ANIMATION OF HUMAN MOTION WITH INVERSE KINEMATICS USING NONLINEAR PROGRAMMING

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER

ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

by

A. Sezgin Abalı

September, 2001

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Uğur Güdükbay (Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Bülent Özgüç

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Özgür Ulusoy

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Science

ABSTRACT

ANIMATION OF HUMAN MOTION WITH INVERSE KINEMATICS USING NONLINEAR PROGRAMMING

A. Sezgin Abalı

M.S. in Computer Engineering

Supervisor: Assist. Prof. Dr. Uğur Güdükbay

September, 2001

Animation of articulated figures has always been an interesting subject of computer graphics due to a wide range of applications, like military, ergonomic design etc. An articulated figure is usually modelled as a set of segments linked with joints. Changing the joint angles brings the articulated figure to a new posture. An animator can define the joint angles for a new posture (forward kinematics). However, it is difficult to estimate the exact joint angles needed to place the articulated figure to a predefined position. Instead of this, an animator can specify the desired position for an end-effector, and then an algorithm computes the joint angles needed (inverse kinematics). In this thesis, we present the implementation of an inverse kinematics algorithm using nonlinear optimization methods. This algorithm computes a potential function value between the end-effector and the desired posture of the end-effector called goal. Then, it tries to minimize the value of the function. If the goal cannot be reached due to constraints then an optimum solution is found and applied by the algorithm. The user may assign priority to the joint angles by scaling initial values estimated by the algorithm. In this way, the joint angles change according to the animator's priority.

Keywords: animation, human motion, inverse kinematics, nonlinear programming, optimization.

ÖZET

DOĞRUSAL OLMAYAN PROGRAMLAMA KULLANAN TERS KİNEMATİK YÖNTEMİYLE İNSAN MODELLERİNİN ANİMASYONU

A. Sezgin Abalı
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yrd. Doç. Dr. Uğur Güdükbay
Eylül, 2001

Askeri uygulamalar, ergonomik tasarım gibi geniş uygulama alanları olan insan modellerinin animasyonu bilgisayar grafiğinin en önemli konularından biri olmuştur. Bir eklemli vücut genellikle eklemlerle bağlanmış segmanlar seti olarak modellenir. Eklem açılarındaki değişim, vücuda yeni bir duruş verir. Bir animatör, yeni bir duruş için eklem açılarını belirleyebilir (ileri kinematik). Fakat, vücudu bir pozisyona yerleştirmek için gerekli kesin eklem açılarını tahmin etmek zordur. Bunun yerine bir animatör vücuttaki uç bir nokta için istenen bir pozisyonu tanımlayabilir ve sonra ters kinematik algoritması vücudu yerleştirmek için gerekli eklem açılarını hesaplar. Bu tezde, insan modellerinin animasyonu için doğrusal olmayan optimizasyon ile ters kinematik yönteminin gerçekleştirilmesi anlatılmaktadır. Bu yöntem, son-etkileyici ve hedef olarak adlandırılan son etkileyicinin istenen pozisyonu için bir potansiyel fonksiyon tanımlar. Doğrusal olmayan optimizasyon algoritması bu fonksiyonun değerini küçültmeye çalışır. Eklem açılarının üst ve alt limitleri olduğundan dolayı fonksiyon değeri sıfırlanamayabilir. Bu durumda algoritma en iyi çözümde durur (lokal optimum). Kullanıcı, eklem açılarına, algoritma tarafından tahmin edilmiş ilk değerlerinin ağırlıklarını değiştirerek öncelik verebilir. Böylece, eklem açılarının animatörün önceliğine göre değişmesi sağlanmaktadır.

Anahtar sözcükler: animasyon, insan hareketi, ters kinematik, doğrusal olmayan programlama, optimizasyon.

Türk Silahlı Kuvvetleri'ne

ve

Aileme.

ACKNOWLEDGEMENTS

I am very grateful to my supervisor, Assist. Prof. Dr. Uğur Güdükbay, for his invaluable support, guidance and motivation.

I also would like to thank my thesis committee members Prof. Dr. Bülent Özgüç and Assoc. Prof. Dr. Özgür Ulusoy for their valuable comments to improve this thesis.

I would like to mention some people who helped me during this study in different ways. I would like to thank Captain Gültekin Arabacı for his invaluable support who shares some parts of this research with me. I would also like to thank Captain Türker Yılmaz for his invaluable help on OpenGL libraries.

Finally, I cannot forget my love and my wife, Berrin Abalı. I would like to thank her for invaluable moral support and love.

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

DOF	: Degrees of Freedom
fps	: Frames per Second
CSG	: Constructive Solid Geometry
SVD	: Singular Value Decomposition
DFP	: Davidon-Fletcher-Powell algorithm
BFGS	: Broyden-Fletcher-Goldfarb-Shanno algorithm
θ_i	: ith joint angle
H	: Hessian matrix
A	: n-by-m constraint matrix where n denotes the number of joint angles in joint group and m denotes the sum of all active and inactive constraints
q	: The number of active constraints
a_j	: jth column vector of the constraint matrix
g	: Gradient of the objective function as a vector
d	: Search direction
$G(\theta)$	: Objective function
GUI	: Graphical User Interface

Chapter 1

Introduction

Animation of articulated figures has always been an interesting subject of computer graphics due to a wide range of applications. In film industry, animators spend much time in order to fill the key-frames between two adjacent different positions of a figure. The usage of articulated figures in simulated environments helps the accomplishment of educational purposes and ergonomic designs. During a presentation, an animated figure increases the understandability and the imagination of the audiences. One of the goals of computer graphics community is to model articulated figures, and to animate their actions realistically.

The level of automation in motion control of a human figure has a tradeoff between the user and the computer. On one side, the user is supposed to define all control factors for the movements of individual body parts during each time instance. In order to achieve realistic motions, the user must be highly experienced. On the other side, the user only gives a goal or a list of goals for the motion of the figure and the computer computes the parameters needed to reach the goals defined. However, the result may not be the same as what the user expects precisely.

Two basic approaches to motion control were developed. In one approach, the geometric properties of the articulated bodies are used for motion control. Positions and orientation of body segments, rotations and translations around

local and global coordinate frames are basic parameters of the approach. Interpolations, computing the Jacobian and the gradients, optimization, etc. are helpful mathematical tools for the methods of the approach. Neglecting the physical behaviors of the objects in motion is the basic weakness of the approach causing a serious loss of realism. The other approach simulates the physical behavior of the objects. Newton's laws are the fundamental principle of the approach. The forces and torques for rotations are linked to the resulting motion with Newton's equations. The approach generates realistic motions. However, the added complexity makes real time animation difficult.

One of the important issues in motion control of articulated figures is to handle the constraints. Constraints are the result of body segment limits. For example, head cannot turn back with an angle of 180-degree. While the animator gives a motion to a figure, constraints cannot be violated. Different methods have been used to handle the constraints. None of these methods is trivial.

The motivation behind this thesis is to implement an algorithm that reduces user's job during an animation. In order to prevent undesired motions, user can also interfere with some control factors. The user will not need to take care of constraints of body parts while moving the figure reach a goal. In order to handle constraints, a constraint-based nonlinear optimization algorithm is implemented.

1.1 Organization of The Thesis

Chapter 2 explains how human figure is modelled and is represented. Chapter 3 reviews computer animation techniques in general, and discusses their applicability to the concept of articulated figure animation. In Chapter 4, the inverse kinematics problem is discussed and common approaches to solving the problem are reviewed. In Chapters 5 and 6, an implementation of inverse kinematics using constraint-based nonlinear optimization is presented. Results of the implementation are presented in Chapter 7. Chapter 8 gives conclusion and future work. In Appendix A, the user interface of the system is described.

Chapter 2

Modelling of Articulated Bodies

In this chapter, some common mathematical notations on articulated body modelling and geometric body modelling techniques are introduced. Furthermore, articulated figure representation and data structures of our implementation are discussed.

2.1 Representing Articulated Figures

2.1.1 Mathematical Notation

In order to represent an articulated figure, a mathematical notation is necessary. Two common notations are Denavit-Hartenberg (DH) notation [10] and axis-position (AP) joint representation [29].

The most common kinematic representation in robotics is the notation of Denevit and Hartenberg [10]. The notation defines four parameters that construct the transformation matrix between two consecutive links:

- the angle of rotation for a rotational joint or distance of translation for a prismatic joint,
- the length of the link, or the distance between the axes at each end of a

link along the common normal,

- the lateral offset of the link, or the distance along the length of the axis between subsequent common normals, and
- the twist of the link, or the angle between neighboring axes.

First problem with this notation is that it only allows one DOF per joint. Joints with more than one DOF are represented with more than one joint at the same position, each of which has one DOF. The other problem is that it is suitable for chain type of links, however, it cannot incorporate branching joints.

Sims and Zeltzer [29] introduce axis-position (AP) joint representation which stores:

- the position of the joint,
- the orientation of the axis of the joint, and
- pointers to the link(s) that each joint is attached to.

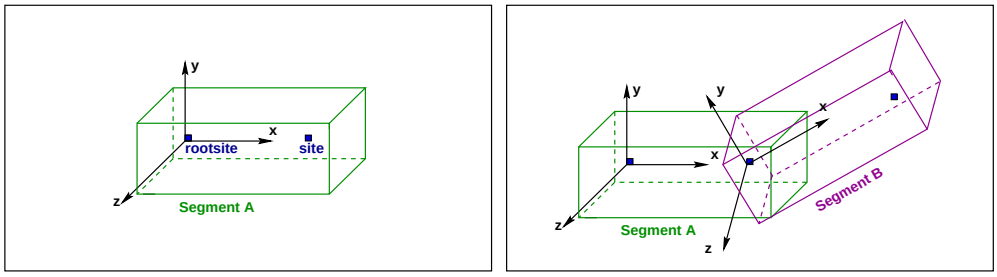
AP joint representation uses more parameters (three for position, three for axis orientation and one for joint angle) than those of DH notation and is more convenient for articulated figures.

In our implementation, AP joint representation is adopted with some modifications. Transformation matrix of a joint with respect to the root joint is stored for the sake of easy implementation. The details of this representation are explained in Subsection 2.1.2.

2.1.2 Representation of Articulated Figures in Our Implementation

Some common terms are helpful to represent an articulated figure. An articulated figure is constructed by segments that might be thought as body parts.

Each segment has at least two sites and one of them is rootsite of the segment (see Figure 2.1.a). A site is an attachment point on a segment. A joint connects the site of a segment to the site of another segment (see Figure 2.1.b). The location of the axes of the joints defines the placement of the segments. This provides flexibility on the length and the shape of the segments. All transformations are provided by the joints.



(a) (b)

Figure 2.1: (a) *segment representation*; (b) *two segments connected with a joint.*

Each joint can have six DOFs (three for rotational and three for translational). The number of DOFs of an articulated structure is the number of independent position variables necessary to specify the state of a structure [33]. Articulated figures have only rotational DOFs. A rotation with a corresponding angle around the axis of the DOF forms a transformation and the product of transformation matrices of the DOFs belonging to the same joint defines the location of the site and the segment. A DOF includes the rotation axis, current joint angle and the upper and lower limits of the joint angle (see Figure 2.7).

In order to use this segment and joint representation in the construction of an articulated figure, it is convenient to use a tree-structured hierarchy. A site is selected as root site of this hierarchical structure. Normally, the joints represent the edges and the segments represent the nodes in the tree structure (see Figure 2.3). Each joint has a local coordinate frame that describes the transformation of the segment with respect to its parent segment and has a global coordinate frame that describes the transformation of the segment with respect to the root segment. This hierarchical structure provides that the transformation at the parent node affects the displacements of child nodes.

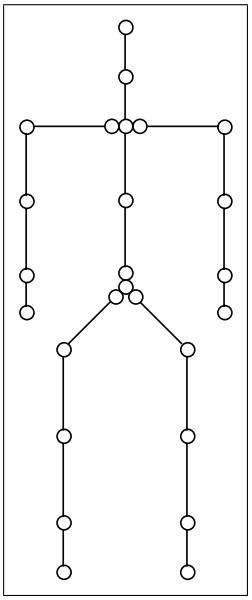


Figure 2.2: Placement of joints on the articulated figure.

In Figure 2.2, joints are represented with circles. Circles at the end of the limbs are end-effectors.

Each branch of the tree is defined as a joint group. Joint group provides a way to handle relevant joints as a whole. A joint group consists of not only joints but also a linear constraint matrix (see Figure 2.8). Linear constraint matrix is constructed by the upper and lower limits of the joint angles. Its use will be explained in detail in Chapter 5.

