Real-Time Animation of Walking and Running using Inverse Kinematics

Ying Ying She

A Thesis

in

The Department of Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements for the Degree of Master of Computer Science at Concordia University

Montreal, Quebec, Canada

March 2006

Your file Votre référence
ISBN: 0-494-14335-5
Our file Notre référence
ISBN: 0-494-14335-5

Abstract

Real-Time Animation of Walking and Running using Inverse Kinematics

Ying Ying She

Computer animation technology is a rapidly developing topic in computer science. Realistic human motion, such as walking and running, is an important part of computer animation. However, the simulation of walking and running in articulated human figure is difficult, especially in real time simulation. Fortunately, recently, Inverse kinematics algorithms provide an approach to simulate human motion in articulated figures.

In this thesis, I analyzed Jacobian based Inverse Kinematics algorithms that allow theses methods to be used as an efficient real-time inverse kinematics simulator. I describe the implementation of a human motion simulator, which mainly focuses on using inverse kinematics algorithm to simulate (wounded) walking and (wounded) running motion in an articulated body. With the assistance of inverse kinematics algorithms, the animator merely gives the desired location of certain chosen points on the body and relies on the algorithm to automatically compute a set of joint angles that satisfy the end-effectors constraints. This kinematics simulation approach can be used for real time animation in articulated figure without any motion capture data. In addition, the thesis shows that it is also possible to simulate human animation by using purely mathematical techniques without physical considerations.

Acknowledgement

I would like to take this opportunity to give grateful thanks to my supervisor, Professor Peter Grogono, for his supports and guidance throughout this research project. This work would not have been completed without his concise instructions and flexible supervision.

I would also like to express appreciation to my husband Jian Lin for his endless support and patience; my parents for setting high expectations and being consistently serious about my education.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer animation technology is a rapidly developing topic in computer science. It has been used in video games, feature movies, digital simulation industries. Research on articulated models is well developed in the fields of robotics and mechanical engineering. Computer graphics researchers simulate arbitrarily linked models and control them in arbitrarily complex environments. Inverse kinematics algorithms play a key role in the computer animation and simulation of articulated figures. This thesis presents an approach that uses inverse kinematics algorithms for computer animation. Before further discussion of the motivation of this thesis, I will first provide a brief overview of methods for creating human figure animations.

1.1 Computer Animation

There are three widely used methods to simulate feature animation: key frame, motion capture, and kinematics simulation.

The key frame technique is the oldest technique for computer animation and it was first used for 2D figure animation. It uses a mixture of pre-recorded or pre-scripted sequences that are blended together to create the illusion of continuous character motion.

This technique provides good results for making animation films. With sufficient study of animated objects and the environments in which they interact, animators can create convincing animation sequences in films using the key frame animation technique.

The motion capture technique was first used in the TV and film industry. Later, the game industry embraced the technology as a routine method for animating figure characters. Nowadays, almost all games employ Motion Capture technology to drive figure animation.

Kinematics simulation technology is a potentially powerful tool to create figure animation. It has developed very rapidly in recent years. Various approaches based on dynamics/kinematics algorithms have been adopted in computer animation. Compared with key frame and motion capture methods, kinematics/dynamics simulation has the advantage of creating animation either off-line or in real time.

1.2 Motivation

Presently, animators use these three kinds of animation technologies in game and real-time figure animation. There are problems associated with key frame animation and motion capture animation. Key frame animation provides a fixed sequence of animation; any interaction, even for one frame of the whole sequence, will lead to incorrect visual results. There are problem with motion capture data, too. For example, applying motion data captured from one person to a virtual person of a different size does not work well. If a transformation (rotation angle) applied to one body is applied to another body with different length of limb, the end position for the limb will be different. As a result, real-time animation technology based on kinematics/dynamics simulation is better than the other two animation technologies.

Realistic human motion is an important part of computer animation. Walking and running is a seemingly simple behavior for people. However, the simulation of walking and running is difficult, especially in real time simulation. The inherent instability of walking makes it one of the hardest motions to model, especially when the conditions underfoot are unpredictable.

In this thesis, I describe the implementation of a human motion simulator, which mainly focuses on using inverse kinematics (IK) algorithm to simulate walking and running motion in an articulated body. This kinematics simulation approach can be used for real time animation. With the assistance of inverse kinematics algorithms, the animator merely gives the desired location of certain chosen points on the body and relies on the algorithm to automatically compute a set of joint angles that satisfy the end-effectors constraints.

1.3 Thesis Organization

The remainder of this thesis is structured as follow. After reviewing previous related work and background information in Chapter 2, I discuss the modeling in Chapter 3. Chapter 4 describes four inverse kinematics algorithms used in this thesis. Chapter 5 demonstrates the practicing system of inverse kinematics algorithms discussed in Chapter 4. Chapter 6 provides the implementation detail of this simulation system. Finally, the Chapter 7 presents conclusion and future work.

Chapter 2

Background and Related Work

2. 1 Kinematics and Dynamics

Kinematics is the science of motion [JOCH 00]. Kinematics is restricted to a pure geometrical description of motion by means of position, orientation, velocity, and acceleration [JOCH 00]. Kinematics does not consider physical laws of the real world. Kinematics methods include inverse kinematics and forward kinematics. In computer animation, people use kinematics to simulate the motion of real world objects. There are several forward and inverse kinematics methods, originally derived for robotics applications.

Traditional computer animation, using professional animators to manually create character or skeleton animation, required large amounts of processing time to complete. Moreover, with the application of computer animation to different fields, people demanded more detail and 3D characters that were more controllable and better looking. Obviously, the conventional way of making 3D animation was inadequate. Forward and inverse kinematics techniques have become alternative techniques for speeding up 3D animation processing.

Dynamics is a more detailed theory of motion that takes into account the effect of force on bodies. In contrast to kinematics, it takes into account physical laws. Dynamics can be, and, in fact, is used as a theoretical foundation for simulation and, in particular, for graphical simulation. Forward dynamics indirectly controls the motion of object, and requires large amounts of calculation. Inverse dynamics can be used to compute torque and force problems. However, there is a large amount of computation required to solve inverse force and torque problems and it is therefore difficult to create fast simulations. Kinematics techniques can be used to simulate some dynamic problem when forces and torques do not need to be calculated. In this way, kinematics can be used to save time in 3D animation processing.

2.2 Forward Kinematics Animation

Usually, we use matrices to describe the geometry of world space or local space. Forward kinematics refers to the process of computing world matrices based on the rotation angles and /or translations of joints. Chapter 3 provides a more detailed description of joints.

Forward kinematics explicitly sets the position and orientation of each segment at a specific frame time by specifying the joint angles for each joint [GE 00, 4]. In order to create a pose for an articulated body, this means directly setting the rotation angles and orientation at selected joints and applying a global translation at the root joint. To avoid doing the complete calculation in each frame, we can specify a series of keyframes for different poses, and calculate joint parameters between keyframes. The articulated body can then be animated with a reasonable amount of computation.

Linear interpolation between keyframes can lead to jerky animation. Higher order interpolation methods, such as piecewise splines, can provide continuous velocity and acceleration, and hence smoother translation between and through keyframes. While forward kinematics combined with a simple interpolation scheme may suffice for animating simple objects, it is not really up to the task of animating articulated figures [WELM 93, 8]. The reason is the structure of articulated figures. Usually, the articulated figure is constructed by simple objects. The animation of the whole articulated figure is animations of every simple object in the articulated body. The interpolation scheme is a way to indirectly control the animation of simple objects in articulated body by specifying rotation angles. However, the rotation of every simple object in the articulated body has interrelationship. We can't simply specify each simple object's animation to control the animation of the whole articulated figure.

2.3 Inverse Kinematics Animation

Inverse kinematics is an attractive alternative to forward kinematics, since animators can specify the position of end effectors directly, using inverse kinematics algorithms to calculate rotation angles at specific joints. The inverse kinematics technique was first studied in robotics, but has recently been used in computer animation as well. In this approach, an inverse kinematics problem is cast either into a system of nonlinear equations or into an optimization problem which can be solved using an iterative numerical algorithm [TOLA 00, 2]. Zhao and Badly [ZHAO, 94] formulated the inverse kinematics problem of a human figure as a constrained nonlinear optimization problem. Rose [ROSE, 96] extends this formulation to generate seamless and dynamically plausible transitions between motion segments by combining a hierarchical curve fitting technique with an inverse kinematics solver. In current games, for example, inverse kinematics is mostly used in simple ways, such as controlling the player's direction of gaze, aiming guns, etc.

Inverse kinematics provides better higher-level control over joint hierarchies than simple forward kinematics. For example, moving the limbs of a skeleton becomes much more manageable [WELM 93, 9]. Animators combined inverse kinematics and motion capture data for articulated body applications. Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popovic [GROC 04] present an inverse kinematics system based on a probability model of human poses. Given a set of arbitrary algebraic constraints, their system can produce the most likely pose satisfying those constraints, in real-time. Inverse kinematics theory also can be used to determine which objects in the environment can be reached. Deepak Tolani, Ambarish Goswami, and Norman I. Badler [TOLA 00, 1] use a combination of numerical and analytical inverse kinematics methods to solve a variety of problems including position, orientation, and aiming constraints.

2.4 Animation Data Processing

2.4.1 Animation Data Generation

Due to the popularity of motion capture system, many researchers have started to work on topics such as editing and synthesizing motions using motion capture data [HO 05, 163]. There are two general ways of obtaining animation data for animation characters.

The first way is to obtain data by recording the motion of a live subject. Motion capture is an animation technique that uses sensors attached to a real actor's body; as the real actor moves, the sensors send the data to the computer so that the computer character mimics the movements and create corresponding Motion capture data. The second way is to simulate the motion by using some techniques, such as key-framing, inverse kinematics or inverse dynamics algorithms. The first technique usually has less computational expense than the second. However, one of the main disadvantages in using Motion capture data is the problem of retargeting. The Motion capture data retargeting is a process of mapping Motion capture data to fit onto a new 3D object with accurate geometry; in general, the new 3D object does not have the same geometric features as the Motion capture data which is derived from the old 3D object. Simply mapping Motion capture data to the new 3D object can cause poor animation performance. In order to solve the retargeting problem, there are some new approaches which rescale Motion capture data for the new 3D object in order to let the new 3D object have same animation aspects as the source 3D object. The general idea of these two new approaches is to combine these two techniques. First, researchers loaded the motion capture data and then used simulation techniques to adjust the data to the dimensions of the new object. And then, the experimenters mapped the adjusted motion data onto the new objects. Inverse kinematics algorithms are the most useful simulation technique in animation.

2.4.2 Motion Capture Data Format

Motion capture data is stored in files with formats such as BVH, C3D, CSM, and MOT. Most of the Motion capture data format data is represented in a hierarchical manner, and requires matrix calculations to demonstrate the motion decoding.

For example, one of the popular Motion capture data format is the BVH file format which was originally developed by Biovision, a motion capture services company, as a way to provide motion capture data to their customers. The name BVH stands for BioVision Hierarchical data. It consists of two parts; the first part describes the body hierarchy and initial pose information, and the second part is motion section which describes the channel data for each frame. In each bone definition, BVH file format provides information such as bone name, offset value (DOFs), position relationship respect to parent bone. And for bones serving as end effectors, BVH format uses the keyword "end Site" to identify them.

