1985, pp. 245-254.

[63] K. Yamane, J. J. Kuffner, and J. K. Hodgins, “Synthesizing animations of human manipulation tasks,” *ACM Trans. Graph.*, vol. 23, no. 3, pp. 532-539, 2004.

[64] J. Pan, L. Zhang, M. C. Lin, and D. Manocha, “A hybrid approach for simulating human motion in constrained environments,” *Comput. Animat. Virtual Worlds*, vol. 21, p. 137–149, May. 2010.

[65] B. J. H. van Basten, P. W. A. M. Peeters, and A. Egges, “The step space: example-based footprint-driven motion synthesis,” *Computer Animation and Virtual Worlds*, vol. 21, p. 433–441, May. 2010.

[66] N. Pronost, A. Sandholm, and D. Thalmann, “Correlative joint definition for motion analysis and animation,” *Computer Animation and Virtual Worlds*, vol. 21, p. 183–192, May. 2010.

[67] F. Aurenhammer, “Voronoi diagrams—a survey of a fundamental geometric data structure,” *ACM Computing Surveys (CSUR)*, vol. 23, p. 345–405, Sep. 1991.

[68] K. Pullen and C. Bregler, “Animating by Multi-Level Sampling,” in *Proceedings of the Computer Animation*, Washington, DC, USA, 2000, p. 36–.

[69] M. Brand and A. Hertzmann, “Style machines,” *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, p. 183–192, 2000.

[70] H. J. Shin and J. Lee, “Motion synthesis and editing in low-dimensional spaces: Research Articles,” *Comput. Animat. Virtual Worlds*, vol. 17, p. 219–227, Jul. 2006.

[71] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. Springer, 2002.

[72] A. Frank and A. Asuncion, *UCI Machine Learning Repository*. University of California, Irvine, School of Information and Computer Sciences, 2010.

[73] K. T. Ge, “Solving Inverse Kinematics Constraint Problems for Highly Articulated Models,” University of Waterloo, 2000.

[74] Q. Gu, J. Peng, and Z. Deng, “Compression of Human Motion Capture Data Using Motion Pattern Indexing,” *Computer Graphics Forum*, vol. 28, no. 1, pp. 1-12, Mar. 2009.

[75] O. Arikan, “Compression of motion capture databases,” in *ACM SIGGRAPH 2006 Papers*, Boston, Massachusetts: ACM, 2006, pp. 890-897.

[76] Z. Li, Y. Deng, and H. Li, “Generating Different Realistic Humanoid Motion,” in *Advances in Artificial Reality and Tele-Existence*, vol. 4282, Springer Berlin / Heidelberg, 2006, pp. 77-84.

[77] J. Tilmanne and T. Dutoit, “Expressive gait synthesis using PCA and Gaussian modeling,” *Proceedings of the Third international conference on Motion in games*, p. 363–374, 2010.

[78] M. P. Johnson, *Exploiting Quaternions to Support Expressive Interactive Character Motion*. Massachusetts Institute of Technology, 2003.

[79] J. Min, Y.-L. Chen, and J. Chai, “Interactive generation of human animation with deformable motion models,” *ACM Transactions on Graphics (TOG)*, vol. 29, p. 9:1–9:12, Dec. 2009.

[80] S. R. Carvalho, R. Boulic, and D. Thalmann, “Interactive low-dimensional human motion synthesis by combining motion models and PIK,” *Comput. Animat. Virtual Worlds*, vol. 18, no. 4-5, pp. 493-503, 2007.

[81] T. Grudzinski, “Exploiting Quaternion PCA in Virtual Character Motion Analysis,” *Proceedings of the International Conference on Computer Vision and Graphics: Revised Papers*, p. 420–429, 2009.

[82] L. Ren, A. Patrick, A. A. Efros, J. K. Hodgins, and J. M. Rehg, “A data-driven approach to quantifying natural human motion,” in *International Conference on Computer Graphics and Interactive Techniques*, Los Angeles, California, 2005.

[83] F. Multon, R. Kulpa, L. Hoyet, and T. Komura, “Interactive animation of virtual humans based on motion capture data,” *Computer Animation and Virtual Worlds*, vol. 20, p. 491–500, Sep. 2009.

[84] S. Ishigaki, T. White, V. B. Zordan, and C. K. Liu, “Performance-based control interface for character animation,” in *ACM Transactions on Graphics (TOG)*, New York, NY, USA, 2009, vol. 28, p. 61:1–61:8.

[85] S. M. LaValle, *Planning Algorithm*. Cambridge University Press, 2006.

[86] David C. Lay, *Linear Algebra and Its Applications*, 2nd ed. Addison Wesley, 2000.

[87] S. R. Buss, “Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares Methods,” 2004. [Online]. Available: http://groups.csail.mit.edu/drl/journal_club/papers/033005/buss-2004.pdf.

[88] S. Buss and J.-S. Kim, “Selectively Damped Least Squares for Inverse Kinematics,” *Journal of Graphics, GPU, & Game Tools*, vol. 10, no. 3, pp. 37-49, Jan. 2005.

[89] D. Cosker, S. Paddock, D. Marshall, P. L. Rosin, and S. Rushton, “Toward Perceptually Realistic Talking Heads: Models, Methods and McGurk,” *ACM Transactions on Applied Perception*, vol. 2, no. 3, pp. 270-285, Jul. 2007.

[90] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popovic, “Style-based inverse kinematics,” in *ACM SIGGRAPH 2004 Papers*, Los Angeles, California: ACM, 2004, pp. 522-531.

[91] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C book set: Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, 1992.

[92] Harold Hotelling, "Multivariate Quality Control," in *Techniques of Statistical Analysis*, New York: McGraw-Hill, 1947, pp. 111-184.

[93] J. Chai and J. K. Hodgins, "Performance Animation from Low-dimensional Control Signals," *ACM TRANSACTIONS ON GRAPHICS*, vol. 24, p. 686--696, 2005.

[94] S. E. M. Jansen and H. Welbergen, "Methodologies for the User Evaluation of the Motion of Virtual Humans," in *Intelligent Virtual Agents*, vol. 5773, Z. Ruttkay, M. Kipp, A. Nijholt, and H. H. Vilhjálmsson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 125-131.