2.2 Data Structures

Each segment has an unique `name` (see Figure 2.4). The `rootsite` field is helpful to determine the displacement of segment. All sites belonging to a segment are stored in a linked list. The color of the segment is defined in `color` field. Since the other fields are related with the geometric model of a body (figure), they will be explained in Section 2.3.

The site is defined with `name`. The `segment` field indicates the segment to which the site belongs. The `rootjoint` field gives the joint that connects the site to the site of neighbor segment. If this site is the root site of the segment

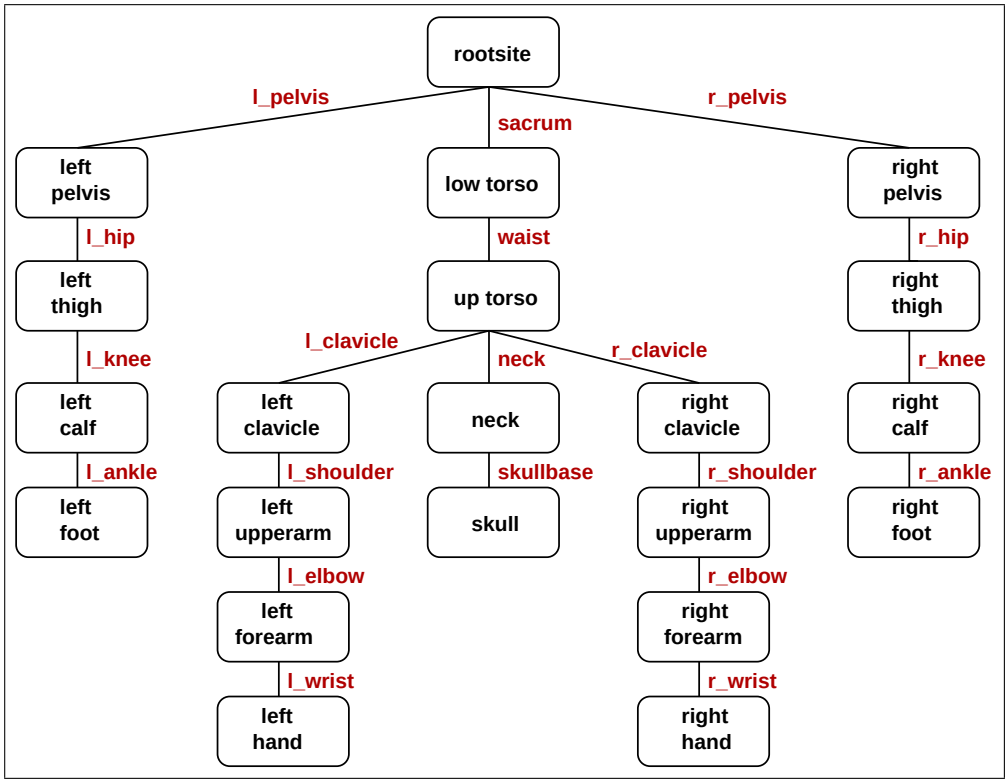


Figure 2.3: Tree-structured hierarchy of human figure.

```
struct segment {
    char *name;
    Figure *figure;
    Site *rootsite;
    List sites;
    char *filenamev;
    char *filenamef;
    float vertices[1000][3];
    int faces[2000][3];
    float normal[1000][3];
    float color[3];
    int nedges;
    int nnodes;
}
```

Figure 2.4: Segment data structure

```
struct site {
    char *name;
    Segment *segment;
    Joint *rootjoint;
    List joints;
    Matrix *global;
}
```

Figure 2.5: Site data structure

then the neighbor segment is the parent segment of the segment to which the site belongs. The site can be connected to more than one joint. These joints are stored in a linked list. `global` field stores the transformation of the site with respect to the root segment (see Figure 2.5).

Since all computations are done using the fields of the joints, the joint has a key role in the data structure. The `site1` and `site2` fields indicate the sites which are connected by the joint. The `rootjoint` field indicates the parent of the joint in tree-structured hierarchy. Joint's DOFs are stored as a linked list in the `dofs` field and the `ndofs` field stores the number of the DOFs of the joint. The `displacement` field is a vector that gives the translation of the joint with respect to the parent joint. `global` is a transformation matrix that gives the position and orientation of the joint with respect to the root segment. The `joint_group_it_belongs` field defines the joint group of the joint. Index of the joint in the joint group is stored in the `index_of_joint_in_JointGroup` field (see Figure 2.6).

The `type` field in `dof` structure defines that it is either rotational ('r') or translational ('t'). Since the joints in articulated figures have no translational DOF, this field currently has no usage. The `axis` of the DOF is set once and it never changes. The `angle` changes if a motion occurs on DOF. The `llimit` and `ulimit` fields give the lower and upper limits of DOF respectively. The pointer to the next DOF of the same joint is stored in `next` field (see Figure 2.7).

The Joint groups are numbered. Number of DOFs, linear constraints and

```
struct joint
{
    char *name;
    Site *site1, *site2;
    Joint *rootjoint;
    DOF *dofs;
    int ndofs;
    Vector displacement;
    Matrix global;
    JointGroup *joint_group_it_belongs;
    int index_of_joint_in_JointGroup;
}
```

Figure 2.6: Joint data structure

```
struct dof {
    char type;
    float axis[3];
    float angle;
    float llimit, ulimit;
    dof *next;
}
```

Figure 2.7: DOF data structure

```
struct group_member {
    Joint *joint;
}

struct joint_group {
    int group_number;
    int number_of_dofs;
    float group_angles[20];
    int number_of_linear_constraints;
    float linear_constraints_matrix[25][25];
    int number_of_members;
    GroupMember member[20];
}
```

Figure 2.8: Joint group data structure

members in the group are stored. Group angles are obtained from DOF angles of the joints in the group (see Figure 2.8).

2.3 Geometric Body Modelling

In order to animate a human figure, the geometry of the body must be modelled. This encapsulates constructing a surface or a volume geometry for the human body shape.

2.3.1 Geometric Body Modelling Techniques

In this subsection, current geometric modelling schemes are briefly reviewed. Geometric models can be classified into three categories:

Stick models: Segments are defined as lines. Joints link these segments. Figure looks like a skeleton. Since it is not realistic, usage of it is not common.

Surface models: Surface models can be grouped as polygons and curved surfaces.

The polygonal models are defined as networks of polygons forming 3D polyhedra. Each polygon consists of some connected vertex, edge, and face structure [2].

Mathematical formulations are used for constructing true curved surfaces called patches. There are many formulations of curved surfaces, including: Bezier, Hermite, bi-cubic, B-spline, Beta-spline, and rational polynomial [5, 11].

Volume models: Volume models can be grouped as Voxel models and CSG. In Voxel models, space is completely filled by a tessellation of cubes called voxels (volume elements). In CSG, there is no requirement to tessellate the entire space. Also, the primitive objects are not limited to cubes. There are many number of simple primitives such as cube, sphere, cylinder, cone, half-space, etc. Each primitive is transformed or deformed and positioned in space.

2.3.2 Surface Scheme with Triangular Polygons

In our application, a polygonal representation is used to model the geometry.

Each segment has a physical shape constructed by triangular polygons. Two files are read for each segment (see Figure 2.4). One of the files consists of point coordinates with respect to the `rootsite` local coordinate frame (see Figure 2.9). These coordinates are written to a 2D array, `vertices` (see Figure 2.4). The other file stores the polygon (face) information where triangles are used as polygons (see Figure 2.10). These connections are written to another 2D array, `faces` (see Figure 2.4). The normals of the faces are stored in `normal` field of the `segment` structure.

The resulting shape for the vertex and face lists presented in Figures 2.9 and 2.10 can be seen in Figure 2.11.

```
v  0.0   0.0   0.0
v  1.0   0.0   0.0
v  0.0   1.0   0.0
v  0.0   0.0   1.0
```

Figure 2.9: An example file showing the 3D coordinates of points.

```
f 0 1 2
f 1 2 3
f 0 1 3
f 0 2 3
```

Figure 2.10: An example file that stores the polygon (face) information.

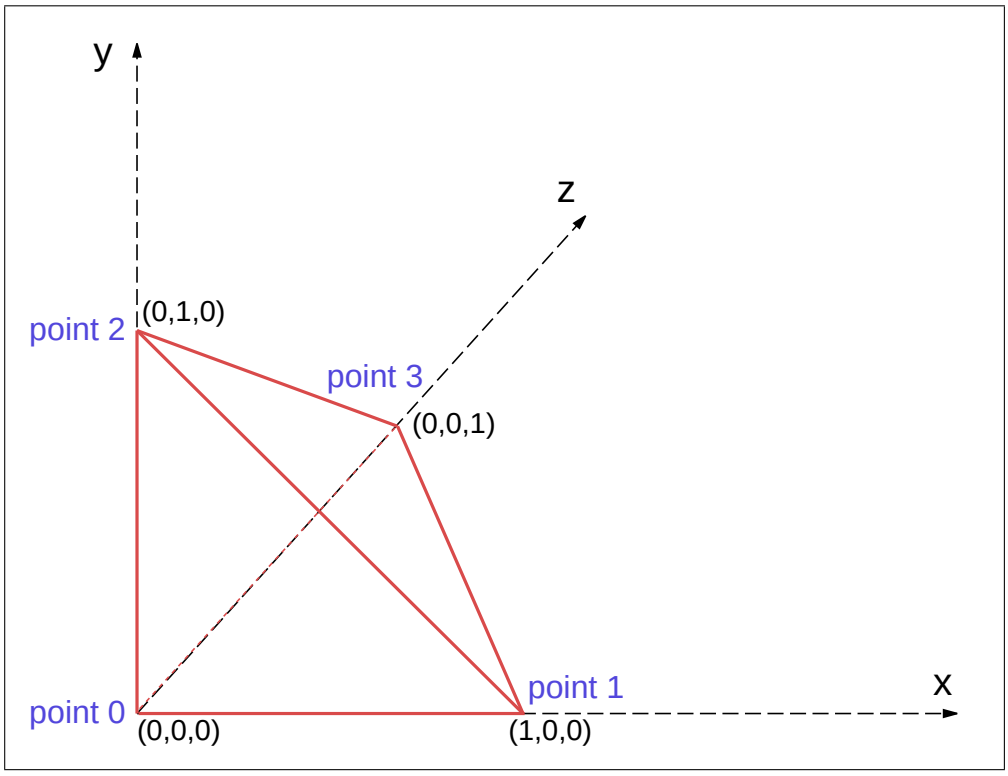


Figure 2.11: An example shape read from files.

Chapter 3

Human Animation Techniques

In this chapter, we give an overview of the previous work that has been produced in this area.

In computer graphics, a variety of human animation techniques have been produced. These techniques can be classified into three main categories:

- kinematics,
- dynamics, and
- motion capture.

3.1 Kinematic Methods

Kinematics studies the geometric properties of the motion of the objects independent from the underlying forces that cause the motion.

3.1.1 Forward Kinematics

Forward kinematics explicitly defines the state vector of an articulated figure at a specific time. The state vector is

$$\Theta = (\theta_1, ..., \theta_n), \tag{3.1}$$

where Θ is the set of joint angles including independent parameters defining the positions and orientations of all joints belonging to the figure.

A set of joints linked to each other hierarchically, forms a chain (i.e. a branch of tree at Figure 2.3). The most distal end of this chain is called the end-effector. With a given Θ, the cascaded transformations of joints in the chain affect the displacement of end-effector (Equation 3.2).

$$X = f(\Theta). \tag{3.2}$$

This specification is done with the manual input of a small set of poses (key-framing) by animator explicitly. In order to generate intermediate poses (in-betweens), interpolation techniques are used.

The choice of an adequate interpolation technique is a problem. The interpolated values for a single DOF between two key-frames form a trajectory curve. Most of the studies are concentrated on to form a proper shape for this trajectory and to control the variations of the speed along the trajectory in order to obtain a realistic motion [4, 20, 22, 30]. Linear interpolation technique, piecewise splines and double-interpolant methods with some modifications have been used to generate the intermediate poses.

Forcing the user to specify values for parameters is often inconvenient, especially for tasks with too many DOFs. Too many parameters to control, drive the animator to make errors. In addition to this, combined effects of transformations from root to end-effector make it difficult to control the positional constraints while creating a key-frame. During interpolation, obtaining the true trajectory for realistic motion is also hard job to achieve.