File Extension	Associated Company / Description	File Format Reference
ASC	Ascension	NO LINK
ASF & AMC	Acclaim	http://www.darwin3d.com/gamedev/acclaim.zip
ASK & SDL	BioVision/Alias	NO LINK
BVA & BVH	BioVision	http://www.biovision.com/bvh.html
BRD	LambSoft Magnetic Format	http://www.dcs.shef.ac.uk/ ~mikem/fileformats/brd.html
C3D	Biomechanics, Animation and Gait Analysis	http://www.c3d.org/c3d_format.htm
CSM	3D Studio Max, Character Studio	http://www.dcs.shef.ac.uk/ ~mikem/fileformats/csm.html
DAT	Polhemous	NO LINK
GTR, HTR & TRC	Motion Analysis	http://www.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/ {HTR.html, TRC.html}
MOT & SKL	Acclaim-Motion Analysis	(Under Development - http://www.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/SKL-MOT.html)

Table 2. 1 Motion capture file formats and references [MERE 02, 3]

In this thesis, I use a tree hierarchy similar to the Motion capture data format to represent the human structure of bones. The configuration of each joint (bone) is similar to the bone configuration in BVH format. In addition, in order to perform inverse kinematics algorithms, some additional joints, tree configurations, and operations are defined in the thesis. I will introduce these in the Chapter 3. In addition, all animation data used in this thesis was obtained from real-time inverse kinematics calculations, not from a Motion capture device. The animation data processing required for this thesis is described in Chapter 4.

Chapter 3

Modeling

3.1 Articulated Human Body

An articulated body consists of a set of rigid objects, called links, joined together by joints. Simple kinds of joints include revolute (rotational) and prismatic (translational) joints. It is also possible to work with more general types of joints, and thereby simulate articulated objects. The rotation joint is the most common type of joint, and its configuration is described by a scalar angle value.

In this thesis, I simulated a articulated human body as a set of links and joints. Each joint performs rotation with one degree of freedom (1 DOF). The human skeleton is represented as an array of nodes. Each node represents a joint formed by a pair of links representing bones. The whole body is modeled with n joints, and the scalar θ_j ($1 \leq j < n$) is called a "joint angle". The whole body's complete configuration for inverse kinematics animation is a set of scalars $\theta_1,\ldots,\theta n$ giving the angular position at each joint.

The main properties of each Node are:

- Size
- Joint or End Effector

- Global Position and Rotation Axis
- Parent node, Left child node and Right sibling node
- Rotation angle , Minimum and Maximum rotation angles in Radian
- Joint sequence number
- Attachment position

All of the nodes are arranged in a tree structure. The root of the tree is a node close to the hip, which we consider to be the centre of the body for the purposes of animation. The tree structure sets up an explicit hierarchy. In the tree structure, the parent of a node is the node which is attached to that node and will cause that node to move if it moves. The only node which does not have a parent is the root node. The root node is not really a node in a physical sense but rather it is used as a convenient way of moving the entire skeleton with a single translation or rotation.

The advantage of using a tree hierarchy to represent the skeleton is that it is easy to apply joint limits to prevent unfeasible motion of the skeleton. In addition, based on the tree structure, we can explicitly get the parent or child node of each node. We can also use the tree structure to obtain the Jacobian matrix, J, which describes the motion. Since the inverse kinematics algorithms should loop over the whole body to find the corresponding change in rotation angle at each joint, the tree structure is a perfect match for the calculation process of the inverse kinematics algorithms.

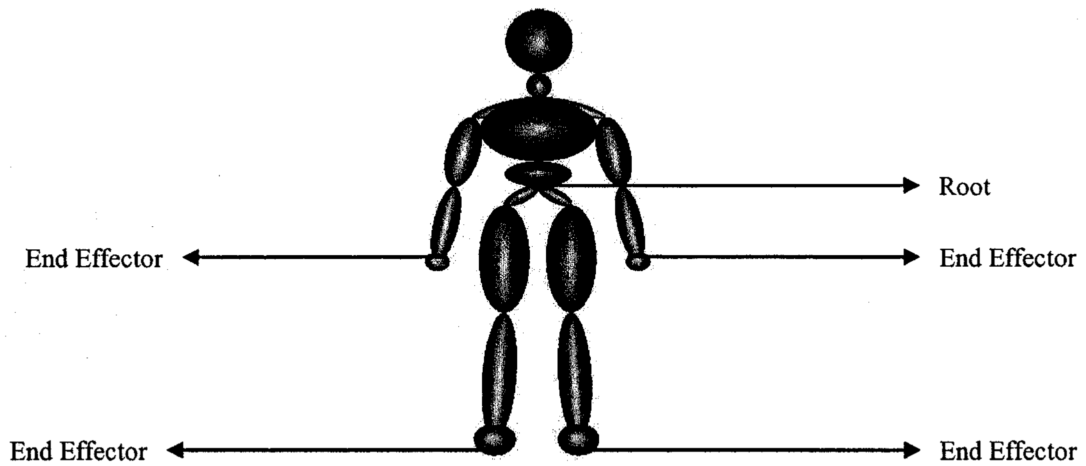


Figure 3. 1 Articulated human body, with 4 end-effectors and tree structure

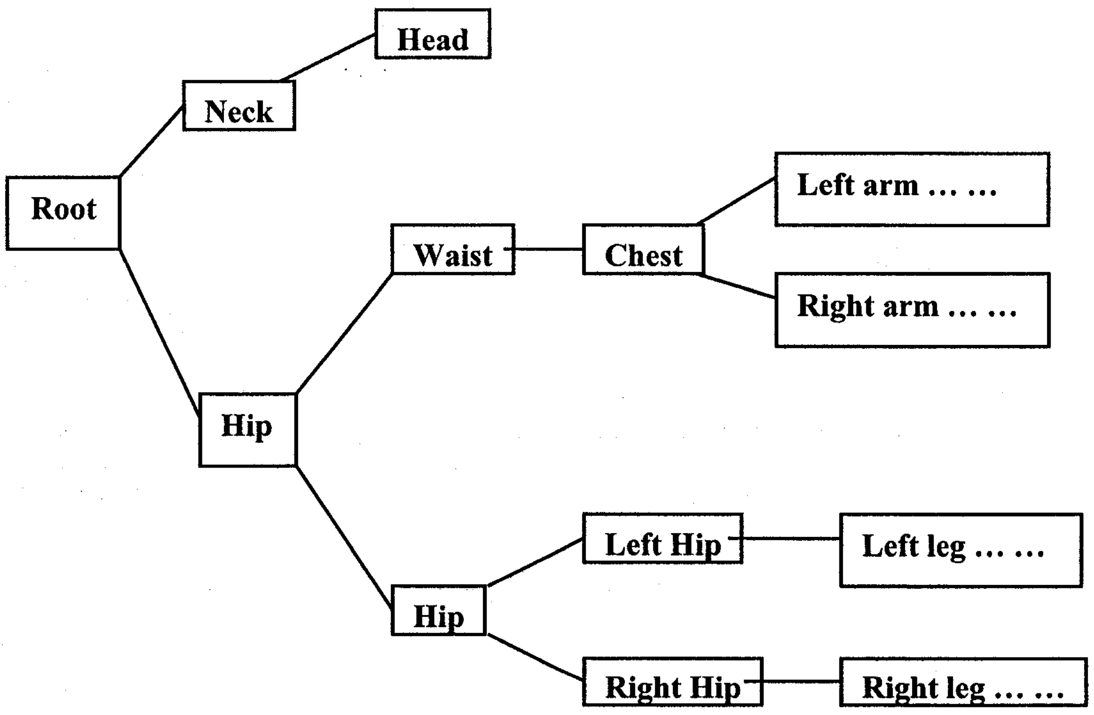


Figure 3. 2 Tree structure of the articulated human body

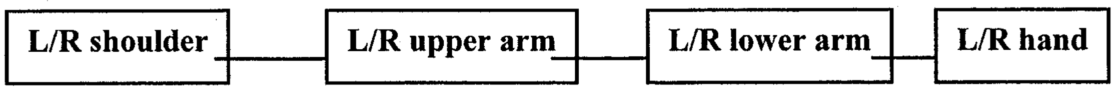


Figure 3. 3 Tree structures of left/ right arms

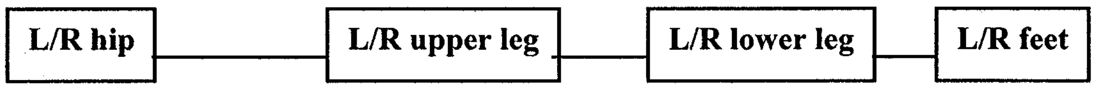


Figure 3. 4 Tree structures of left/ right legs

3.2 End Effectors and Targets

The joint (bone) at the leaf end of a chain of links is called an "end effector". I set the end effector as a joint with position and orientation. In order to use the inverse kinematics algorithm, I defined left hand, right hand, left foot, and right foot as end effectors.

Figure 3. 5 Single joint

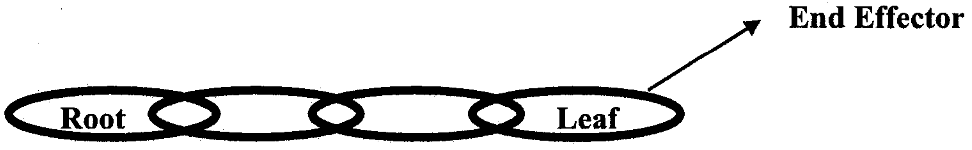


Figure 3. 6 Linear chain of links with 1 DOF rotational joints

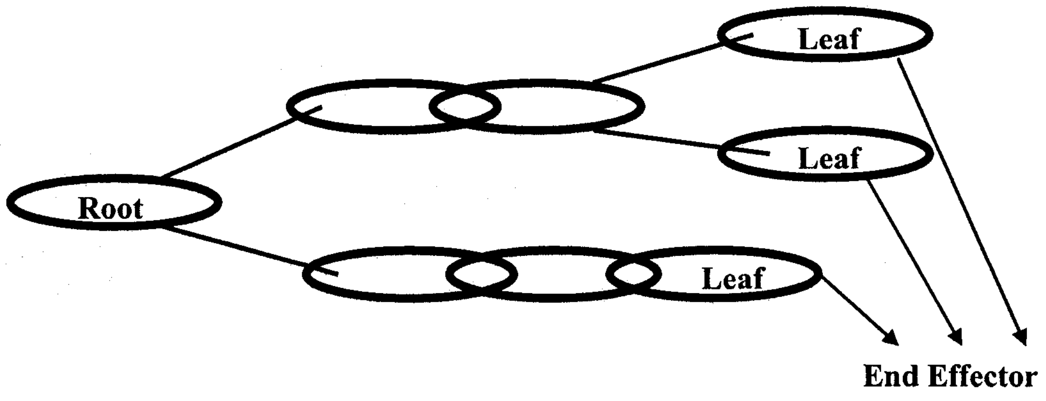


Figure 3. 7 Tree structure links with 1 DOF rotational joints

Also, there are four targets corresponding to these four effectors when we calculate the movement in 3D space. The animation of the articulated body is controlled by targets' positions. Usually, in Jacobian inverse kinematics algorithms, end effectors are tracking targets' positions. I used two ways to control the Jacobian formulation. The first way tries to move targets towards the end effectors; the second way tries to move end effectors towards the targets' positions.

If there are k end effectors, there are k targets. The positions of k end effectors are denoted as a column vector $\vec{s} = (s_1,...,s_k)^T$, where s_i is the position of ith end effector. Also, the k targets are denoted as a column vector $\vec{t} = (t_1,...,t_k)$, where t_i is the position of the ith target position. Our model uses 4 end effectors and 4 targets for the human body.

