10-1994

Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures

Jianmin Zhao
University of Pennsylvania

Norman I. Badler
University of Pennsylvania, badler@seas.upenn.edu

Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures

Abstract
An articulated figure is often modeled as a set of rigid segments connected with joints. Its configuration can be altered by varying the joint angles. Although it is straightforward to compute figure configurations given joint angles (forward kinematics), it is more difficult to find the joint angles for a desired configuration (inverse kinematics). Since the inverse kinematics problem is of special importance to an animator wishing to set a figure to a posture satisfying a set of positioning constraints, researchers have proposed several different approaches. However, when we try to follow these approaches in an interactive animation system where the object on which LOoperate is as highly articulated as a realistic human figure, they fail in either generality or performance. So, we approach this problem through nonlinear programming techniques. It has been successfully used since 1988 in the spatial constraint system within Jack 'W,a human figure simulation system developed at the University of Pennsylvania, and proves to be satisfactorily eff]cient, controllable, and robust. A spatial constraint in our system involves two parts: one constraint on the figure, the end-eflector, and one on the spatial environment, the goal. These two parts are dealt with separately, so that wc can achieve a neat modular implementation. Constraints can be added one at a time with appropriate weights designating the importance of this constraint relative to the others and are always solved as a group. [f physical limits prevent satisfaction of all the constraints, the system stops with the (possibly local) optimal solution for the given weights. Also, the rigidity of each joint angle can be controlled, which is useful for redundant degrees of freedom.

Keywords
algorithms, performance, articulated figures, inverse kinematics, nonlinear programming

Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures

Jianmin Zhao and **Norman I. Badler**

Department of Computer and Information Science

University of Pennsylvania

Philadelphia, PA 19104-6389

Abstract

An articulated figure is often modeled as a set of rigid segments connected with joints. Its configuration can be altered by varying the joint angles. Although it is straightforward to compute figure configurations given joint angles (forward kinematics), it is not so to find the joint angles for a desired configuration (inverse kinematics). Since the inverse kinematics problem is of special importance to an animator wishing to set a figure to a posture satisfying a set of positioning constraints, researchers have proposed many approaches. But when we try to follow these approaches in an interactive animation system where the object to operate on is as highly articulated as a realistic human figure, they fail in either generality or performance, and so a new approach is fostered. Our approach is based on nonlinear programming techniques. It has been used for several years in the spatial constraint system in the *Jack*™ human figure simulation software developed at the Computer Graphics Research Lab of the University of Pennsylvania, and proves to be satisfactorily efficient, controllable, and robust. A spatial constraint in our system involves two parts: one on the figure, called the *end-effector*, and the other one on the spatial environment, called the *goal*. These two parts are dealt with separately, so that a neat modular implementation is achieved. Constraints can be added one at a time with appropriate weights designating the importance of this constraint relative to the others, and the system solves them and retains them whenever a constraint is violated because either the figure or the goal is moved. In case it is impossible to satisfy all the constraints thanks to physical limits, the system stops with the optimal solution for the given weights. In addition, the rigidity of each joint angle can be controlled, which is useful when degrees of freedom are redundant.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism – *animation*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Inverse kinematics, highly articulated figures, nonlinear programming

1 Introduction

In computer animation, an articulated figure is often modeled as a set of rigid segments connected by joints. A joint is, abstractly, a constraint on the geometric relationship between two adjacent segments. This "relationship" is expressed by a number of parameters called joint angles. With judicious selection of joints, so that, e.g., segments are connected to form a tree structure, a collection of the joint angles of all the joints corresponds one-on-one to a configuration of the figure. While this correspondence provides an immediate computer representation of articulated figure configurations in the sense that given a set of joint angles it is straightforward to compute the corresponding configuration, the problem of finding a set of joint angles that corresponds to a given configuration, the inverse kinematics problem, persists in practice.

The inverse kinematics problem, however, is extremely important in computer animation, since it is often the spatial appearance, rather than the joint angles, that an animator is interested in. Naturally, the problem has received attention of many researchers in computer animation, as well as in robotics (see the next section), but the various algorithms reflect particular aspects of the problem and fail to provide a general, efficient, and robust solution for positioning highly articulated figures in an interactive animation system.