Forward kinematics is successful for animating simple objects, but it is not really a good choice for animating highly articulated figures.

3.1.2 Inverse Kinematics

In inverse kinematics, the desired position and orientation of the end-effector are given by the user. Inverse kinematics computes all joint angles in the chain that orient the end-effector to the desired posture. In order to derive Θ with a given X, the inverse of Equation 3.2 is required (Equation 3.3).

$$\Theta = f^{-1}(X) \tag{3.3}$$

Solving the Equation 3.3 is quite difficult. Since the function in Equation 3.3 is nonlinear, there may be more than one solution set of Θ for a given X (Figure 3.1). In one approach, as the nonlinear property of function makes

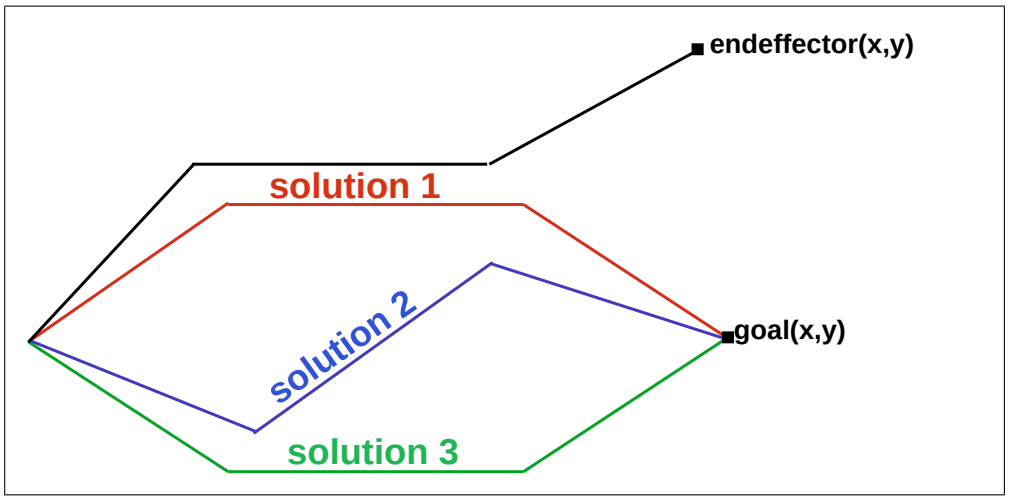


Figure 3.1: Given goal, more than one solution.

the solution difficult, the problem can be made linear by localizing around the current operating position [34]. Girard's PODA animation system is an example using this approach [15].

Another numeric approach is nonlinear optimization technique. This approach tries to minimize a function of relation between the end-effector's position and the user defined goal. It applies iterative non-linear optimization techniques to obtain the minimum. Jack[1] animation system developed at the University of Pennsylvania uses this approach [1, 38].

Tolani et al. [32] also offer a combination of analytic and numerical methods

[1]Jack is a registered trademark of Transom Inc.

to solve inverse kinematics problems. These approaches will be discussed at Chapter 4 in detail.

3.2 Dynamic Methods

In kinematic approaches, articulated figures are animated with geometric computations. However, laws of physics have an important effect on the motion of articulated figure in reality. In computer animation, simulating the physical behavior of objects can produce more realistic motions.

The fundamental principle of dynamics is Newton's law which can be formulated as;

$$F = ma, \tag{3.4}$$

where F represents the force applied to an object, m represents mass of the object and a represents its acceleration, which is the second derivative with respect to time of the position vector. The force is linked to the resulting motion (Equation 3.4). The same equation can also be formulated as follows:

$$T = i\ddot{\theta}, \tag{3.5}$$

where T represents the torque, i represents the inertia matrix, and $\ddot{\theta}$ represents the angular acceleration. Torque is linked to rotation (Equation 3.5).

3.2.1 Forward Dynamics

Forward dynamics applies Equations 3.4 and 3.5 in order to calculate the motion with a given force. Equations 3.4 and 3.5 are only for a rigid body. A hierarchy of rigid parts linked by joints is constructed in order to apply them to an articulated figure.

Animator applies external forces to rigid bodies. The motion is calculated step by step in time. At each step, acceleration is computed with respect to the applied forces. Computed acceleration is twice integrated to the position of the rigid body. The fundamentals of forward dynamics can be found in [35].

It is harder to solve the equation for an articulated structure than for a single rigid body. There is one equation for each DOF and the obtained acceleration from the equation affects the forces applied to adjacent segments. High computation cost of forward dynamic approaches prevents them to be used in real time applications.

3.2.2 Inverse Dynamics

Inverse dynamics applies Equations 3.4 and 3.5 in order to calculate the forces with a given motion. Some of the notable research using inverse dynamics can be found in [3, 6, 18, 36].

Witkin, Fleischer, and Barr [36] uses "energy" constraints to assemble 3D models, for changing the shape of parametrically-defined primitive objects. Constraints are expressed as energy functions, and the energy gradient followed through the model's parameter space. The constraints are satisfied if and only if the energy function is zero.

Isaacs and Cohen [18] does physical simulation of rigid bodies, for the special case of linked systems without closed kinematic loops. They embedded the key-frame animation system within a dynamic analysis by the help of kinematic constraints. Joint limit constraints are also handled through kinematic constraints. This provides to escape to define additional forces. Instead, kinematic constraints remove constrained DOFs of joints, and inverse dynamic handles unconstrained DOFs.

Barzel and Barr [6] build objects by specifying geometric constraints. They classified these constraints as *point-to-nail* that fixes a point on a body to a user-specified location in space, *point-to-point* that forms a joint between two bodies, etc. Each rigid body is independently simulated at each time step. A rigid body is subjected to external forces and constraint forces. These forces act until all the constraints are satisfied. Constraint forces are solved from a set of dynamic differential equations.

In addition to these, Witkin and Kass [37] propose a new formulation called

spacetime constraints. The main idea is to compute the figure motion and the time varying muscular forces on the whole animation sequence instead of doing it sequentially in time. The discrete values of forces, velocities and position over time are put in a large vector of unknowns. A set of constraints between these unknowns is specified. The vector of unknowns is computed with a constrained optimization. A cost function is specified for minimization. This function consists of the sum of squared muscular forces over time.

As it has been emphasized in forward dynamics, approaches based on dynamic simulation suffer from high computational cost compared with the kinematic approaches. However, the motion of the figure in dynamic approaches is more realistic than that in kinematic approaches.

3.3 Motion Capture

Since each individual has his own motion style, kinematics and dynamics stay insufficient to detect this competence. Moreover, for many different motions, it cannot be possible to obtain them realistically. In recent years, progress in motion capture techniques make it possible to use human motion data directly.

Magnetic and optical technologies make it possible to obtain and to store positions and orientations of points on the human body. However, the stored data are raw and need to be processed. Mostly, the synthetic skeleton does not match with the real one. In [7, 24], the synthetic skeleton is adapted to another one by recovering angular trajectories. They used an inverse kinematic optimization algorithm to obtain the correct angular trajectories. It is also possible to have some errors during capturing because of calibration error, electronic noise etc. These techniques also take care of this problem.

In recent years, two new techniques have been introduced in order to arrange interaction between synthetic actors: *motion blending* and *motion warping*. Motion blending technique constructs a database of characteristic motions and produces new motions by interpolating between parameters of this defined motions. Motion warping obtains well-known trajectories and changes the

motion by modifying these trajectories.

Animation techniques based on motion data produce realistic motions. However, this realism depends on the modifications done by the animator and capturing the correct motion data.

Chapter 4

Inverse Kinematics

This chapter discusses the solution methods for the inverse kinematics approach. These methods are classified into three categories:

- analytical methods,
- numerical methods, and
- a combination of analytic and numerical methods.

Analytical methods can only be used for very simple articulations, like a two-link arm. For more complex articulations, no analytical solutions exist.

4.1 Numerical Methods

4.1.1 Linearized solutions

Inverse kinematics problem is nonlinear since the joint transformations involve rotations. This method tries to solve the nonlinear inverse kinematics problem with linear solution. As a first step, Equation 3.2 is differentiated with respect to Θ,

$$dX = J(\Theta)d\Theta, \tag{4.1}$$

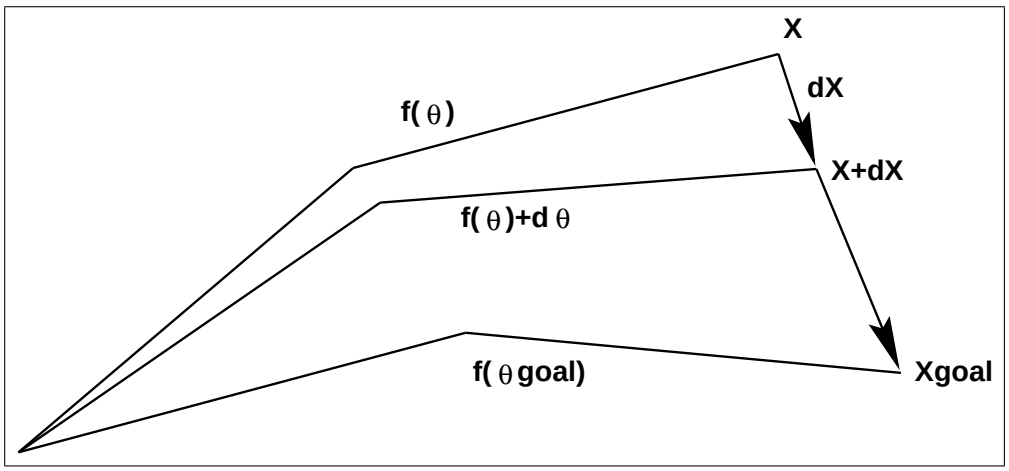


Figure 4.1: Iteration steps towards the desired goal.

where X is the end-effector position and orientation and Θ is the vector of joint angles from the root of the hierarchy to the end-effector and Jacobian J is a matrix of partial derivatives relating differential changes of Θ to differential changes in X

$$J = \frac{\partial f}{\partial \theta}. \tag{4.2}$$

If we invert Equation 4.1 and iterate towards a final goal position with incremental steps, inverse kinematics problem can be solved linearly (Figure 4.1).

$$d\Theta = J^{-1}(dX). \tag{4.3}$$

Usually, it is not possible to take the inverse of Jacobian matrix. Because in order to invert Jacobian matrix, it is supposed to be square and nonsingular. However, it is commonly rectangular, because the dimension of Θ is usually larger than that of X. In this situation, pseudoinversion techniques are brought into play [15, 19]. Instead of J^{-1}, pseudoinverse of Jacobian written as J^{+}, is used and it is defined as follows:

$$J^{+} = J^{T}(JJ^{T})^{-1}. \tag{4.4}$$

Then, Equation 4.3 becomes,

$$d\Theta = J^{+}(dX). \tag{4.5}$$

Pseudoinverse solutions have the following problem. The rectangular structure of the Jacobian causes redundancy. A manipulator is considered kinematically

redundant when it possesses more DOFs than needed to specify a goal (Figure 3.1). It is often useful to consider exploiting the redundancy in an attempt to satisfy some secondary condition. This can be accomplished by adding a new term to Equation 4.5,

$$d\Theta = J^{+}(dX) + (I - J^{+}J)dZ, \tag{4.6}$$

where I is the identity matrix, $(I - J^{+}J)$ is a projection operator on the null space of the linear transformation J, and is called the homogeneous part of the solution. dZ describes a secondary task in the joint variation space. Whatever the secondary task is, the second term does not affect the achievement of the main task. In addition to prevent the redundancy, the secondary task is used to account for joint angular limits [15] and to avoid kinematic singularities [27].

Another problem is the singularity of the Jacobian. Even though the pseudoinverse can be used when the Jacobian is singular, as the articulation moves, there may be sudden discontinuities in the elements of the computed pseudoinverse due to the changes in the rank of the Jacobian. Physically, the singularities usually occur when the articulation is fully extended or when the axes of separate links align themselves [33].

With singular value decomposition (SVD), it is possible to analyze whether Jacobian matrix is singular. The details of SVD approach can be found in [25]. Generally, the approaches to prevent the singularity track this situation and when it occurs, they try to avoid from singularity. However, the analysis of the Jacobian for singularity condition brings extra processing cost.