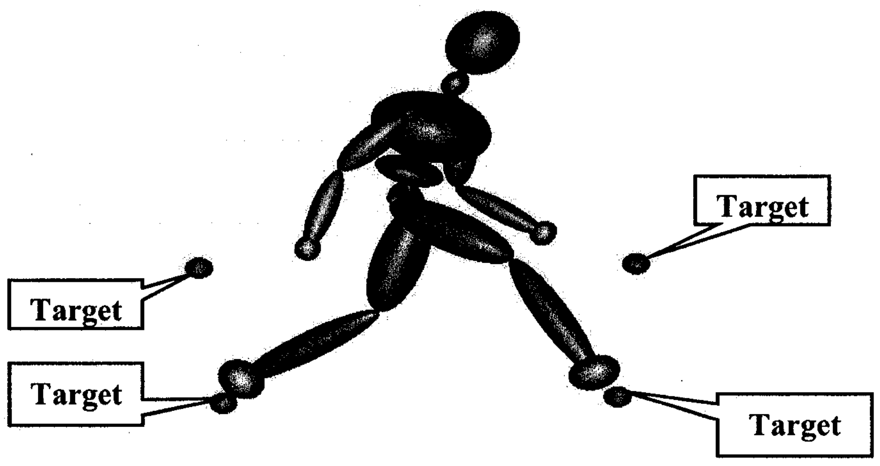


Figure 3. 8 Targets for running

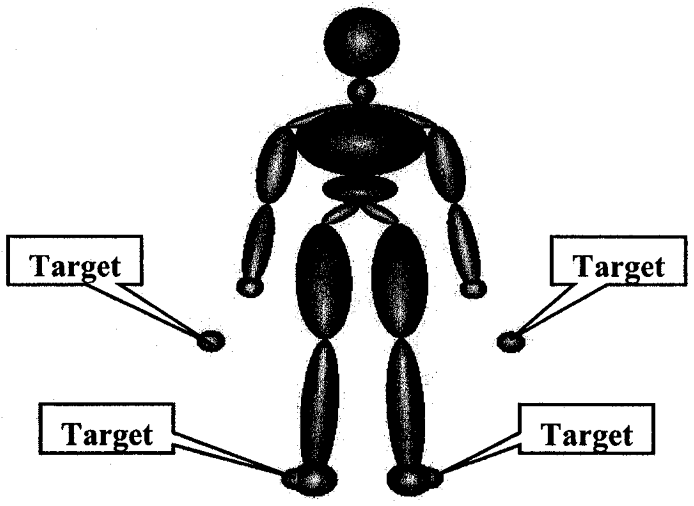


Figure 3. 9 Targets for walking

Chapter 4

Inverse Kinematics Algorithms

4.1 Problems with Forward Kinematics

In computer-generated films, most simplified characters are represented by skeletons structure in order to be manipulated motion by kinematics theories. Usually, skeleton structure of these characters are animated by forward kinematics using data which is either captured from live performers or interpolated from keys designed by animators. See Chapter 2 for definition of forward kinematics.

The problem with forward kinematics is that it is time consuming and difficult to control. Forward kinematics gives user absolute control on the articulated body; however, it can be time-consuming if every movement requires that user should select and rotate multiple joints. Usually, forward kinematics can control the position of each segment only indirectly by specifying the joint angles for each joint; the result is unpredictable behavior of the animated skeleton. In contrast, inverse kinematics can directly specify the position of end effectors and can use inverse kinematics algorithms to calculate the rotation angle of each joint.

In symbols, we can express Forward Kinematics by the equation $P = f(\theta)$ and Inverse Kinematics by the equation $\theta = f^{-1}(P)$.

4.2 Overview of Inverse Kinematics Solutions

For inverse kinematics (IK) solutions, $f(\theta)$ may be relatively easy to evaluate while $f^{-1}(P)$ usually isn't. There may be several solutions for θ , or there may be no solution. As a result, there are many different approaches to solving IK problems. General solutions of inverse kinematics are usually either analytical methods or numerical methods.

Analytical methods use mathematical techniques to obtain an exact solution by directly inverting the forward kinematics equations. Analytical solvers provide explicit solution to calculate the generalized coordinates from the position information [HO 05, 163]. Numerical methods use approximation and iteration to converge on a solution; they tend to be more expensive, but are far more general purpose. For complex systems, hybrid methods that combine analytical and numerical techniques often work best. Numerical solvers linearize the relationship of the generalized coordinates and the 3D coordinates of the end effectors around the current posture to obtain the IK solutions for new 3D coordinates of the end effectors close to the current position / orientation [HO 05, 164]. In this chapter, we discuss numerical inverse kinematics technique based on Jacobian matrix algorithms.

4.3 The Jacobian Matrix

In this thesis, we use closed-form solutions to solve inverse kinematics problems. The Jacobian Matrix can provide a linear approximation to the function that we have to iterate the process. So we can assume a linear relationship between joints movement and end effectors movement based on the movement of end effectors. So, for one end effector, we can have one linear equation for all the links that linked to the end effector. For an articulated rigid body, the best method may be to convert the system of linear equations into a matrix equation, and then apply matrix manipulation operations to solve this equation. Consequently, using the Jacobian matrix is an effective way to solve inverse kinematics problems.

A Jacobian Matrix *J* is a vector derivative with respect to another vector. If we have *f(x)*, a vector function of a vector argument, the Jacobian is a matrix of partial derivatives for each combination of components of the vectors. The Jacobian is usually written as *J(f, x)*.

$$J(\mathbf{f},\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_M}{\partial x_1} & \cdots & \cdots & \frac{\partial f_M}{\partial x_N} \end{bmatrix}$$

Consider the example of a 2D arm with 1-DOF rotation joint,

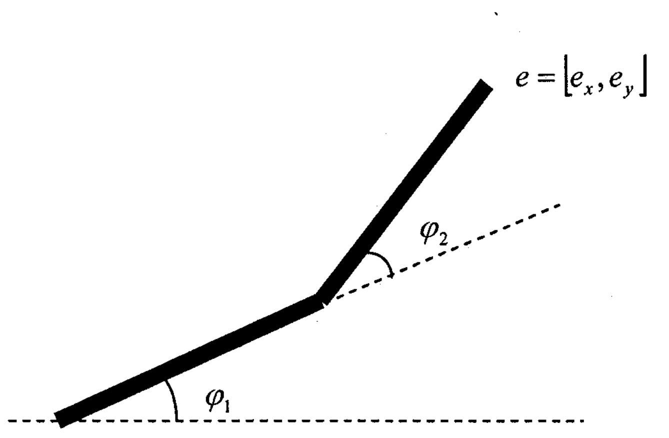


Figure 4. 1 a 2-D arm with rotation joints

The Jacobian Matrix J is

$$J(e,\varphi) = \begin{bmatrix} \frac{\partial e_x}{\partial \varphi_1} & \frac{\partial e_x}{\partial \varphi_2} \\ \frac{\partial e_y}{\partial \varphi_1} & \frac{\partial e_y}{\partial \varphi_2} \end{bmatrix}$$

The Jacobian matrix $J(e,\varphi)$ shows how e_x and e_y vary with respect to each joint angle in the body. In this 2D example, the end effector is denoted as $e = \lfloor e_x, e_y \rfloor$. With the movement of *e*, the joint rotation angles change and the Jacobian matrix J changes accordingly.

4.3.1 End Effectors Jacobian Matrix

For the inverse kinematics algorithm, J is a multidimensional derivative relating the end effectors movement to joints movement. Based on the information from the articulated body tree, the Jacobian matrix is a $m \times n$ matrix, where n is the number of joints of the body, and $m = 3k$, where k is the number of end effectors of the body.

As mentioned in Chapter 3, end effector positions are denoted as $\vec{s} = (s_1,...,s_k)^T$ and target positions are denoted as $\vec{t} = (t_1,...,t_k)$, which k is the number of end effectors. The end effector positions $\vec{s} = \vec{s}(\theta)$ are functions of joint angle vector $\theta = (\theta_1,...,\theta_n)^T$. In order to solve inverse kinematics problems, we have to find the value for $\theta = (\theta_1,...,\theta_n)^T$ from $t_i = s_i(\theta)$, for i=1... k. Unfortunately, there may not be always a unique solution. (To see why, note that you can hold your hand in a fixed position while moving your arm.) The Jacobian matrix $J(s,\theta)$ varies over the domain of all possible values of θ. For any given joint position vector s, we can explicitly compute the individual components of the Jacobian matrix. In this case, we can use iterative methods to linearly approximate the function s_i using Jacobian matrix by

$$J(s,\theta) = (\frac{\partial s_i}{\partial \theta_j})_{i,j}$$

We can take a geometric approach to computing the Jacobian matrix. If the jth joint is a rotational joint with a single degree of freedom, the joint angle is θ_j, the position of the joint is P_j, and v_j is a unit vector pointing along the current joint rotation axis; we can calculate the corresponding entry for this Jacobian matrix J as

$$\frac{\partial s_i}{\partial \theta_j} = v_j \times (s_i - p_j)$$

In addition, if ith end effector is not affected by the movement of the jth joint, the entry of $\frac{\partial s_i}{\partial \theta_j} = 0$. In this paper, all rotation joints in the rigid body are based on 1DOF.

In inverse kinematics animation, we are looking for the update value of joint angle $\Delta\theta$ to increment the joint angle by $\theta_{new} = \theta_{old} + \Delta\theta$. The change of end effector positions is caused by the change of joint angles, which is $\Delta\vec{s} \approx J\Delta\theta$. We use $e_i = t_i - s_i$ to denote the desire position change of ith end effector in order to track the target movement. So $\Delta\vec{s}$ is approximately equal to $\vec{e}$. As a result, we can use this equation

$$\vec{e} = J\Delta\theta,$$

for choosing $\Delta\theta$ in order to update joint angles. Given some desired incremental change in end effector configuration $\vec{e}$, we can get the approximate update of joint angle θ by computing

$$\Delta\theta = J^{-1}\vec{e}.$$

4.3.2 Inverse of Jacobian Matrix and Singularity

The previous section reduced the inverse kinematics problem to the form $\Delta\theta = J^{-1}\vec{e}$; we can obtain $\Delta\theta$ by calculating the inverse of Jacobian Matrix J^{-1}. However, in most cases, Jacobian Matrix J will not be invertible since it is non-square. For example, if we have a 4-joints linear chain, and only one end effector for this chain. The Jacobian Matrix for this chain is a 3×4 Matrix, which is not a square matrix.

Even if J is invertible; calculating J^{-1} can lead to a poor result since the matrix may be singular or almost singular. In Figure 4.2, the target moves along the direction along the x axis, and the 3-jointed arm track the target position. Since the arm is fully extended, as a result, an incremental change to any joint rotation angle will be in approximately the same movement of the end effector along x axis. Physically, this means that no infinitesimal change in joint angles can effect a change in the horizontal position of the end effector. [BUSS 03, 314] Mathematically, for Jacobian matrix, the first row for the 3-jointed arm will consist entirely of zeroes. Consequently, the Jacobian matrix is almost singular and cannot be accurately inverted.

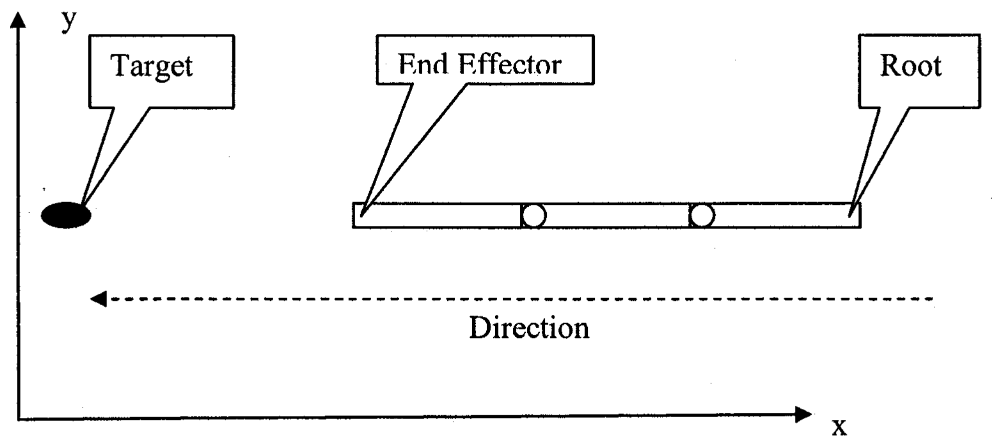


Figure 4. 2 An example of singular configuration

For the remainder of this chapter, we discuss several strategies for choosing solutions which avoid computing the J^{-1} in order to get $\Delta\theta$ joint angles' animation update.

4.4 Inverse Kinematics Algorithms

Different inverse kinematics algorithms have been discussed and used in recent years. Fedor [FEDOR, 03] present three of such methods: Algebraical method based on limbs positioning; iterative optimization method based on Jacobian pseudo-inversion; and heuristic CCD iterative method. In this section, I discuss four different approaches of Jacobian Inverse Kinematics algorithms. They are Jacobian Transpose Method, Pseudoinverse Method, Damped Least Squares Method (DLS), and Selectively Damped Least Squares Method (SDLS).

4.4.1 Jacobian Transpose Method

As discuss in section 4.3.2, sometimes Jacobian inverse method works poorly, because the Jacobian matrix J is not square or invertible. The Jacobian Transpose Method uses the Jacobian transpose matrix J^T to avoid the need for the inverse matrix.

In the Jacobian transpose method, we set $\Delta\theta$ equal to

$$\Delta\theta = \alpha J^T \vec{e}$$

for some appropriate α. [BUSS 04, 7] It is much faster to calculate the transpose than to calculate the inverse of J or the pseudo-inverse of J.

The way to choose the value of α is to try to minimize the new value of the error vector $\vec{e}$ after the update. [BUSS 04, 7] To do this, we assume that the change in end effectors position will be exactly $\alpha J^T \vec{e}$, and choose α so as to make this value as close as possible to $\vec{e}$. In this paper, the scalar α is calculated from

$$\alpha = \frac{\Delta s \bullet \Delta t}{2 \times (|\Delta t|)^2}$$

The α value will be changed within each step of calculating $\Delta\theta$ and J^T .

The Jacobian transpose method has the effect of localizing the computations. With the Jacobian transpose method, we can just loop through each joint and compute the change to that joint angle directly. In addition, Jacobian transpose method work acceptably with only one end effector for an articulated body.

4.4.2 Pseudoinverse Method

The pseudoinverse method is a technique for finding a matrix that effectively inverts a non-square matrix. The pseudoinverse of a matrix J is also called the "Moore-Penrose inverse" of J. It is defined for all matrices J, even ones which are not square or not full row rank [SAMU 04, 8]. If we have a non-square matrix arising from an overconstrained or underconstrained system, we can try using the pseudoinverse method. For inverse kinematics problems, we set $\Delta\theta$ equal to

$$\Delta\theta = J^{*}\vec{e},$$

where the J^{*} is the pseudoinverse of J matrix. The pseudoinverse matrix J^{*} is defined by

$$J^{*} = J^{T}(JJ^{T})^{-1}.$$

The pseudoinverse method is widely discussed in the literature but it often performs poorly because of instability near singularities [BUSS 04, 9]. Singularity problems will lead to a situation in which end effectors move to impossible directions or oscillate. We can use the corresponding configuration to minimize the chances of bad effectors of singularities. The corresponding configuration can be setting the maximum rotation angle changed value (constant value). If $\Delta\theta$ exceed the maximum rotation angle, the $\Delta\theta$ will be scaled. However, the pseudoinverse method did not have good performances in articulated body animation even after configurations. After configuration, the pseudoinverse method might have the problem which is unable to effectively track targets. So, in the rest part of this chapter, we discuss other two more superior IK methods.

4.4.3 Damped Least Squares Method

The damped least squares (DLS) method avoids many of the pseudoinverse method's problems with singularities and can give a numerically stable method of selecting $\Delta\theta$ [BUSS 04, 9]. The damped least square solution uses

$$\Delta\theta = J^T(JJ^T + \lambda^2 I)^{-1}\vec{e},$$

to compute $\Delta\theta$, where $\lambda \in R$ is a non-zero damping constant. This damping constant must be chose very carefully and it depends on the configuration of the articulated body and target positions. The use of a suitable damping constant can make the Damped Least Squares method numerically stable.

4.4.4 Singular Value Decomposition

It is a useful fact that every $m \times n$ matrix has singular value decomposition. The singular value decomposition (SVD) provides solution for analyzing the pseudoinverse method, the DLS method and the SDLS method. The SDLS method is discussed in next section.

If a matrix J is a $m \times n$ matrix with $m>n$, then J can be written as so called singular value decomposition of the form

$$J = UDV^T.$$

In which U is a $m \times n$ matrix and V is a square $n \times n$ matrix, and both are orthogonal matrices; D is a $m \times n$ diagonal matrix. All diagonal elements of matrix D have non-zero values $d_{i,i} = \sigma_i$. If J is a complex matrix, then there always exists such decomposition with positive singular values .The singular value decomposition of J always exists, then it J can be written in the form below

$$J = \sum_{i=1}^{r} \sigma_i u_i v_i^T \qquad \text{[BUSS 04, 11],}$$

in which r is the largest rank value of matrix D with $\sigma_r \neq 0$.

The pseudoinverse of J is equal to

$$J^* = \sum_{i=1}^{r} \sigma_i^{-1} v_i u_i^T . \qquad \text{[BUSS 04, 12]}$$

The DLS methods can be wrote as below,

$$J^T(JJ^T + \lambda^2 I)^{-1} = \sum_{i=1}^{r} \frac{\sigma_i}{\sigma_i^2 + \lambda^2} v_i u_i^T . \qquad \text{[BUSS 04, 12]}$$

4.4.5 Selectively Damped Least Squares Method

Selected Damped Least Squares method (SDLS) is a refinement of Damped Least Square (DLS) method. SDLS method is based on the numeric filtering theory from Maciejewski and Klein [MACI 85,109]. The SDLS method selectively applies different filtering to all singular vectors. It is similar to choosing different damping value based on different targets' positions. In addition to DLS method, SDLS method considers the relative position of end effectors and target positions. So the SDLS method selects damping constants based on the configuration of articulated body and related position between end effectors and targets.

The steps to calculate the SDLS method is:

1. Compute Singular Value Decomposition from $J = UDV^T$

2. Calculate response vector $\Delta\theta$ that is the SDLS solution from $\Delta\theta = \sum_{i=1}^{r} \alpha_i \tau_i v_i$, for some scalars τ_i, and $\alpha_i = u_i^T \vec{e}$.

3. Calculate the value of $N_i = \sum_{j=1}^{k} \|u_{j,i}\|$

4. Calculate $M_{i,\ell} = \sigma^{-1} \sum_{j=1}^{n} |v_{j,i}| \rho_{\ell,j}$ with $\rho_{\ell,j} = \|\partial s_\ell / \partial \theta_j\|$. $M_{i,\ell}$ estimates the distances moved by the ℓth end effector caused by the individual changes in joint angles. [BUSSKIM 04,7] . And then calculate $M_i = \sum_\ell M_{i,\ell}$.

5. Set global constant $\gamma_{\max} = 4/\pi$ .This will be the maximum permissible change in any joint angle [BUSSKIM 04, 6]. And then set $\gamma_i = \min(1, N_i / M_i) \cdot \gamma_{\max}$.

6. Calculate $\varphi_i = ClampMaxAbs(\sigma_i^{-1} \alpha_i v_i, \gamma_i)$

7. Finally, clamp the value for $\Delta\theta$, $\Delta\theta = ClampMaxAbs(\sum_i \varphi_i, \gamma_{\max})$, and update joints angle.

The intuition behind the SDLS method is to consider each joint angle individually and decide how much it is trying to move the end efffector to the target position. [BUSSKIM 04, 6] The damping process of $\Delta\theta$ is implemented by restricting the maximum changes in joint angles. Compared with the DLS method, the SDLS avoids choosing damping constant. Compared with the pseudoinverse method, SDLS reduced oscillate when near singularity or singularity since the SDLS method treats singular vectors separately.

For the pseudoinverse, DLS and SDLS methods, we must first loop through the joints, compute and store the *J*, use SVD to get solutions for pseudoinverse, DLS and SDLS, and then compute the change in joints, and finally apply the change to all joints. Comparing these methods above with the Jacobian transpose method, the Jacobian transpose method is more efficient in memory access and caching, as well as using fewer computations than pseudoinverse, DLS and SDLS methods. However, the Jacobian transpose method still has worse performance than the DLS and the SDLS method. In Chapter 5, I discuss the experimental results of these four methods introduced in this chapter.

Chapter 5

Implementation of Inverse Kinematics Algorithms

5.1 Modules

The practical system described in this thesis implements inverse kinematics algorithms for an articulated human body. The implementation is organized into three modules: the Control Module, the Calculation Module, and the Animation Module. Unlike other inverse kinematics animation systems, the Data Module does not get input animation data from key-framed or Motion capture data but, instead, all the animation data in this system are calculated by inverse kinematics algorithms in the Calculation Module in real-time and then mapped onto the articulated human body by the Animation Module. The next three sections describe these modules in detail.

5.1.1 Control Module

The Control Module is the top-level component of the system; its purpose is to generate a set of values for the parameterized motion. The Control Module encapsulates the information required to produce a motion, and passes this information on to the Calculation Module and the Animation Module.

Since the implementation uses inverse kinematics algorithms to simulate human animation, the main control parameters of motions are targets' positions of the articulated body. There are four targets for the body: they are close to the left hand, right hand, left foot, and right foot. The state of the Control Module can be dynamically changed in response to the system's control sources, which are the four targets. Based on the changing positions of targets, the Calculation Module uses an inverse kinematics algorithm to calculate the $\Delta\theta$ value for each joint in the body.

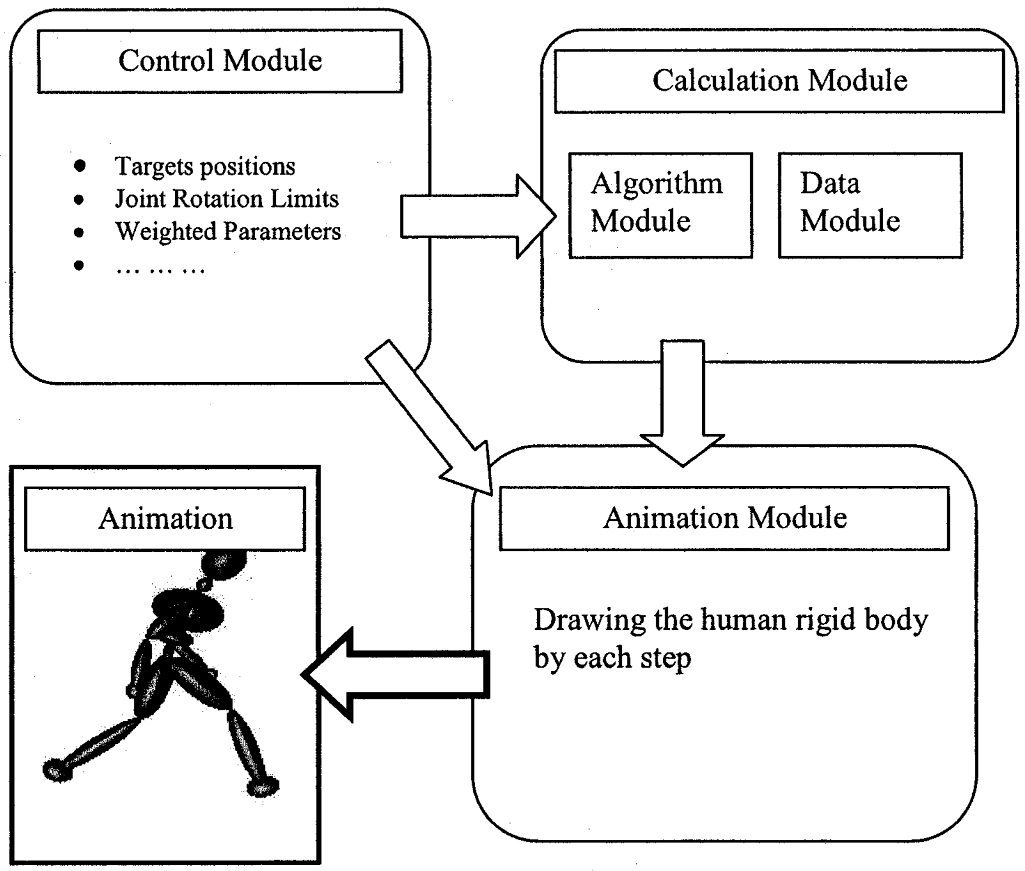


Figure 5. 1 Control structure of the inverse kinematics system

There are several other control parameters in the system. A user can choose any of the four inverse kinematics algorithms to calculate the $\Delta\theta$ values for joints. Also, the user can choose “walking”, “running”, “injured walking”, and “injured running” to control the type of action for the articulated body animation.