4.1.2 Nonlinear Optimization

Optimization based methods approach the problem as a minimization problem. Let $e(\theta)$ be the positional and orientational definition of end-effector depending on joint angles and g be the positional and orientational definition of desired goal,

$$P(e(\theta)) = (g - e(\theta))^2, \tag{4.7}$$

where $P(e(\theta))$ is a potential function that gives the distance (positional and orientational) between the end-effector and the goal. If the value of potential

function is zero, then the goal is reached. If the goal is not reachable because of the joint limits, the potential function value is tried to be minimized sufficiently. The optimization problem can be formulated as follows [14]:

$$\begin{aligned} &\text{Minimize} && P(e(\theta)), \\ &\text{subject to} && l_i \leq \theta \leq u_i \qquad \text{for} \qquad i = 1, \ldots, n. \end{aligned} \tag{4.8}$$

Here, l_i and u_i are the lower and upper limits of the joint angles, respectively.

There are solvers, for example in MATLAB package [31], in order to handle constraint-based nonlinear optimization problems. They can be used as a black box and be integrated to an animation package. However, it is obvious that this integration would increase the computation cost drastically. Therefore, we choose to embed an optimization algorithm into our implementation. The details of the nonlinear optimization method will be explained in the next chapters.

4.2 A Combination of Analytic and Numerical Methods

Tolani et al. [32] offer a combination of analytic and numerical methods to solve generalized inverse kinematics problems including position, orientation and aiming constraints. They develop a set of algorithms for arm or leg.

They approach the problem with transformation and rotation matrices. A seven DOF limb is defined that has 3 joints. Each of first and last joints has three DOF and joint at the middle has one DOF rotating around y axis. Let us call the joints as J_1, J_2 and J_3, respectively. The equation that has to be solved in order to reach the defined goal for the limb is

$$T_1 A T_y B T_2 = G, \tag{4.9}$$

where T_1 and T_2 denote the rotation matrices of J_1 and J_3 as functions of three DOFs belonging to the related joint. T_y is the rotation matrix of J_2 defined as a function of one revolutionary DOF. A and B are the constant transformation

matrices from J_1 to J_2 and J_2 to J_3, respectively. Finally, G is the matrix of the desired goal.

In order to solve Equation 4.9 analytically, trigonometric equations are generated from it. These analytic equations are enough for positional goals, and positional and orientational goals. However, for aiming goals, and positional and partial orientational goals, a combination of analytic and numeric methods is used. Seven variables are reduced to two variables with trigonometric equations, The details of this method can be found in [21]. An unconstrained optimization algorithm is applied to solve for these two variables.

Analytical methods offer high performance for arm or leg limb in computer animation but may require special kinematic structure for the entire body because of the huge trigonometric computations needed. Tolani et al. [32] offers an approach in which an inverse kinematics problem is broken into subproblems [39] that can be solved with the analytical method.

Chapter 5

Implementation Details

The animation system consists of two main parts:

- the Function Generator Module, and
- the Nonlinear Programming Module.

Figure 5.1 gives an overall structure of the animation sytem.

5.1 Function Generator Module

First of all, the human figure is placed to an initial configuration according to the joint parameters entered by the user. The following parameters are entered by the user for each joint to specify the initial configuration of the human figure:

- the angle θ that rotates around an arbitrary vector (x, y, z),
- `displacement` vector with respect to the parent joint, and
- the parent joint of the joint.

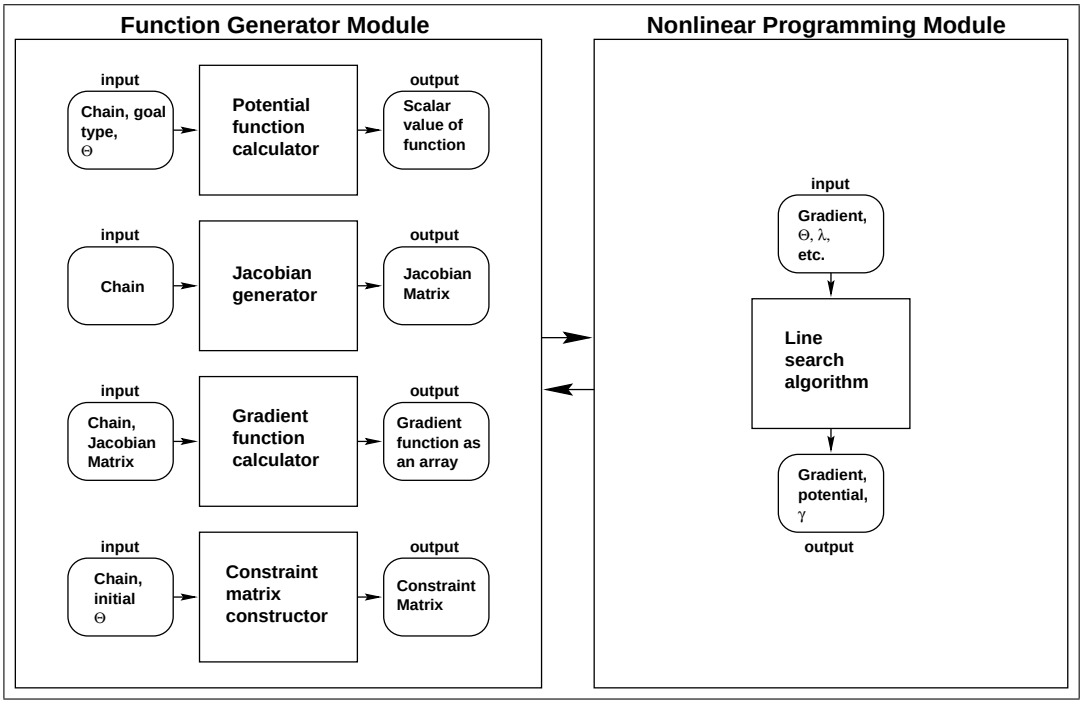


Figure 5.1: The overall structure of the animation system.

A transformation matrix M is constructed for each joint,

$$M_{Ji} = T_{Ji}(t_x, t_y, t_z) R_{Ji}(\theta, rotation\_axis), \tag{5.1}$$

where $T_{Ji}(t_x, t_y, t_z)$ is four by four translation matrix with respect to the parent joint, $R_{Ji}(\theta, rotation\_axis)$ is four by four rotational matrix that gives the rotation around the axis of the DOF. If we concatenate the individual rotation and translation matrices, then the matrix M_{Ji} will be a four by four matrix that has the following form:

$$\mathbf{M_{Ji}} = \begin{pmatrix} & & & 0 \\ & R_{Ji} & & 0 \\ & & & 0 \\ t_x & t_y & t_z & 1 \end{pmatrix}. \tag{5.2}$$

If the joint has more than one DOF then the general transformation matrix for the joint is obtained by cascaded multiplication of the rotation matrices for each of the principal axes (Equation 5.3), and then by concatenation of the translation and rotation matrices (Equation 5.4).

$$R_{Ji} = R^z_{Ji}(\theta_z, z\_axis).R^y_{Ji}(\theta_y, y\_axis).R^x_{Ji}(\theta_x, x\_axis), \tag{5.3}$$

$$M_{Ji} = T_{Ji}(t_x, t_y, t_z)R_{Ji}. \tag{5.4}$$

The implementation of the system can be explained through a simple example. Consider a three-joint articulated structure. Joint parameters of the initial configuration is given in (Figure 5.2). If we write Equation 5.1 for joint J2 in our example, then we obtain

$$\mathbf{T_{J2}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 5 & 0 & 1 \end{pmatrix}, \text{ and } \mathbf{R_{J2}} = \begin{pmatrix} 0.707 & 0.707 & 0 & 0 \\ -0.707 & 0.707 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$M_{J2} = T_{J2}(0, 5, 0).R^z_{J2}(45, z\_axis),$$

$$\mathbf{M_{J2}} = \begin{pmatrix} 0.707 & 0.707 & 0 & 0 \\ -0.707 & 0.707 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 5 & 0 & 1 \end{pmatrix}.$$

M_{Ji} only gives the transformation of ith joint with respect to $(i-1)$th joint. In order to calculate transformation of jth joint with respect to ith joint, we should do cascaded multiplications:

$$M_{Jij} = M_{Ji}.M_{Ji+1}...M_{Jj-1}.M_{Jj}. \tag{5.5}$$

If the ith joint is the root joint, then M_{Jij} gives the global position and orientation of jth joint with respect to the root. If we apply Equation 5.5 to our example then we obtain

$M_{J12} = M_{J1}.M_{J2}$,

$M_{J13} = M_{J1}.M_{J2}.M_{J3}$,

$$\mathbf{M_{J1}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \mathbf{M_{J2}} = \begin{pmatrix} 0.707 & 0.707 & 0 & 0 \\ -0.707 & 0.707 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 5 & 0 & 1 \end{pmatrix}, \text{ and}$$

```
Joint 1(root)
{
  name:           J1
  rootjoint:      J1
  displacement:   {0,0,0}
  number of dofs: 1
  axis of dof:    {0,0,1}
  angle of dof:   90 degree
}

Joint 2
{
  name:           J2
  rootjoint:      J1
  displacement:   {0,5,0}
  number of dofs: 1
  axis of dof:    {0,0,1}
  angle of dof:   45 degree
}

Joint 3 (end-effector)
{
  name:           J3
  rootjoint:      J2
  displacement:   {-4,4,0}
  number of dofs: 1
  axis of dof:    {0,0,1}
  angle of dof:   0 degree
}
```

Figure 5.2: Joint parameter values for the initial configuration of a three-joint articulated structure.

$$\mathbf{M_{J3}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -4 & 4 & 0 & 1 \end{pmatrix}.$$

$$\mathbf{M_{J12}} = \begin{pmatrix} -0.707 & 0.707 & 0 & 0 \\ -0.707 & -0.707 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 5 & 0 & 1 \end{pmatrix}, \text{ and } \mathbf{M_{J13}} = \begin{pmatrix} -0.707 & 0.707 & 0 & 0 \\ -0.707 & -0.707 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -4 & 9 & 0 & 1 \end{pmatrix}.$$

These cascaded matrix multiplications give the global position and orientation of related joint with respect to the root joint. After the initial configuration of the figure, new DOF axes of each joint are determined and set from obtained `global` matrix of the joint. These DOF axes are never changed again. When a motion is detected, only angles are changed and new position and orientation of each joint is constructed. In our example, if the joint J3 has one DOF around x axis then, the axis of the DOF will be set to $(-0.707, 0.707, 0)$. During a motion, any angle set for the DOF will rotate around $(-0.707, 0.707, 0)$.

5.1.1 Potential Function Calculation

Potential function $P(e(\theta))$ is a function of the difference between the end-effector and the goal positions and orientations. It should be a nonnegative real number. The motivation behind the nonlinear optimization is to decrease this difference by trying to obtain a value for the potential function as close to zero as possible. In order to measure this difference, first of all we have to define the end-effector. Defined as the most distal joint in the chain, the end-effector can be thought as a 9D vector on the distal segment. First three components define the position of it as a positional vector. Second and third triples are the unit vectors specifying the orientation. The angle between these two unit vectors should be 90 degrees so that they can specify the orientation.

It is obvious that the end-effector is a function $(e(\theta))$ of the state vector that is defined as in Equation 3.1. An instance of the joint angles θ of all the

joints in the chain determines $e(\theta)$.

For each joint in the chain, we have a rotation matrix for rotation angle θ around the joint axis, and cascaded multiplication of rotation matrices, corresponding to as many as the number of DOFs of the joint, construct a transformation matrix with respect to the parent joint. Cascaded multiplication of these transformation matrices from the root joint to the related joint gives the transformation matrix of related joint with respect to the root joint. Multiplication of the global matrix, which is constructed at initial position, with the transformation matrix gives the new position and orientation of the related joint. If this related joint is end-effector, we can obtain position vector and the two unit vectors of the end-effector from all these matrix multiplications. For example, the joint angles belonging to the chain of joints from sacrum to the left wrist changes the position and orientation of joints in the chain. Finally, the position and orientation of the left hand $e(\theta)$ are obtained.

Since the goal is definite constant and the end-effector position and orientation change with the joint angles, we can write the potential function as a function of the end-effector $P(e(\theta))$. Usually, all components of the end-effector are not used in order to compute the potential function because of different types of goal.

In order to calculate the potential function $P(e(\theta))$, we have to define the goal types. Although more types of goal are defined in [38], we defined two types of goal in our implementation because of time restrictions:

- positional goals, and
- positional and orientational goals.

A positional goal is defined as a 3D point vector in space. Therefore, only the positional vector of end-effector is used in order to compute potential function for a positional goal. If we define the positional goal as r_g and the position vector of end-effector as r_e then the potential function becomes,

$$P(r_e) = (r_g - r_e)^2. \tag{5.6}$$

Since the value of potential function is supposed to be a nonnegative scalar value, it is computed as the square of the difference.