In addition, the Control Module also provides information to control the calculation and output data. This information includes:

- Joint angle rotation limitations

The system set a fixed value range for each joint to prevent unnatural rotation of the human body. This value is provided as a joint configuration at the beginning. Based on the algorithm calculation, if the $\Delta\theta$ value is not in the range of the joint rotation limitation, the control system provides a solution to adjust the rotation which is within the joint angle rotation limit range.

- Weighted parameters

In order to imitate wounded walking, weighted parameters are set to allow some joints to rotate more easily than others. For example, the θ_i is the rotation angle for left knee. Before updating the value for the rigid body, we multiply or add the weighted parameter value w and θ_i, so that $\theta_{i-new} = w\theta_{i-old}$ or $\theta_{i-new} = \theta_{i-old} + w$ and we then map the new θ_i value to the human body. As a result, the human body performs an injured walking or running motion.

- Speed parameters

The Control Module provides some additional controls, such as "Step Frequency", "Slowdown Factor" and "Speed" to control the speed of walking and running.

- Step parameters

The Control Module also provides control of the leg step gait. There are two vectors, one is the front leg's maximum position, another one is the back leg's maximum position. These two vectors can be changed in order to change the gait of each step.

- Hand parameters

Similar to step parameters, the Control Module also provides hand swing parameters to control the positions of hands. There are two vectors were set to be maximum swing position of front hand and back hand when the articulated body was performing walking or running.

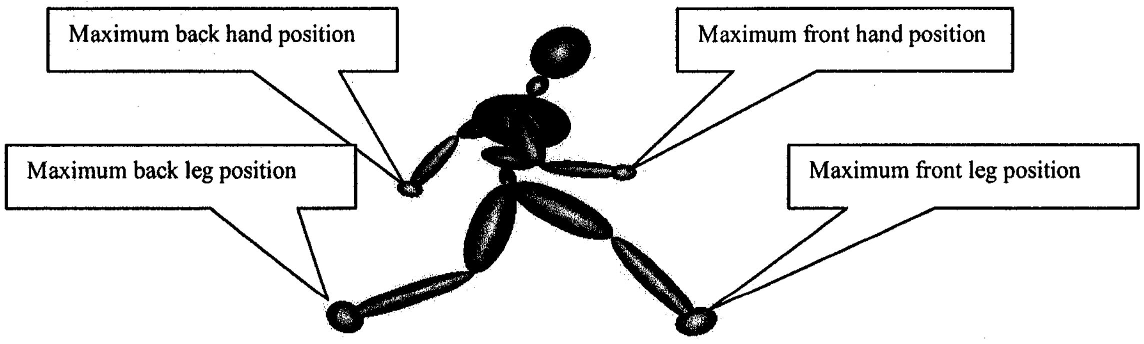

Maximum back hand position
Maximum front hand position
Maximum back leg position
Maximum front leg position

Figure 5. 2 Step parameters and hand parameters

5.1.2 Calculation Module

The Calculation Module consists of the Data Module and Algorithm Module. Based on the control parameters transferred from the Control Module, the Algorithm Module sets the selected inverse kinematics algorithm as the current algorithm. Based on the current algorithm, the Algorithm Module creates a corresponding Jacobian matrix J and other factors. At each step, the target positions are updated and the current algorithm calculates the corresponding rotation angle, changing the value $\Delta\theta$ for each joint. For example, for the transpose Jacobian method $\Delta\theta = \alpha J^T \vec{e}$, the Algorithm Module first calculates the scalar α , then transposes the J to J^T, and finally calculates the $\Delta\theta$ value.

The Data Module in this implementation provides data to perform actions for the Animation Module. The data is in the form of a degree-of-freedom (DOF) value, specifically, the rotation-angle $\Delta\theta$ for each joint in the articulated body. The $\Delta\theta$ values for each joint are stored in a multi-dimension vector $[\Delta\theta_{1,}\Delta\theta_{2,}......,\Delta\theta_n]^T$, where n is the number of joints in the rigid body. Then the Data Module updates the vector $[\Delta\theta_{1,}\Delta\theta_{2,}......,\Delta\theta_n]^T$ for each joint in the tree structure, obtaining the global position and axis of each node. The information for each node is then transferred to the Animation Module.

5.1.3 Animation Module

The Animation Module in this completes the human animation simulation. It gets combined information from Control Module and Calculation Module, and maps the animation data onto the rigid human body.

The Animation Module provides visualization for the joints in the articulated body. In each step, after the Calculation Module has calculated the vector $[\Delta\theta_{1,}\Delta\theta_{2,}......,\Delta\theta_n]^T$ based on the current target positions transferred from the Control Module, the Animation Module draws each node on the predefined joint's geometrical shape (obtained from the Control Module). In each step, the Animation System rotates and draws joints in different positions. As the body is drawn in different positions at each step of the computation, the user perceives the illusion of natural human motion.

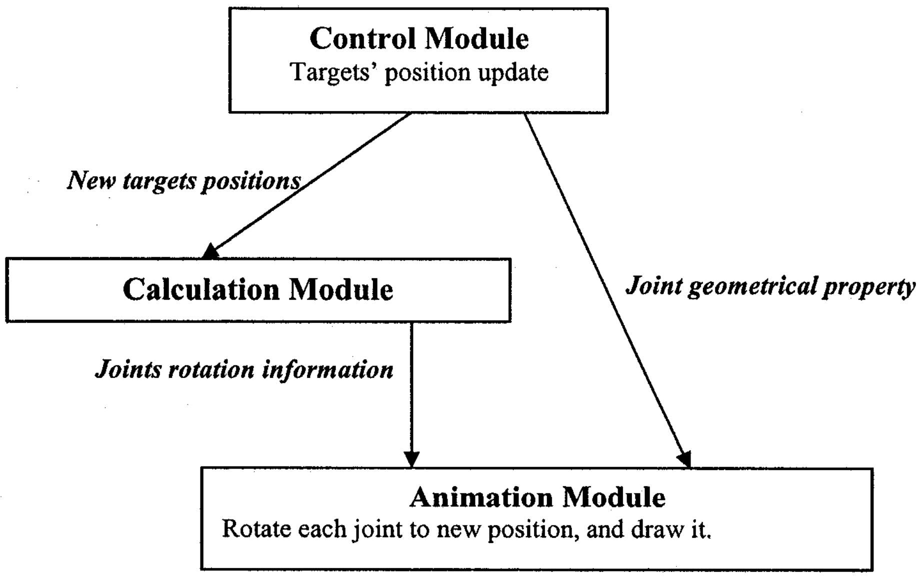

Control Module
Targets' position update
New targets positions
Calculation Module
Joint geometrical property
Joints rotation information
Animation Module
Rotate each joint to new position, and draw it.

Figure 5. 3 An example of data transfer between modules

5.2 Animation Practicing

The animation component of this system includes four action simulations: "Walking", "Running", "Wounded Walking", and "Wounded Running". These actions are simulated on an articulated model of a human body with 20 joints and 4 end effectors. The target positions are moved sinusoidally in *X, Y*, and *Z* directions, each component with its own period.

5.2.1 Human Walking or Running Cycle

Simulation of realistic human walking and running is a one of the more difficult research problems in multibody systems and robotics due to its complexity and high dimensionality. There are many solution are based on periodic walking or running cycle and moving on a flat surface based on a constant speed. In a general approach, the human walking step is composed of two different phases. The first phase is the swing phase or single support phase when one foot is on the ground while the other swings. [HART, 99] The second phase is called the double support phase as both feet are on the ground while the body is moving forward [HART, 99]. In this thesis, I assume a human walking cycle that can be simulated by a left leg swing phase, a double support phase, and a right leg swing phase.

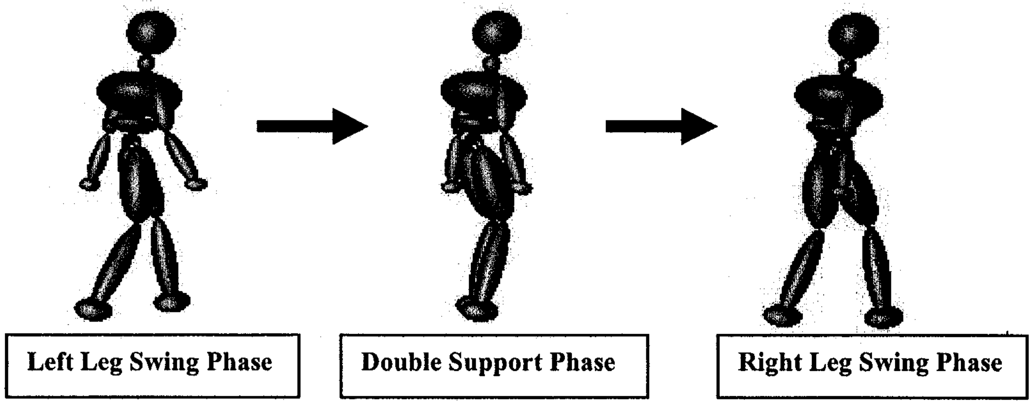


Figure 5. 4 One human walking cycle

A simple human running cycle is a cyclic behavior in which legs swing forwards and backwards. A running cycle consists of two swing phases; a left leg swing phase, and a right leg swing phase. A running cycle doesn't have the double support phase since both feet are never on the ground at the same time. However, in this thesis, I still assume that there is an intermediate phase, which is similar to double support phase in the walking cycle but both legs are off the ground.

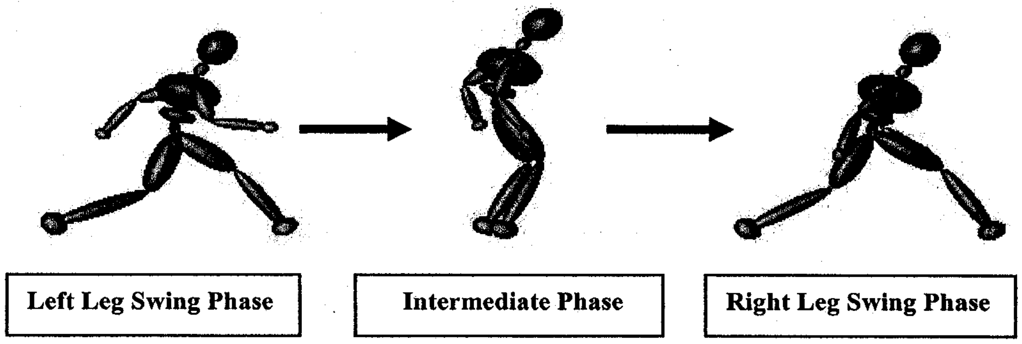


Figure 5. 5 One human running cycle

In this thesis, the walking and running motions of the human body are based on sinusoidal curves. These curves, and the way in which they are applied to animation, is the topic of the next section.

5.2.2 Sinusoidal Curve

The sinusoidal curve provides a good approximation to the motion of the joints of the leg during walking or running. In this thesis, I use sinusoidal curves to control the movement of target positions in order to control whole walking and running cycles.

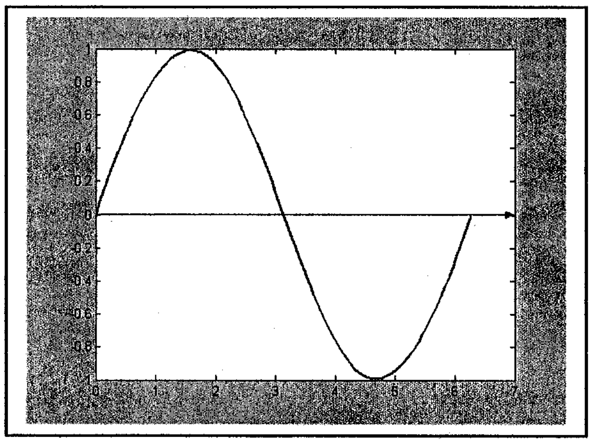


Figure 5. 6 An example of sinusoidal curve

- Walking cycle control source code:

*target[0].Set(0.35,back_y_leg,back_z_leg*sin(walk_speed*T));*

*target[1].Set(-0.35,front_y_leg,front_z_leg*sin(walk_speed*T));*

*target[2].Set(-0.8,back_y_hand,-0.3*sin(walk_speed*T));*

*target[3].Set(0.8,front_y_hand,0.3*sin(walk_speed*T));*

- Running cycle control source code:

```
target[0].Set(0.35,back_y_leg,back_z_leg*sin(run_speed*T));
target[1].Set(-0.35,front_y_leg,front_z_leg*sin(run_speed*T));
target[2].Set(-0.8,0,-1.0*sin(run_speed*T));
target[3].Set(0.8,0,1.0*sin(run_speed*T));
```

The source code above indicates the usage of sine function in order to control the *z* axis value of each target's position. The statement *"target [num].set(x, y, z)" sets a* target position. "*walk_speed*" and "*run_speed*" are two variables set by the user before starting the walking and running simulation. In addition, the coefficients, "± 0.35", "± 0.8", *"front_y_leg", " back_z_leg", "back_z_leg", "front_z_leg", "back_y_hand"* and *"front_y_hand"* are constants. We chose corresponding coefficients for the *"target [num].set(x, y, z)"* function based on walking and running step distance in the coordinator.

The variable *T* is incremented each time the target positions change. The sine function changes its value as *T* increases. As a result, the targets move through a sinusoidal curve in 3D space. In addition, this thesis assume that the right leg swing phase and the left leg swing phase in walking/running cycle take the same time period. Based on this feature of walking and running cycle, we can use a sine function to control the target positions' movements.

5.2.3 Wounded Walking and Running

The Control Module uses some weighted parameters to simulated walking and running for a wounded body with rigid limbs. In the Calculation Module, before mapping rotation angles to joints in the rigid body, the system multiplies or adds weighted parameters to specific joint rotation angle values. In figure 5.3 the specified wounded effect is applied to the left hip, upper leg and right upper arm. In figure 5.4 the specified effect is applied to the left upper leg and right upper arm.

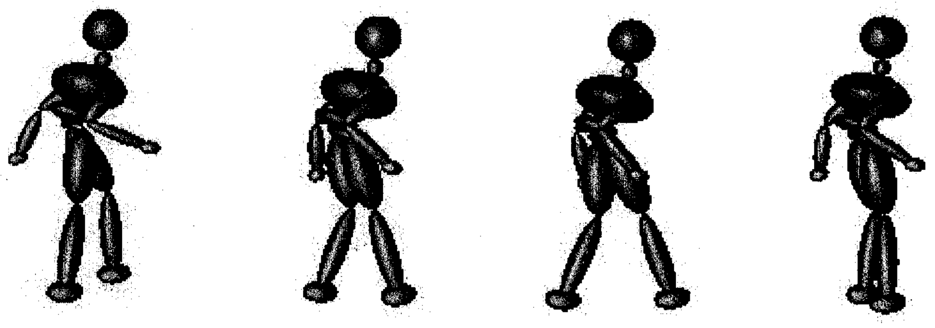


Figure 5. 7 Wounded walking

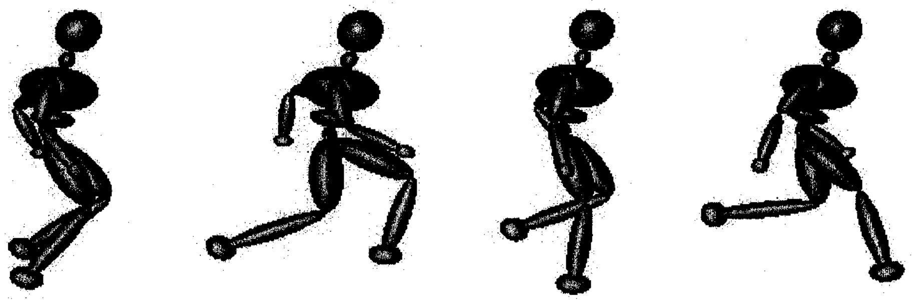


Figure 5. 8 Wounded running

5.3 Comparison of Inverse Kinematics Algorithms in Walking and Running Animation

The previous chapter introduced four inverse kinematics algorithms: the Jacobian Transpose Method, the Pseudoinverse Method, the Damped Least Squares Method (DLS), and the Selectively Damped Least Squares Method (SDLS).

To compare these inverse kinematics algorithms, I implemented each of them on a human rigid human body in order to perform walking and running animation. The target position were moved by varying each *x, y, z* component sinusoidally, each component with its own period. In these experiments, the target position increments were small enough to provide a smooth visual result. The end effectors track the movement of target positions, and the system used one of the algorithms to calculate the corresponding rotation angles for each joint in the rigid body. Two leg target positions in walking cycle are reachable, while others are not reachable.

Based on the distance between targets and corresponding end effectors, there are two different types of target positions. One is reachable target positions; another is unreachable target positions. I set d as the distance between one target and the corresponding end effector. During the animation simulation, if $d \leq 0$, the target position is reachable. If $d > 0$, the target position is unreachable. For example, in Figure 5.10, two targets which are related to end effectors in legs are always reachable during the walking simulation. And other two targets related to hands are always unreachable during the walking simulation.

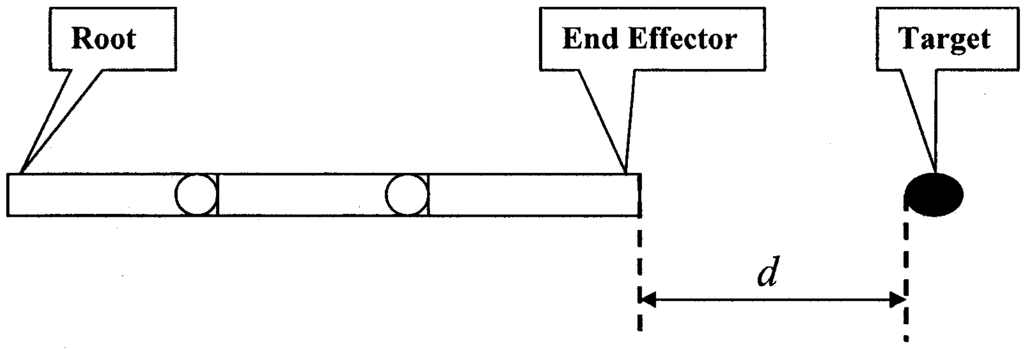


Figure 5. 9 Distance (*d*) between end effector and target

- Jacobian Transpose Method

This method is fast and easy to implement. However, it performed poorly in walking and running animation, even when the target position increments are very small. The human articulated body looks jerky and oscillates in walking and running animation applications. Usually, this method is suggested only for single target position tracking.

- Pseudoinverse Method

This method performs a little bit better than Jacobian transpose method in giving a smooth visual result. However, its performance was still poor in the walking and running cycles.

Comparing this application between walking and running cycles, the walking cycle had better performance. The reason has to do with setting the target position to be reachable or unreachable. In the running cycle, four targets position were set to be unreachable, while in walking cycle, two legs target positions were set to be reachable. One of the features of this method is that it can work properly only when the target positions are reachable. As a result, the walking animation application of this method is better than running animation application. However, the worst feature of the pseudoinverse method was that the unreachable target positions caused the rigid body to oscillate.

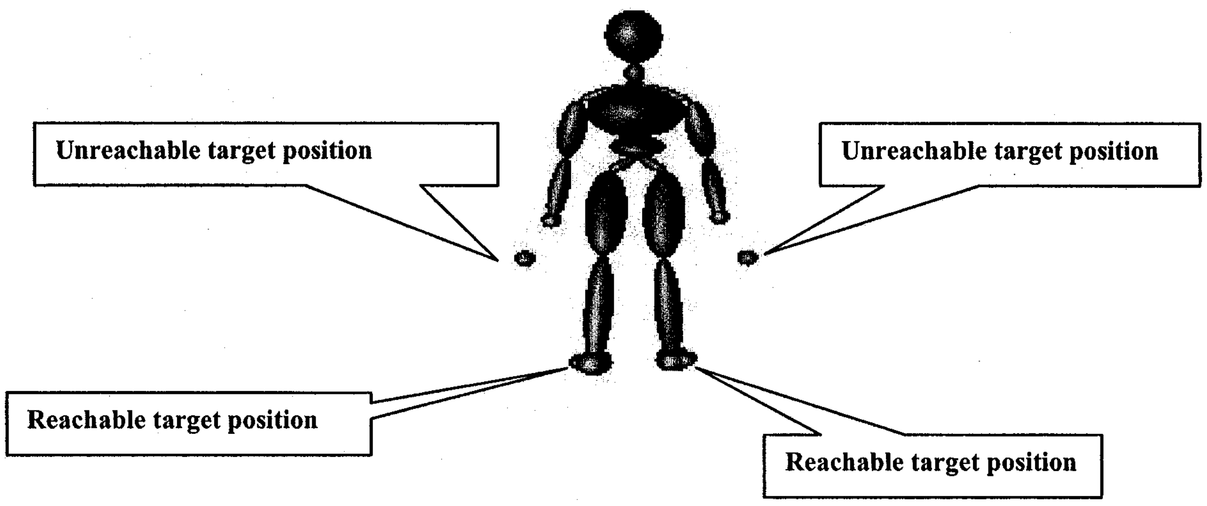

Unreachable target position
Unreachable target position
Reachable target position
Reachable target position

Figure 5. 10 Reachable target positions and unreachable target positions in walking cycle

- Damped Least Squares Method

The application of this method worked substantially better than either the Jacobian Transpose Method or the Pseudoinverse Method. Walking and running animation is natural and smooth. Compared with SDLS method, the DLS method did not have good performance when the target position was reachable.

- Selectively Damped Least Squares Method

This method has the best performance of all four algorithms. But it is slower than the other algorithms since it requires the most calculation. It also has the best performance for both reachable target positions and unreachable target positions. For the rigid human body with multi-end effectors that I have used in this thesis, Selectively Damped Least Squares Method is a good choice.