Even though it is possible to define the orientational goal separately, we implemented the combination of positional and orientational goals. In addition to the coordinates of the goal r_g in space, orientation of the goal is defined by two orthonormal vectors x_g and y_g. Two orthonormal unit vectors of end-effector x_e and y_e are used together with positional vector r_e. The potential function becomes

$$P(r_e, x_e, y_e) = w_p(r_g - r_e)^2 + w_o((x_g - x_e)^2 + (y_g - y_e)^2). \tag{5.7}$$

where w_p and w_o are weights of position and orientation, respectively. The priorities of position and orientation are adjusted by the weights and the sum of w_p and w_o is equal to one.

The problem with equation 5.7 is that the values generated by the orientation part are too small according to the values generated by the position part. In order to reach to one unit difference at position part, one radian difference is needed at orientation part. To arrange this, the term corresponding to the orientation part is multiplied with a scalar value $k = 360/(2\pi d)$, compensates one length unit to d degrees. Then, for positional and orientational goals, potential function becomes

$$P(r_e, x_e, y_e) = w_p(r_g - r_e)^2 + w_o k^2 \left((x_g - x_e)^2 + (y_g - y_e)^2\right). \tag{5.8}$$

5.1.2 Jacobian Generation

Chosen constraint based nonlinear optimization algorithm needs the derivatives of $e(\theta)$ with respect to the joint angles, $\Theta = (\theta_1, \theta_2, \ldots, \theta_n)$,

$$\frac{\partial e}{\partial \theta} = \begin{pmatrix} \frac{\partial e}{\partial \theta_1} & \frac{\partial e}{\partial \theta_2} & \cdots & \frac{\partial e}{\partial \theta_n} \end{pmatrix}. \tag{5.9}$$

$\frac{\partial e}{\partial \theta}$ is called the Jacobian matrix. As we know from the Section 5.1.1, r_e is the point vector of end-effector, x_e and y_e are the unit vectors of end-effector $e(\theta)$. And they are computed with cascaded multiplications of four by four homogeneous matrices. Let the rotation axis (axis of DOF) of i^{th} joint angle

θ_i in the chain be unit vector u. Then, the derivatives of $e(\theta)$ with respect to θ_i become

$$\frac{\partial r_e}{\partial \theta_i} = u \times (r_e - r_i), \tag{5.10}$$

$$\frac{\partial x_e}{\partial \theta_i} = u \times x_e, \tag{5.11}$$

$$\frac{\partial y_e}{\partial \theta_i} = u \times y_e. \tag{5.12}$$

Each of Equations 5.10, 5.11, and 5.12 is a vector equation. For a joint group with n joint angles, the dimension of the Jacobian matrix is $9 \times n$.

5.1.3 Gradient Function Calculation

The gradient of the potential function is necessary for the nonlinear optimization algorithm implemented. Gradient of the potential function $P(r_e, x_e, y_e)$ is a 9×1 column vector formed by partial derivatives of the potential function with respect to the r_e, x_e and y_e:

$$\nabla_{\mathbf{e}(\theta)} = \begin{pmatrix} \frac{\partial P}{\partial r_e} \\ \frac{\partial P}{\partial x_e} \\ \frac{\partial P}{\partial y_e} \end{pmatrix} = \begin{pmatrix} \nabla_r P(r_e) \\ \nabla_x P(x_e) \\ \nabla_y P(y_e) \end{pmatrix}. \tag{5.13}$$

Each of partial derivatives $\partial P/\partial r_e$, $\partial P/\partial x_e$, $\partial P/\partial y_e$ is a 3×1 column vector. For positional goals, the values of $\partial/\partial x_e$ and $\partial/\partial y_e$ are set to zero.

For positional goals, gradient of Equation 5.6 is

$$\nabla_r P(r_e) = 2(r_e - r_g). \tag{5.14}$$

For positional and orientational goals, gradient of Equation 5.6 is formed by the following equations:

$$\nabla_r P(r_e) = 2(r_e - r_g), \tag{5.15}$$

$$\nabla_x P(x_e) = 2k^2(x_e - x_g), \tag{5.16}$$

$$\nabla_y P(y_e) = 2k^2(y_e - y_g). \tag{5.17}$$

5.1.4 Constraint Matrix Construction

Let us equalize $P(e(\theta))$ to a function of joint angles

$$G(\theta) = P(e(\theta)). \tag{5.18}$$

As it is explained before if we decrease the value of $G(\theta)$ to zero then we reach the goal. However, it is not always possible because of the joint angle limits. For this reason, $G(\theta)$ is minimized:

$$\begin{aligned} &\text{Minimize} \qquad G(\theta), \\ &\text{subject to} \qquad l_i \leq \theta_i \leq u_i \qquad \text{for} \qquad i = 1, \ldots, n. \end{aligned} \tag{5.19}$$

Here l_i and u_i are lower and upper joint limits, respectively. In order to use the joint limits in our nonlinear optimization algorithm, they have to be rearranged as linear equality and inequality constraints:

$$\begin{aligned} a_i^T\theta &= b_i \qquad \text{for} \qquad i = 1, 2, \ldots, l \\ a_i^T\theta &\leq b_i \qquad \text{for} \qquad i = l+1, l+2, \ldots, k \end{aligned} \tag{5.20}$$

where a_is are $n \times 1$ column vectors and the total number of them is k. The total number of θs is n. l of a_is represent equality constraints. Inequality constraint representation is in the form of $-\theta_i \leq -l_i$, and $\theta_i \leq u_i$.

In order to clarify, let us support formation of constraints with an example:

$\theta_1 = \pi$,

$\pi/4 \leq \theta_2 \leq \pi/2 \quad \Rightarrow \quad -\theta_2 \leq -\pi/4, \quad \theta_2 \leq \pi/2,$

$-\pi/4 \leq \theta_3 \leq \pi/4 \quad \Rightarrow \quad -\theta_3 \leq \pi/4, \quad \theta_3 \leq \pi/4,$

Then $A^T\Theta \leq B$,

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} \leq \begin{pmatrix} \pi \\ -\pi/4 \\ \pi/2 \\ \pi/4 \\ \pi/4 \end{pmatrix}.$$

5.2 Nonlinear Programming Module

There are two major families of algorithms for multidimensional nonlinear minimization with calculation of first derivatives. Both families require a line search sub-algorithm. The first family goes under the name conjugate gradient methods. The second family goes under the names quasi-Newton or variable metric methods. Since both of the methods are for unconstrained nonlinear equations and there is no superiority to each other, we used to a modification of variable metric method which is more common than conjugate gradient method.

Variable metric methods come in two main flavors. One is the Davidon-Fletcher-Powell (DFP) algorithm. The other goes by the name Broyden-Fletcher-Goldfarb-Shanno (BFGS). The BFGS and DFP are used together in our implementation.

Algorithm can be briefly defined in five steps:

1. Guess an initial feasible Θ.
2. Form an approximation to Hessian matrix H updated with constraints.
3. Compute a search direction $d = -Hg(\theta)$.
4. Find $\theta = \theta_{old} + \lambda d$ using a line search to insure sufficient decrease.
5. Check the result whether the function is at the minimum (Kuhn-Tucker point), then stop else go to step 2.

$g(\theta)$ at step 3 is the gradient of objective function $G(\theta)$,

$$g(\theta) = \nabla_\theta G \quad \Rightarrow \quad g(\theta) = \left(\frac{\partial e}{\partial \theta}\right)^T \nabla_e P(e(\theta)), \tag{5.21}$$

where the Jacobian matrix $\partial e/\partial\theta$ is computed by the Jacobian Generator Module and $\nabla_e P(e(\theta))$ is computed by the Gradient Function Calculator Module. In fact, the computation of $g(\theta)$ belongs to the Gradient Function Calculator Module. At each iteration, the Nonlinear Programming Module requests the objective function and its gradient from the Function Generator Module.

At each iteration, the value of objective function $G(\theta)$ decreases monotonically, and stops at a minimum value.

Chapter 6

The Nonlinear Optimization Algorithm

6.1 Determining an Initial Feasible Point

In nonlinear optimization, the key to stability and rapid convergence is an initial guess of joint angle set not too far from the final result. Initially, we assign zero to each joint angle and select the goal next to end-effector. However, the important condition is that zero value will satisfy the equality and inequality constraints, else the value near zero is selected. If the selected goal is too far away from end-effector, we divide the goal into subgoals and run the algorithm more than once for in-between goals until reaching the desired goal. The joint angles found for an in-between goal become the initial feasible joint angle set for the next in-between goal. If the motion is changed to another direction by the user, the zero values are assigned to the joint angle set again and the optimization process continues. An initially selected joint angle is feasible if it satisfies equality and inequality constraints.

Another property of the algorithm is that the user interacts with initial joint angles by assigning a scalar multiplier to them, meanwhile the implementation takes care of joint limits. Nonlinear algorithm gives higher priority to the variables that have greater partial derivatives. When the multiplier scales the

joint angle, it also scales the derivative as well. The user can arrange the priority of joint angles with this method. This provides the user to escape creating unrealistic motions.

6.2 Active Constraints

If a constraint is an equality constraint or in an inequality constraint, given θ_i satisfies the equality in border, then this constraint is active. For example, at an inequality constraint $\theta_i \leq u_i$, if with a given θ_i, $\theta_i = u_i$ equality is satisfied then constraint becomes active. Nonlinear optimization algorithm that we choose handles the active constraint. This situation affects our constraint matrix and needs a modification on constraint matrix. Let us examine the example given in Subsection 5.1.4:

$$\theta_1 = \pi \qquad \text{(constraint 1)},$$

$$-\theta_2 \leq -\pi/4 \quad \text{(constraint 2)},$$

$$\theta_2 \leq \pi/2 \qquad \text{(constraint 3)},$$

$$-\theta_3 \leq \pi/4 \qquad \text{(constraint 4), and}$$

$$\theta_3 \leq \pi/4 \qquad \text{(constraint 5)}.$$

Let initial Θ vector be specified as $\theta_1 = \pi$, $\theta_2 = \pi/3$, and $\theta_3 = -\pi/4$. Then,

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} \pi \\ \pi/3 \\ -\pi/4 \end{pmatrix} \leq \begin{pmatrix} \pi \\ -\pi/4 \\ \pi/2 \\ \pi/4 \\ \pi/4 \end{pmatrix}.$$

Constraint 1 is an active constraint, because it is an equality constraint. Constraint 4 is an active constraint because it is at the border. Then, we have one active equality constraint, one active inequality constraint and three inactive inequality constraints. In order to bring the constraint matrix convenient

for operations in the algorithm, we have to swap and shift the rows of matrix in a way that first q (i.e. 2) rows of k (i.e. 5) row matrix will be active constraints:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} \pi \\ \pi/3 \\ -\pi/4 \end{pmatrix} \leq \begin{pmatrix} \pi \\ \pi/4 \\ -\pi/4 \\ \pi/2 \\ \pi/4 \end{pmatrix}.$$

6.3 Linearly Constrained Nonlinear Optimization Algorithm

Variable Metric Method has been introduced by Davidon [9] for the unconstrained problem as described by Fletcher and Powell [13] (hereafter called the DFP algorithm). The DFP algorithm has been improved and the BFGS algorithm has been introduced [8, 12, 17, 28]. In his original paper, Davidon envisaged the extension of his algorithm for unconstrained minimization to the case of linear equality constraints. Goldfarb [16] extended the DFP algorithm to the problem of linear inequality constraints by utilizing the techniques described by Rosen [26] in association with his projected-gradient method by which the search direction determined from the corresponding unconstrained problem is orthogonally projected to the subspace defined by those constraints on variables. At each iteration, a number of the constraints are regarded as being active and on that set of constraints, an equality problem is solved. The algorithm we presented here is taken from [38]. It is a combination of the BFGS algorithm and Rosen's projected-gradient method.

An iterative algorithm calculates the least value of objective function $G(\theta)$ of n variables subject to equality and inequality constraints as in Equation 5.19. Because the algorithm is iterative, it requires an initial estimate of the solution θ^0 and then for $i = 0, 1, 2, \ldots$ the ith iteration replaces θ^i by θ^{i+1}, which should be a better estimate of the solution. All the calculated angles θ^i are forced to satisfy the constraints. The termination condition of algorithm is Kuhn-Tucker point. For detailed explanation about the Kuhn-Tucker point, readers

are referred to [14]. The steps of algorithm are presented as follows [38]:

1. θ^0 is an initial feasible n size joint angle set, and H_0^0 is an initially chosen n-by-n positive definite symmetric matrix which is identity matrix in our application. A is n-by-m constraint matrix where m denotes the sum of all active and inactive constraints and q of them are active at point θ^0. A_q is composed of q column vector a_i of A, and the first l columns of A_q are equality constraints $a_i : i = 1, 2, \ldots, l$. H_q^0 is computed by applying Equation 6.3 q times; $g^0 = g(\theta^0)$.