Chapter 6

Implementation

6.1 Coding

The objective of this thesis is the simulation of walking and running animation of an articulated human body. In order to implement the algorithms, there are two important pieces of code. The first part is the coding of the articulated structure of the body. The second part is the Jacobian Matrix based inverse kinematics algorithm. In this section, we show some abstract coding samples. I used Buss's inverse kinematics algorithms coding samples code as start point.

6.1.1 Coding for Rigid Body

The articulated human body consists of joints and limbs. The joints are arranged in a tree structure, as described in Chapter 3. The joint is the basic data type of this articulated human body. The tree structure is a container used to store joints as nodes in an ordinary structure.

- Joint (Node) class: normal joints and end effectors
- Tree class: A binary tree structure consists of joints

6.1.1.1 Joint Class

Joints have various properties from the point of view of the simulation. The joint (node) class defined in this thesis represents the joints in articulated human body. Some properties are fixed and some properties change with time. I separate the properties of a joint into two categories: immutable properties, and mutable properties.

- Immutable properties:

These properties were provided during initialization of the joints and the tree structure, and cannot be changed. This ensures that the body maintains its essential, invariant features during the animation.

```
double size;            // geometry size of this joint
```

The *double size* is the geometrical size of each joint. In this thesis, we assume that basic body characteristics do not change during the motion, so that the size of a joint does not change during the simulation.

```
VectorR3 attach;        // the attached point of this joint with its neighbors
VectorR3 r;             // the related position difference with its neighbors
```

During the simulation, joints are moving and rotating. These vectors are the key to ensuring that the joints remain connected after being translated and rotated. If they were allowed to change during the simulation, the body would come apart after even a few steps.

```
double minTheta;          // lower limit of joint angle
double maxTheta;          // upper limit of joint angle
```

These joint limits, *minTheta* and *maxTheta*, are predefined during initialization of the joint. When the system performs simulations, these properties are used to prevent unnatural rotation of the joints. These two limits are set as the upper limit and lower limit of joint rotation angles, respectively. If the algorithm calculates a rotation angle larger than *maxTheta* or less than the *minTheta*, the system will use the limit value (*maxTheta* or *minTheta*) instead of the calculated result.

```
Node* left;               // left child node
Node* right;              // right child node
Node* parent;             // pointer to parent node
```

These three pointers to *Node* determine the position of the joint in the tree structure. It will not be changed during the simulation in order to keep the tree structure consistent.

```
enum Purpose {JOINT, EFFECTOR};
// indicate the purpose of this joint in the tree structure, joint or end effector
Purpose purpose;
```

This enumeration distinguishes normal joints and end effectors in the body. It is very important to treat them separately when the system is using inverse kinematics algorithms to calculate the transformation of joints.

- Mutable properties:

Joints have other properties that change during execution of the simulation. These variable properties include:

```
VectorR3 s;                    // Position vector of this joint
VectorR3 w;                    // Global rotation axis
double theta;                  // rotation angle of this joint
```

These properties represent the position, orientation, and rotation of the joint. They change at each step of the simulation. The changes are determined by mutator functions:

```
void SetTheta(double newTheta)   // Change rotation angle  theta value
void ComputeS(void);             // Change position vector
void ComputeW(void);             // Change rotation axis vector
```

6.1.1.2 Tree Class

The tree structure is a container for joints (nodes). In this simulation, joints are stored in a binary tree structure. Each node (joint) of this tree can have one or two child nodes, and has at most one parent node. A tree has one root node which is the only node with no parent. There are some functions defined for this tree structure; these are described next.

```
void InsertRoot(Node*);   /Iinsert root node to the tree
// Insert left/right child to a node in the tree
void InsertLeftChild(Node* parent, Node* child);//
void InsertRightChild(Node* parent, Node* child);
```

The tree class also provides functions to traverse the tree. For example:

```
Node* GetRoot() const { return root; } //return root for the tree
Node* GetSuccessor ( const Node* ) const; // return child nodes
Node* GetParent( const Node* node ) const ;//return parent node
```

In addition, in both the tree class and the node (joint) class, there are functions for drawing the current tree and node.

```
void DrawTree(Node*);       //In tree class, draw the tree
void DrawNode(bool);        //In node class, draw the node
```

6.1.2 Coding for Inverse Kinematics Algorithms

All articulated human body animation simulations in this thesis are based on Jacobian series inverse kinematics algorithms, as explained in Chapter 4. Since I used a tree structure to present the articulated human body, all calculations performed by the algorithms are based on the tree structure. On the other hand, the Jacobian series inverse kinematics algorithms are all based on the Jacobian Matrix. Consequently, I had to create the Jacobian Matrix based on the tree structure, and then implement all algorithms based on the Jacobian Matrix.

6.1.2.1 Jacobian (Matrix) Class

The detailed description of Jacobian Matrix is given in Chapter 4. The constructor of Jacobian class gets information from the tree structure and creates the corresponding Jacobian matrix.

MatrixRmn J; *// Jacobian matrix ($m \times n$)*

Jacobian (Tree);* *// Constructor of Jacobian matrix*

Based the joints number n and end effectors number m in the tree, the Jacobian matrix is $3m \times n$ matrix. Initially, all elements in the Jacobian Matrix are 0.

VectorRn dTheta; *// n dimention vector*

In the Jacobian class, there is an n-dimensional vector *dTheta*, which stores all joint rotation angles $\Delta\theta$. The dimension of this vector is the number of joints in the articulated human body. At each simulation step, the inverse kinematics function calculates the value of $\Delta\theta$ for each joint. As below, there are four inverse kinematics algorithm calculation functions. In addition, there is another function defined to update new $\Delta\theta$ value to vector *dTheta.*

void CalcDeltaThetasTranspose(); *// Transpose algorithm*

void CalcDeltaThetasPseudoinverse(); //Pseudoinverse algorithm

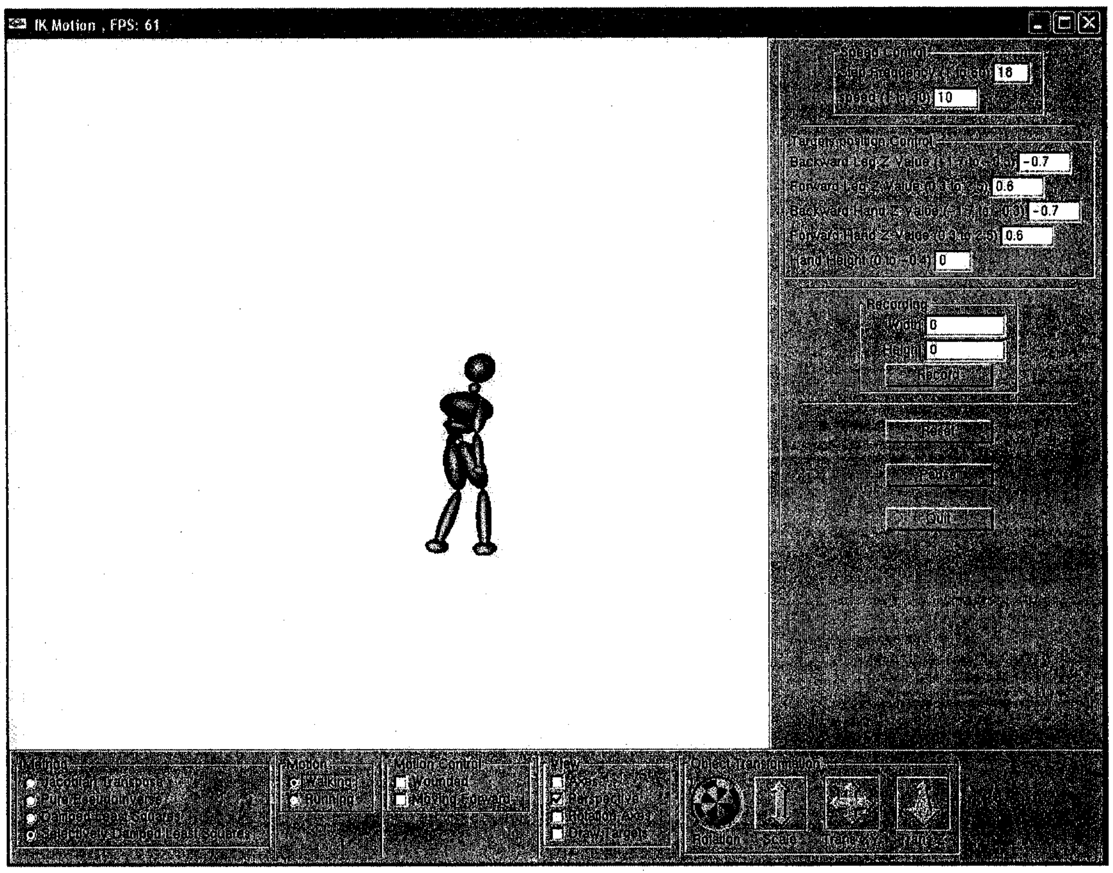
void CalcDeltaThetasDLS(); *//DLS algorithm*

void CalcDeltaThetasSDLS(); *//SDLS algorithm*

void UpdateThetas(bool Wounded,int motion); //Update $\Delta\theta$ *to joints*

6.2 User Interface

Figure 6.1 shows an example of the user interface for the inverse kinematics animation simulation system. The interface is an animator's workbench. The inset-windows on the bottom and right side are control panels.



IK Motion , FPS: 61

Speed Control
Step Frequency (1 to 30) 18
Speed (1 to 30) 10

Targets position Control
Backward Leg Z Value (-1.7 to -0.3) -0.7
Forward Leg Z Value (0.3 to 2.5) 0.6
Backward Hand Z Value (-1.7 to -0.3) -0.7
Forward Hand Z Value (0.3 to 2.5) 0.6
Hand Height (0 to -0.4) 0

Recording
Width 0
Height 0
Record

Reset
Pause
Quit

Method
Jacobian Transpose
Pure Pseudoinverse
Damped Least Squares
Selectively Damped Least Squares

Motion
Walking
Running

Motion Control
Wounded
Moving Forward

View
Axes
[✓ Perspective]
Rotation Axes
Draw Targets

Object Transformation
Rotation Scale Trans XY Trans Z

Figure 6. 1 An example interface for this system

The interface includes nine sub-panels. The panels on the bottom are "Method Panel", "Motion Panel", "Motion Control Panel", "View Panel", and "Object Transformation Panel". Panels on the right side are "Speed Control Panel", "Target Position Control Panel", "Recording Panel" and a panel with three buttons.

- Method Panel

The user can choose one of these four inverse kinematics methods to perform simulation.

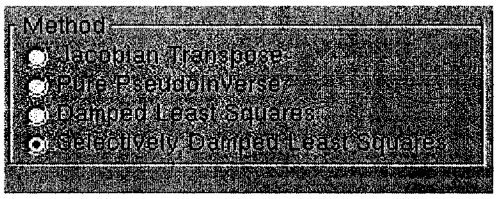


Figure 6. 2 Method panel

- Motion Panel

The user can choose "Walking" simulation or "Running" simulation.

Figure 6. 3 Motion panel

- Motion Control Panel

The user can choose "Wounded" effect or "Moving Forward" forward effect on the rigid human body.

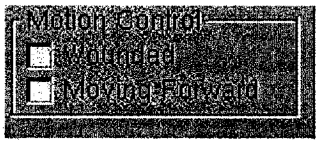


Figure 6. 4 Motion control panel

- View Panel

This panel includes additional viewing options. The "Axes" option draws *X, Y, Z* Axes on the view window. The "Perspective" option provides perspective viewing result. The "rotation Axes" option draws rotation axes for each joint in the human rigid body. The "Draw Targets" option draws target position for the rigid human body.

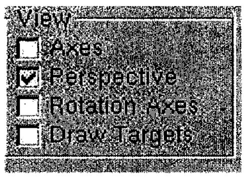


Figure 6. 5 View panel

- Object Transformation Panel

This panel provides transformation options for the rigid human body. For example, when the user touches the "Rotation" ball, the rigid human body will be rotated at the same time as it performs the animation simulation. This panel provides different observational point for users.

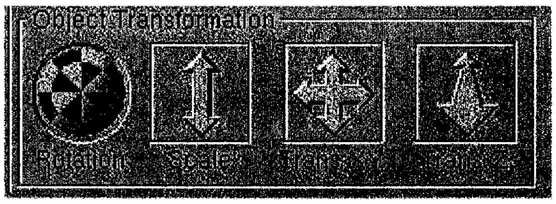
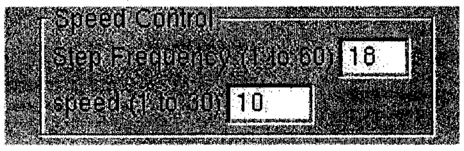


Figure 6. 6 Object transformation panel

- Speed Control Panel

This panel provides speed setting and step frequency setting for the walking and running simulation.



Speed Control

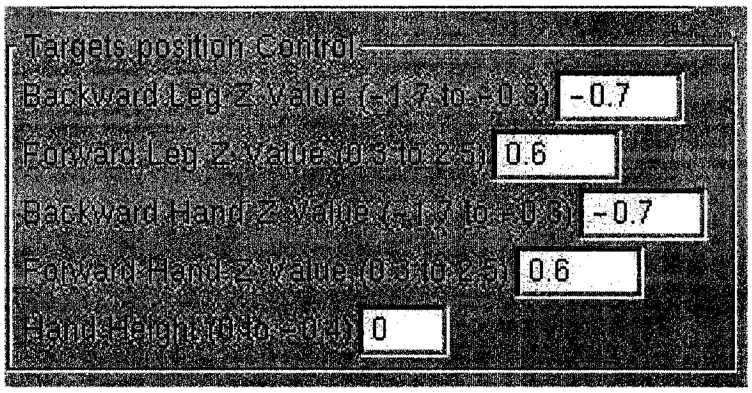
Step Frequency (1 to 60) 18

speed (1 to 30) 10

Figure 6. 7 Speed control panel

- Target Position Control Panel

There are four targets for this rigid human body, two for hands and two for legs. This panel provides target position setting for these four targets.



Targets position Control

Backward Leg Z Value (-1.7 to -0.3) -0.7

Forward Leg Z Value (0.3 to 2.5) 0.6

Backward Hand Z Value (-1.7 to -0.3) -0.7

Forward Hand Z Value (0.3 to 2.5) 0.6

Hand Height (0 to -0.4) 0

Figure 6. 8 Target position control panel

- Recording Panel

In this system, the user can record each frame of animation and create a movie (".avi") file. Users can set the frame size they want for the recording.

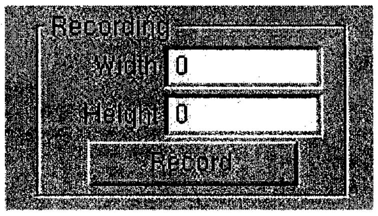


Figure 6. 9 Recording panel

- 3 Button Panel

The "Reset" button resets the animation to initial status. The "Pause" button pauses the animation. And the "Quit" button is the button to exit the system.

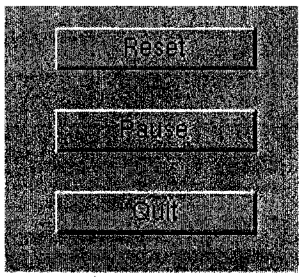


Figure 6. 10 Buttons

Chapter 7

Conclusion and Future Work

7.1 Summary

We have examined the simulation of human walking and running by using Inverse Kinematics algorithms which are based on the Jacobian Matrix. Human walking and running motions are simulated by end effectors in an articulated human body tracking target positions. We used sinusoidal curve as the moving tracks of targets. And, in order to simulate injured walking and running motion, we added weighted parameters in corresponding joints. To complete the animation, we used IK algorithms to calculate joint rotation angles corresponding to the movement of targets.

The first step in solving the inverse kinematics problem is to differentiate a set of non-linear equations to obtain linear equations by constructing the Jacobian Matrix. The second step of the approach is to use the Jacobian Transpose method, the Pseudoinverse method, the DLS method, or the SDLS method to solve IK problems. The Jacobian Transpose method uses the transpose of Jacobian Matrix. It is the simplest of the four methods and requires the least iteration. However, it shows the worst performance in articulated human body animation. This method has good performance with only one target tracking. The Pseudoinverse method is so-called because it uses the pseudoinverse of the Jacobian Matrix rather than the true inverse. In practice, this method applied to walking and running gives slightly better results than the Jacobian Transpose method. However, both methods have problems when the Jacobian Matrix is singular or close to singular. The DLS method and SDLS method have greater complexity than the other two methods. However, according to the research described in this thesis, they are not slower than another two methods in practice. Furthermore, they provide much better animation in human walking and running simulation. After experimenting with several methods in the course of this work, we concluded that the SDLS method has the best performance in human walking and running simulation of these four methods.

7.2 Conclusion

Most cited approaches that use IK use it as part of a system for motion capture data editing and retargeting. The advantage of motion capture technique is that it directly gives exact end effectors' position. The IK technique should estimate targets' position in order to get end effectors' position. However, one of the main issues in using motion capture data to animate human figures is the problem of retargeting [MERE 05, 1]. This is the issue of applying motion data captured from one person to a virtual person of a different size [MERE 05, 1]. In order to solve motion capture data retargeting problems, researchers should spend a lot of time in doing motion capture data editing and synthesis. From the discussion above, we can conclude that it is possible to simulate human motion without any motion capture data; and only rely on kinematics algorithms such as inverse kinematics algorithms.

In addition, the thesis shows that it is also possible to simulate human animation without physical considerations (dynamics). We can use purely mathematical techniques to simulate it. For example, a real walking cycle simulation must take into account gravity, the weight of human body, and the friction of the floor, etc. With different physical factors, the motion of the body will have different resulting motions. However, incorporating these factors into the simulation will consume valuable system resources. Instead, in our system, without physical consideration, we can adjust the target positions in order to have different motion result. And different rotation results lead to different walking or running styles. Consequently, we can reduce system cost in real time motion simulations.

This inverse kinematics animation technique demonstrates the feasibility of designing computer games based upon fully kinematics character simulations. It allows animators to design varieties of motion. And also this technique will play an important role in real time animation simulation systems.

7.3 Future Work

There are several directions which require further research to make the motion more realistic and to simulate more human motions.

- More Motion Simulation

This thesis uses sinusoidal curves to control target positions, and then to control the whole articulated body's motion. For other motions of human body, we can first analyze the path of target movement, using the path as input for this simulation system, and obtain the result transformation of each joint in the body.

- More Complex Motion Simulation

This thesis only considers each joint's animation with 1-DOF. We seek to simulate different motions. So we should consider more DOFs in the calculations for each joint. With some modified of Jacobian Matrix in algorithms, the system can calculate joints with more than one DOF. It will then be suitable for a real human body motion animation simulation, since different joints in the body have different DOF numbers.

- More Constraints and More Effectors

It should also be mentioned that it is necessary to introduce even more constraints in joints in order to make the animation realistic. And this is a requirement for real time motion simulation, since sometimes we cannot predict what kind of pose the articulated human body can perform during real-time interaction.

Another possibility for future work is to provide more effectors on the articulated human body. For motions other than walking and running, we might specify more effectors in the articulated body. The effector locations are not restricted to the end of legs and arms. They can be any joints in the articulated body.

- Combination of Algorithms

Based on different features of these IK algorithms, we can try to give a combination of methods for solving the IK problem. In fact, we can select the most suitable method according to the given conditions. Even in one articulated human body, we can try to apply different IK algorithm in different end effectors.

- Improved User Interface

One of the potential applications of the system is game developers. I plan to investigate how professional animators specified the animation factors to be controlled. I plan to modify the user interface based on the requirement of professional animators. I will add more viewing windows and control panel in order to fulfill their requirement.

References

[BUSS 04] S.R. Buss, "Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares Methods", 17 April 2004, <http://math.ucsd.edu/~sbuss/ResearchWeb >, (22 March 2006).

[BUSSKIM 04] S.R.Buss and J.S. Kim, "Selectively Damped Least Squares for Inverse Kinematics", Journal of Graphics Tools, vol. 10, no. 3, pp.37-49, 2005.

[BUSS 03] S.R. Buss, *3-D Computer Graphics a Mathematical Introduction with OpenGL*, 1st ed., Cambridge, The Press Syndicate of Cambridge University, 2003, pp.289-307.

[GE 00] K.T. Ge, "Solving Inverse Kinematics Constraint Problem for Highly Articulated Models", Master's thesis, Waterloo University, 2000.

[JOCH 00] A. Jochheim, M. Gerke and A. Bischoff, "Modeling and Simulation of Robotic Systems", <http://virtual.cvut.cz/odl/partners/fuh/course_main>, 17 August 2000, (22 March 2006).

[WELM 93] C. Welman, "Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation", Master's thesis, Simon Fraser University, 1993

[MERE05] M. Meredith and S. Maddock, "Adapting Motion Capture Data Using Weighted Real-time Inverse Kinematics", ACM Computers in Entertainment, vol.3, no.1, 2005.

[GROC 04] K. Grochow, S.L. Martin, A. Hertzmann, and Z. Popovic, "Style-Based Inverse Kinematics", ACM Trans. on Graphics, vol. 23, no.3, pp. 522 – 531, August 2004.

[TOLA 00] D. Tolani, A. Goswami and N.I. Badler, "Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs", Graphical Models and Image Processing, vol. 62, no. 5, pp. 353-388, September 2000.

[WATT 03] A. Watt and F. Policarpo, *3D Games Animation and Advanced Real-Time Rendering*, 1st ed., Harlow, Addison Wesley Professional, 2003, pp. 460-523.

[MACI 85] A. Maciejewski and C.A. Klein, "Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments", International Journal of Robotic Research, vol. 4, pp. 109-117, 1985

[MERE 02] M. Meredith and S. Maddock, "Motion Capture File Format Explained", < http://www.dcs.shef.ac.uk/~mikem/>, Department of Computer Science Technical Report CS-01-11, University of Sheffield, 2001, (24 January 2006).

[HART 99] M. W. Hardt, "Human Walking", <http://www.sim.informatik.tu-darmstadt.de/~hardt/papers/heidelberg/node2.html>, October 1999, (22 March 2006).

[HO 05] E.S.L. Ho, T. Komura and R.W.H. Lau, "Computing Inverse Kinematics with Linear Programming", ACM Symposium on Virtual Reality Software and Technology, pp.163-166, 2005.

[ZHAO 94] J. Zhao and N.I. Badler, "Inverse Kinematics Positioning using Nonlinear Programming for Highly Articulated Figures", ACM Transactions on Graphics (TOG), vol.13, no.4, pp. 313-336, October 1994.

[LEE 99] J. Lee, S.Y. Shin, "A Hierarchical Approach to Interactive Motion Editing for Human-like Figures", SIGGRAPH 99, pp. 39-48, July 1999.

[ROSE 96] C. Rose , B. Guenter, B. Bodenheimer and M.F. Cohen, "Efficient Generation of Motion Transitions Using Spacetime Constraints", Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, pp.147-154, 1996.

[FEDOR 03] M. Fedor, "Application of Inverse Kinematics for Skeleton Manipulation in Real-time", Spring Conference on Computer Graphics, Proceedings of the 19th spring conference on Computer graphics, pp. 203-212, 2003.