2. Given θ^i, g^i, and H_q^i, compute $H_q^i g^i$ and

$$\alpha = (A_q^T A_q)^{-1} A_q^T g^i$$

   If $H_q^T g^i = 0$ and $\alpha_j \leq 0, j = l + l, l + 2, \ldots, q$, then stop. θ^i is a Kuhn-Tucker point.

3. If the algorithm did not terminate at Step 2, either $\|H_q^i g^i\| > \max\{0, 1/2\alpha_q a_{qq}^{-1/2}\}$ or $\|H_q^i g^i\| \leq 1/2\alpha_q a_{qq}^{-1/2}$, where it is assumed that $\alpha_q a_{qq}^{1/2} \geq \alpha_i a_{ii}^{-1/2}$, $i = l + 1, \ldots, q - 1$, and a_{ii} is the ith diagonal element of $(A_q^T A_q)^{-}1$. They are all positive [16]. If the first inequality holds, go to Step 4, Else drop the qth constraint from A_q, and obtain H_{q-1}^i , from

$$H_{q-1}^i = H_q^i + \frac{P_{q-1} a_{i_q} a_{i_q}^T P_{q-1}}{a_{i_q}^T P_{q-1} a_{i_q}} \tag{6.1}$$

   where $P_q - 1 = I A_{q-1} (A_{q-1}^T A_{q-1})^{-1} A_{q-1}^T$ is a projection matrix; a_{i_q} is the qth column of A_q; and A_{q-1} is the n-by-(q1) matrix obtained by taking off the qth column from A_q.

   Assign $q - 1$ to q,and go to Step 2.

4. Compute the search direction $d^i = -H_q^i g^i$ for line search sub-algorithm, and compute

$$\lambda_j = \frac{b_j - a_j^T \theta^i}{a_j^T d^i}, \quad j = q + 1, q + 2, \ldots, k$$

$$\lambda^i = min\{\lambda_j > 0\}$$

We used and modified the routine *lnsrch* from [25] for line search to obtain the biggest possible γ^i such that $0 < \gamma^i \leq min\{1, \lambda^i\}$, and

$$\begin{cases} P(\theta^i + \gamma d^i) & \leq P(\theta^i) + \delta_1 \gamma^i (g^i)^T d^i \\ g(\theta^i + \gamma d^i)^T d^i & \leq \delta_2 (g^i)^T d^i \end{cases} \tag{6.2}$$

where δ_1 and δ_2 are positive numbers such that $0 < \delta_1 < \delta_2 < 1$ and $\delta_1 < 0.5$. Let $\theta^{i+1} = \theta^i + \gamma^i d^i$ and $g^{i+1} = g(\theta^{i+1})$. If $(g^i)^T d^i > 0$ which means the function value would not be decreased, then recompute the search direction as $d^i = -g^i$ and run the step 4 again.

5. If $\gamma^i = \lambda^i$, add to A_q the a_j corresponding to $\min\{\lambda_j\}$ in Step 4 (by swapping a_j with a_{q+1}). Then compute

$$H_{q+1}^{i+1} = H_q^i - \frac{H_q^i a_j a_j^T H_q^i}{a_j^T H_q^i a_j} \tag{6.3}$$

Assign $q + 1$ to q and $i + 1$ to i, and go to Step 2.

6. Else, set $\sigma^i = \gamma^i d^i$ and $y^i = g^{i+1} - g^i$, and update H_q^i as follows:

If $(\sigma^i)^T y^i \geq (y^i)^T H_q^i y^i$ then use the BFGS formula:

$$H_q^{i+1} = H_q^i + \frac{\left(\left(1 + \frac{(y^i)^T H_q^i y^i}{(\sigma^i)^T y^i}\right)\sigma^i(\sigma^i)^T - \sigma^i(y^i)^T H_q^i - H_q^i y^i (\sigma^i)^T\right)}{(\sigma^i)^T y^i} \tag{6.4}$$

Else use the DFP formula:

$$H_q^{i+1} = H_q^i + \frac{\sigma^i(\sigma^i)^T}{(\sigma^i)^T y^i} - \frac{H_q^i y^i (y^i)^T H_q^i}{(y^i)^T H_q^i y^i} \tag{6.5}$$

Assign $i + 1$ to i, and go to Step 2.

6.4 Discussion

We obtained the best result with $\delta_1 = 0.0001$ and $\delta_2 = 0.5$. In the optimization algorithm, choosing the initial values that decrease the function value sufficiently, is crucial. Else the algorithm can be failure. Especially, while we are changing the direction of a limb to an opposite site, initial values obtained from the previous computation may cause the algorithm to be failure. In addition, since there are usually more than one solution, initial guess affects the

generated solution. This situation takes control away from the user. As it is explained in Section 6.1, we tried to obtain solution that is desired by the user, by multiplying the initial values with scalars.

The algorithm searches for a local minimum along a direction d in line search sub-algorithm. Sometimes, $(g^i)^T d^i > 0$ is occurred which means that it cannot find local minimum. Therefore, we modified the algorithm as in step 4 and we changed the search direction.

Chapter 7

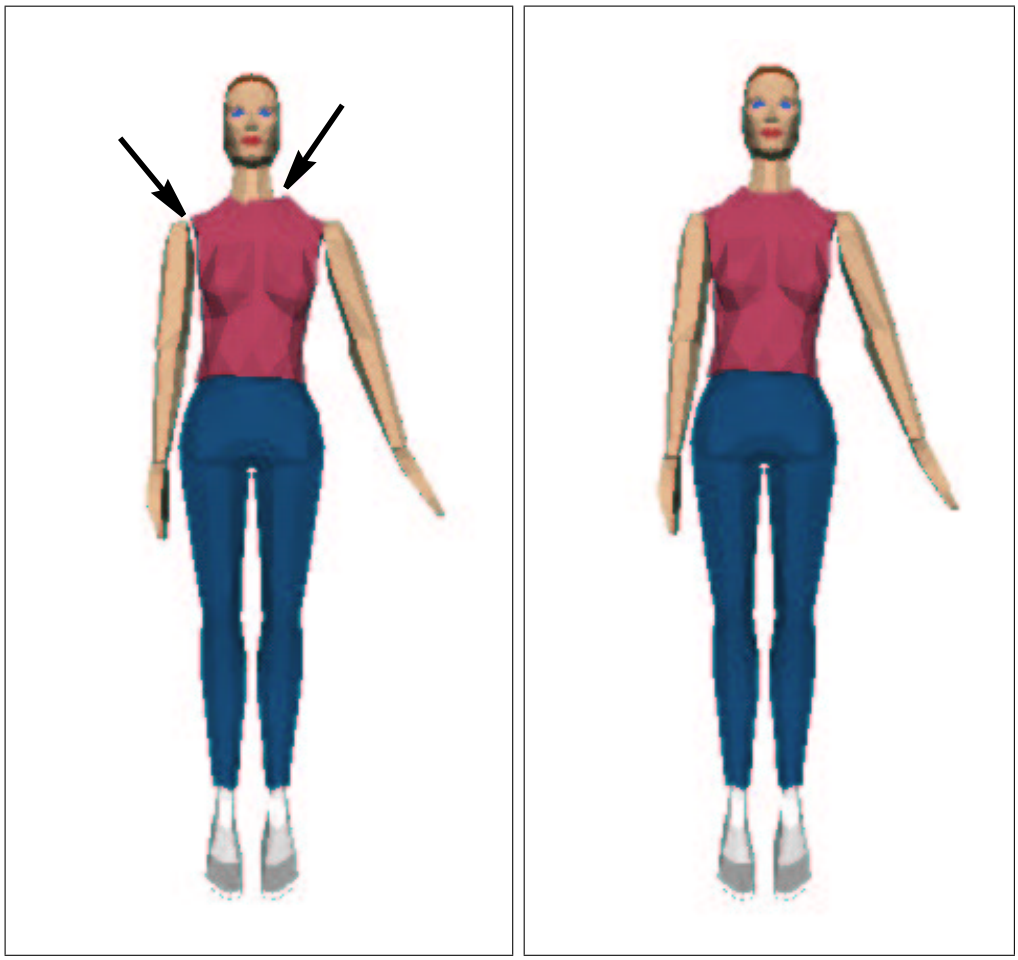
Results

Our human figure consists of 20 segments, 26 joints and 31 DOFs. Four of the joints were defined as the end effectors. These are right and left hands, and right and left feet. Four joint groups were defined. The left upper joint group includes the joints from pelvis to left hand with 9 DOFs. The right upper joint group includes the joints from pelvis to right hand with 9 DOFs. The left lower joint group includes the joints from pelvis to left foot with 8 DOFs. The right lower joint group includes the joints from pelvis to right foot with 8 DOFs.

The left and right joint groups have three common DOFs on torso. If any motion is applied to one of these joint groups, an unrealistic appearance at the segments of the other joint group may occur, since the segments are separated (see Figure 7.1.a). However, tree-structured hierarchy of the human figure enables us to pass the motion on to the other joint group since any change at the position of the parent segment effects all of the child segments (see Figure 7.1.b).

The algorithm only works for one goal instead of multiple goals at each time. Figure 7.2.a is an example of a reachable goal and Figure 7.2.b shows an unreachable goal.

As it is explained in Section 6.1, a scalar multiplier can be assigned to the joint angles. In Figure 7.3.a, an undesirable posture comes out without assigning a scalar multiplier. While the joint angles rotating around the x axis

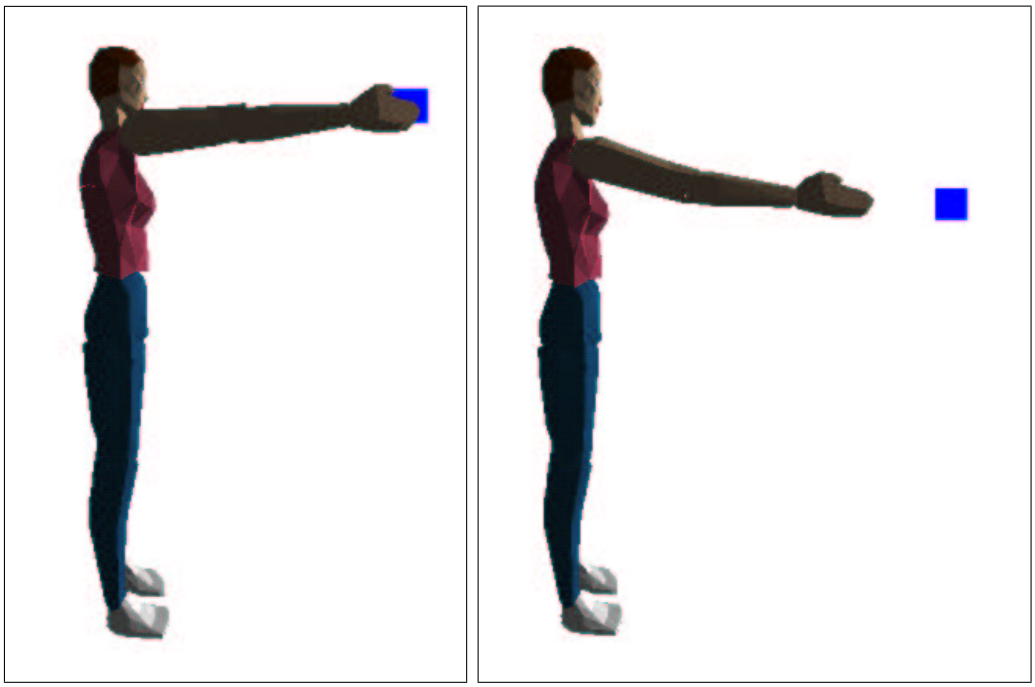


(a) (b)

Figure 7.1: (a) *unrealistic posture*; (b) *realistic posture*.

is sufficient to reach the goal, the joint angles rotating around y and z axes, have initial values that effect the direction of the line search. In order to reach the defined goal in Figure 7.3.b, the joint angles, except the angles rotating around the x axis, are multiplied with 0.1. The priority of the joint angles rotating around the x axis is increased with this way. The algorithm handles them first.

Figure 7.4 gives some example postures produced by the implementation.



(a) (b)

Figure 7.2: (a) *reachable goal*; (b) *unreachable goal*.

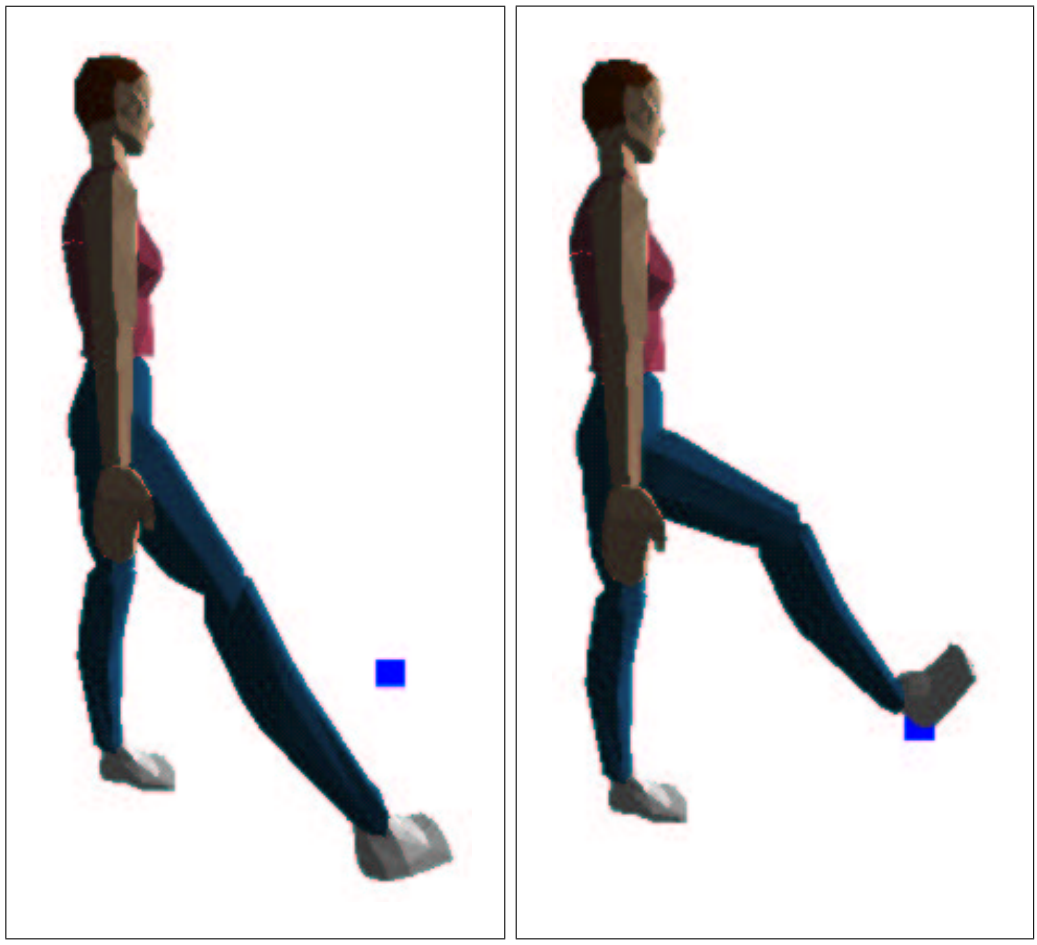
7.1 Performance Experiments

Totally, 899 vertices have been used to draw the segments. Average fps results during motion have been presented at Table 7.1 for the situations that are lights on/off and wireframe/mesh. The results were obtained on a personal computer with Intel Pentium[1] III – 550 Mhz CPU and 192 MB of main memory with 32 MB of graphics memory.

Table 7.1: Average frame rates for animations

Wireframe/Mesh	Shading	Average FPS
Wireframe	Shaded	13.90
Wireframe	Not shaded	26.65
Mesh	Shaded	14.05
Mesh	Not shaded	29.05

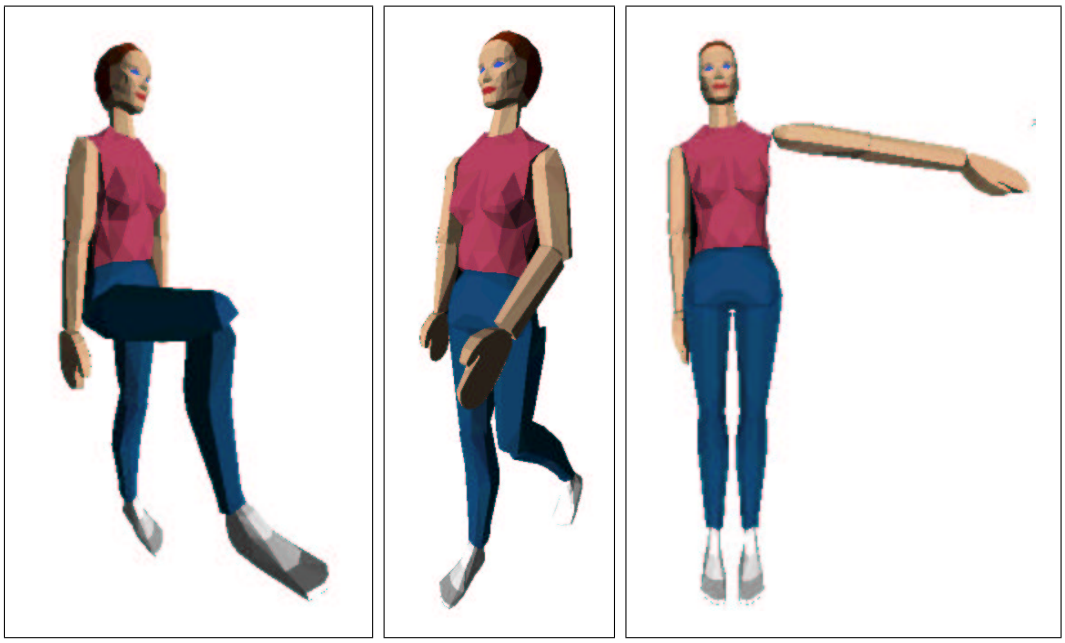
[1]Pentium is a registered trademark of Intel Corporation.



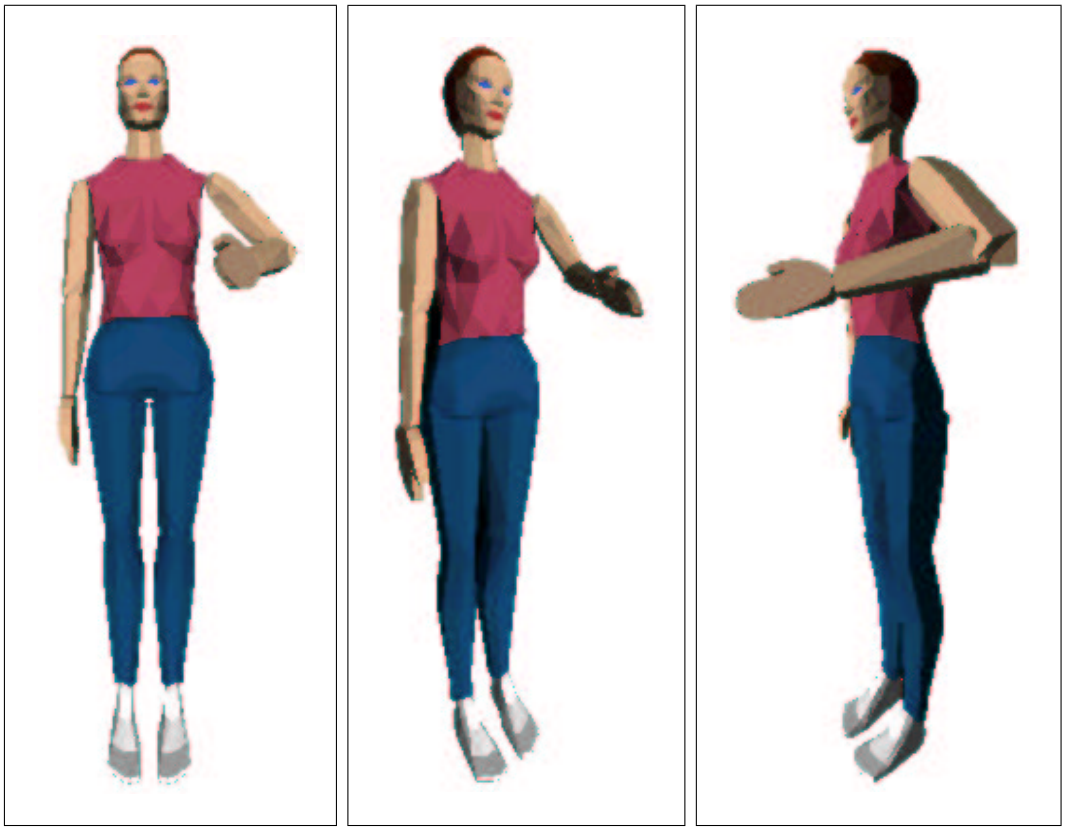
(a) (b)

Figure 7.3: (a) *without scaling factor*; (b) *with scaling factor*.

While the lights are off, the human figure is animated in real time. However, the human figure is animated near real time while the lights are on.



(a) (b) (c)



(d) (e) (f)

Figure 7.4: Example postures.

Chapter 8

Conclusion and Future Work

We implemented an algorithm for human animation with inverse kinematics using nonlinear programming. Usually, the selected end effector completed its motion towards the desired goal. However, the user has to try the same motion for multiple times with different options in order to reach the desired goal realistically. Even the nonlinear optimization algorithm can cause the undesirable motion in some situations. Since the initial angle set that comes from the previous motion, may not be the feasible angle set for the next motion. This situation prevents the line search algorithm to find a minimum value for the function. Constructing a new strategy for finding an initial feasible point for each motion is a necessity as a future work.

Besides, more other goal types such as aiming goal, line goal and plane goal etc. can be defined and be implemented. When the goal is too far away from the end effector, we partitioned the goal into subgoals. However, it may be a better approach to define the paths in order to reach the far goals.

A single goal is inadequate to define a posture. For example, while the hand reaches the goal, wrist may be at an undesirable position. In such a situation, another goal for the wrist has to be defined. For more complex postures such as sitting, multiple goal definition is necessary. Multiple goals can be handled as a future work. Badler et al [1] explained how to handle multiple constraints (goals) in detail.

The human shoulder is a complex structure, which is composed of the three body segments, the clavicle, the scapula and the arm. The shoulder complex has to be handled as a special structure to obtain more realistic motion. Maurel and Thalmann give an example to the human shoulder modelling in [23].

Body awareness is not handled in our implementation. For example, while left hand is moving, it can enter the inside of the torso segment. Body awareness can be handled as a future work.

Analytic solutions proposed in [32] offer high performance for arm and leg limbs with seven DOFs. An analytical solution for the whole body structure can be implemented and the results of the two implementations could be compared.

Bibliography

[1] N. Badler, K. Manoochehri, and G. Walters. Articulated figure positioning by multiple constraints. *IEEE Computer Graphics and Applications*, Vol. 7, No. 6, pp. 28–38, 1987.

[2] N.I. Badler, C.B. Phillips, and B.L. Webber. *Simulating Humans: Computer Graphics, Animation, and Control*. Oxford University Press, Oxford, 1999.

[3] D. Baraff. Linear-time dynamics using lagrange multipliers. In *Proc. of SIGGRAPH'96*, pp. 137–146, 1996.

[4] R. Bartels and I. Hardtke. Speed adjustment for keyframe interpolation. In *Proc. of Graphics Interface'89*, pp. 14–19, 1989.

[5] R.H. Bartels, J.C. Beatty, and B.A. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, Los Alamos, CA, 1987.

[6] R. Barzel and A.H. Barr. A modeling system based on dynamic constraints. In *Proc. of SIGGRAPH'88*, pp. 179–188, 1988.

[7] B. Bodenheimer, C. Rose, S. Rosenthal, and J. Pella. The process of motion capture: dealing with the data. In *Proc. of Eurographics Workshop on Computer Animation and Simulation*, pp. 3–18, 1997.

[8] C.G. Broyden. The convergence of a class of double-rank minimization algorithms, 2. the new algorithm. *Journal of the Institution for Mathematics and Applications*, Vol. 6, pp. 222–231, 1970.

[9] W. Davidon. Variable metric methods for minimization. AEC Research and Development Report. ANL–5990, Argonne National Laboratory., Argonne, IL, 1959.

[10] J. Denavit and R. Hartenberg. A kinematic notation for lower pair mechanisms based on matrices. *ASME Journal of Applied Mechanics*, Vol. 22, No. 6, pp. 215–221, 1955.

[11] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, San Diego, CA, 1988.

[12] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, Vol. 13, No. 3, pp. 317–322, 1970.

[13] R. Fletcher and M.J.D. Powell. A rapidly convergent descent method for minimization. *The Computer Journal*, Vol. 6, No. 2, pp. 163–168, 1963.

[14] P.E. Gill and W. Murray. *Numerical methods for constrained optimization*. Academic Press Inc., New York, Second Edition, 1978.

[15] M. Girard and A. Maciejevski. Computational modeling for the computer animation of legged figures. *In Proc. of SIGGRAPH'85*, pp. 263–270, 1985.

[16] D. Goldfarb. Extension of davidon's variable metric method to maximization under linear inequality and equality constraints. *SIAM Journal on Applied Mathematics*, Vol. 17, No. 4, pp. 739–764, 1969.

[17] D. Goldfarb. A family of variable metric methods derived by variational means. *Mathematics of Computation*, Vol. 24, No. 109, pp. 23–26, 1970.

[18] P.M. Isaacs and M.F. Cohen. Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics. In *Proc. of SIGGRAPH'87*, pp. 215–224, 1988.

[19] C. Klein and C.H. Huang. Review of pseudoinverse control for use with kinematically redundant manipulators. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 13, No. 2, pp. 245–250, 1983.

[20] D. Kochanek and R. Bartels. Interpolating splines with local tension, continuity, and bias control. In *Proc. of SIGGRAPH'84*, pp. 33–41, 1984.

[21] J. Korein. *A Geometric Investigation of Reach*. Ph.D. Thesis, University of Pennsylvania, Department of Computer and Information Science, 1985.

[22] S. K. Mahmud. Animation of human motion: An interactive tool. MS. Thesis, Bilkent University, Department of Computer Engineering and Information Science, 1991.

[23] W. Maurel and D. Thalmann. Human shoulder modeling including scapulo-thoracic constraint and joint sinus cones. *Computers and Graphics*, Vol. 24, No. 2, pp. 203–218, 2000.

[24] T. Molet, R. Boulic, and D. Thalmann. A real time anatomical converter for human motion capture. In *Proc. of Eurographics Workshop on Computer Animation and Simulation*, pp. 79–94, 1996.

[25] W. Press, P. Elannery, T. Tevkolsky, and W. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Second Edition, `http://www.nr.com`, Cambridge University Press, Cambridge, 1992.

[26] J.B. Rosen. The gradient projection method for non-linear programming. part I: linear constraints. *SIAM Journal on Applied Mathematics*, Vol. 8, No. 1, pp. 181–217, 1960.

[27] L. Sciavicco and B. Siciliano. A dynamic solution to the inverse kinematic problem of redundant manipulators. In *Proc. of the IEEE International Conference on Robotics and Automation*, pp. 1081–1086, 1987.

[28] D.F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, Vol. 24, No. 111, pp. 647–664, 1970.

[29] K. Sims and D. Zeltzer. A figure editor and gait controller for task level animation. In *SIGGRAPH Course Notes*, Course Number. 4, pp. 164–181, 1988.

[30] S.N. Steketee and N.I. Badler. Parametric keyframe interpolation incorporating kinematic adjustment and phrasing control. In *Proc. of SIGGRAPH'85*, pp. 255–262, 1985.

[31] The MathWorks, Inc. *MATLAB package*. `http://www.matworks.com`, 2001.

[32] D. Tolani, A. Goswami, and N. Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models*, Vol. 62, No. 5, pp. 353–388, 2000.

[33] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques*. Addison-Wesley Press, New York, 1992.

[34] D. Whitney. Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man-Machine Systems*, Vol. 10, No. 2, pp. 47–63, 1969.

[35] J. Wilhelms. Dynamic Experiences. In N. Badler, B. Barsky, and D. Zeltzer, editors, *Making Them Move: Mechanics, Control and Animation of Articulated Figures*, Chapter 13, pp. 265–280. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[36] A. Witkin, K. Fleischer, and A. Barr. Energy constraints on parameterized models. *In Proc. of SIGGRAPH'87*, pp. 225–232, 1987.

[37] A. Witkin and M. Kass. Spacetime constraints. In *Proc. of SIGGRAPH'88*, pp. 159–168, Atlanta, GA, 1988.

[38] J. Zhao and N. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Computer Graphics*, Vol. 13, No. 4, pp. 313–336, 1994.

[39] X. Zhao. *Kinematic Control of Human Postures for Task Simulation*. Ph.D. thesis, University of Pensylvania, Department of Computer and Information Science, 1996.

Appendix A

The User Interface

In this appendix we give information about the user interface implemented for the human animation with inverse kinematics using nonlinear programming.

A.1 Overview

In the GUI of the implementation, it is intended to give many options to the user in order to control the human figure. The graphical user interface of the implementation is given in Figure A.1. The user interface is developed to give the user the ability to navigate around the figure and the ability to control the motion of the figure. Both the navigation and motion control can be achieved by the helps of menu on the screen, keyboard and mouse.

The user interface is a combination of two parts on the screen. One of them is the viewing area for the figure. The other is the menu section.

A.2 Viewing Area

To use cursor key navigation and function keys, it is needed to click the viewing area with mouse. Likewise in order to use menu section, menu area has to be clicked.

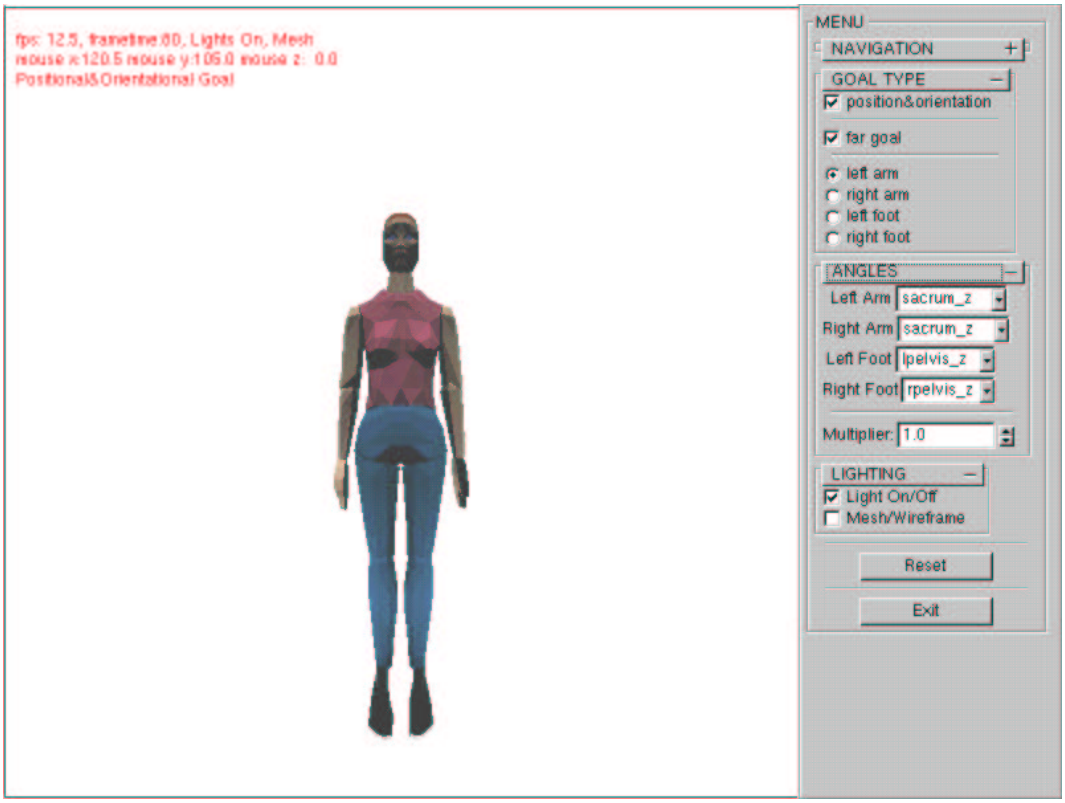


Figure A.1: Graphical user interface of the system.

Viewing area contains information about some parameters. Those parameters are

- frames per second,
- frame drawing time,
- mouse coordinates,
- wire-frame/mesh (*F2 function key*),
- lights on/off (*F3 function key*), and
- goal type (positional/positional and orientational) (*F1 function key*).

A.3 Menu Area

The menu area is a combination of collapsible/expandable menu blocks. The user may turn a block on by pressing its name button on the menu section. This makes menu area flexible.

A.3.1 Navigation Block

In the navigation block, there are two parts:

- location control part, and
- looking direction part (a *rotating ball* and a *reset button*).

Location control part is used to bring the user to a specific location and direction. There are four spinner-buttons that enable the user to achieve this purpose. These are

- *Direction*: Used to change the direction of the viewer. As the value increases the user turns to the right.
- *Altitude*: Used to increase or decrease the altitude of the user. The value shows the altitude of the viewer from the *average elevation* of the figure.
- *Z axis*: Used to change the view point of the user along z axis.
- *X axis*: Used to change the view point of the user along x axis.

Rotating ball makes it available for the user to move the figure in any direction. This is very useful to visualize the figure in any angle when the user comes to a point. *Reset Look* button is used to initialize the ball to its original position.

A.3.2 Selecting Goal Type

Four end-effector is specified in order to move the figure. These are:

- *Left Arm*: Left arm is the end-effector of joint group from pelvis to left arm.
- *Right Arm*: Right arm is the end-effector of joint group from pelvis to right arm.
- *Left Foot*: Left foot is the end-effector of joint group from pelvis to left foot.
- *Right Foot*: Right foot is the end-effector of joint group from pelvis to right foot.

An end-effector can be defined by the user selecting radio buttons in *goal type* menu.

The user has two options to move the joint group defined by an end-effector. One of them is to push the left mouse button on the end-effector and drag the mouse, consequently the end-effector to the desired goal slowly. When the end-effector reaches the goal, the user releases the left mouse button. The other option is to define a far goal by checking *far goal* check box in *goal type* menu. When the user pushes the left mouse button for the desired goal on view area, selected joint group moves and the end-effector tries to reach the goal.

The user also can define goal type by checking *position and orientation* check box. If it is checked, goal is positional and orientational goal, else goal is only positional goal.

A.3.3 Assigning Scalar Multiplier to the Joint Angles

As it is explained in Chapter 6, the user may scale the joint angles by assigning scalar multipliers. For each of the four joint group, there are names of joint angles in list boxes in *angles* menu. First, user assigns a value between 0.1 and

2.5 to multiplier by the help of the spinner in *angles* menu. And select the angle which will be scaled with the multiplier.

A.3.4 Visual Properties Block

There are two check boxes in visual properties block. One of them turns the lights on when checked and off when unchecked. The other check box shows the figure as wire-frame when checked and as mesh when unchecked.

Reset button at the end brings the figure to initial posture if body segments have been transformed. The *exit* button enables the user to terminate the program.

A.4 Keyboard and Mouse Usage for GUI

Functions of keyboard buttons are given as follows:

- *Left cursor key*: Turns the figure around the user from left.
- *Right cursor key*: Turns the figure around the user from right.
- *Up cursor key*: Enables the user come closer to the figure along z axis.
- *Down cursor key*: Takes away the user from figure along z axis.
- *'-' key*: Decreases the altitude of user.
- *'+' key*: Increases the altitude of user.
- *PageUp key*: Increases z value of mouse by 1.
- *PageDown key*: Decreases z value of mouse by 1.
- *F1 function key*: Provides switch between positional and positional and orientational goal.
- *F2 function key*: Provides switch between mesh and wireframe view of figure.

- *F3 function key*: Provides switch between lights on and off.

For positional goal, only the left mouse button is used. If *far goal* check box is checked, clicking the left mouse button at anywhere on view area defines the x and y coordinates of goal. Else if the user presses the left mouse button on any end-effector and drags the mouse, x and y coordinates of goal is defined by the x and y values of the mouse until left button is unpressed.

For positional and orientational goal, pushing left button of mouse provides rotation around x axis, pushing right button of mouse provides rotation around z axis and pushing both left and right button of mouse provides rotation around y axis. Besides, x and y coordinates of mouse while it is pressed, gives the x and y position of goal.