In interactive manipulation of articulated figures, where an animator poses a figure in the spatial context whereas joint angles are merely internal (and possibly hidden) representations of postures (configurations) [18], the joint angles that define the target configuration is much more interesting than the process that the joint angles take in arriving at the target. It is the responsiveness that is essential. Quick response is also essential for practical control of articulated figures where the mapping from spatial configurations to joint angles has to be done repeatedly. For example, in path planning with strength constraints, the prediction

of the next configuration is transformed to joint angles iteratively [14]. Workspace computation is another example [1]. In the former example, the time sequence is handled by some other level of control; and in the latter example, the process that the joint angles take in arriving at target postures is not pertinent.

It is in this context that we offer a new approach to the inverse kinematics problem. In the following section, we shall talk about our motivation in more detail. Our approach is based on nonlinear programming, a numerical method for solving the minimum of a nonlinear function. It searches for the solution in the high-dimensional joint angle space based on computational economy rather than physical meanings. It deals with joint limits intrinsically rather than as a special case. It is successfully implemented and has found wide uses, as noted above.

Because of the complex nature of nonlinear functions, many efficient nonlinear programming algorithms terminate when they find local minima. The algorithm we picked has this limitation, too. In practice, however, this is not an unacceptably serious problem. Local minima are less likely when the target configuration is not too distant from the starting one. If they do occur during interactive manipulation, users can easily perturb the figure configuration slightly to get around the local minima.

2 Background

Inverse kinematics for determining mechanism motion is a common technique in mechanical engineering, particularly in robot research [16]. In robotics, however, people are mostly concerned about the functionality of manipulators; overly redundant degrees of freedom are usually not desired except for special purposes. Moreover, the computation is usually carried out on particular linkage geometries. In contrast, many interesting objects in the computer animation domain, the human figure, for example, have many redundant degrees of freedom when viewed as a tree-structured kinematic mechanism. So it was necessary to look for effective means for solving this problem under various circumstances peculiar to computer animation.

Korein and Badler began to study and implement methods for kinematic chain positioning, especially in the context of joint limits and redundant degrees of freedom [12, 13]. In [3], Badler *et al* used position constraints to specify spatial configurations of articulated figures. They recursively solved for joint angles to satisfy multiple position constraints. But, owing to their simple solver, the constraints handled were limited to the type of point-to-point position constraints only.

Girard and Maciejewski adopted a method from robotics. In [11], they calculated the pseudo-inverse of the Jacobian matrix which relates the increment of the joint angles to the displacement of the end-effector in space. The main formula is

$$\Delta\theta = J^{+}\Delta\mathbf{r}$$

where $\Delta\theta$ is the increment of the joint angle vector, $\Delta\mathbf{r}$ is the displacement of the vector representing the position and/or orientation of the end-effector in space, and J^{+} is the pseudo-inverse of the Jacobian $\partial\mathbf{r}/\partial\theta$. To understand this, we can think of $\mathbf{r}$ as a 3-D column vector denoting the position of the hand, and θ as a n-D column vector consisting of all joint angles which may contribute to the motion of the hand — e.g., all the joint angles from the shoulder to the wrist. This is a differential equality; in other words, the equality holds only if we ignore the displacement of higher order $O(|\Delta\mathbf{r}|^2)$. It was developed to drive the robot, where the increment is small because actual motion has to be carried out physically in continuous way. To simply position a human figure in a computer simulated environment, however, it would not be economical to move the end-effector $\mathbf{r}$ by "small" steps; in making a computer animation sequence, it would not be optimal either to take a step size smaller than necessary. Moreover, the pseudo-inverse calculation required for each step in this formula is normally quite expensive and they did not deal with joint limits.

Witkin *et al* used energy constraints for positioning purposes [24]. Constraints can be positional or orientational. They are satisfied if and only if the energy function is zero. The way they solved constraints is to integrate the differential equation:

$$d\theta(t)/dt = -\nabla E(\theta)$$

where θ is the parameter (e.g., joint angle) vector which defines the configuration of the system, E is the energy function of θ, and ∇ is the gradient operator. Clearly, if $\theta(t)$ is the integral with some initial condition, $E(\theta(t))$ monotonically decreases with time t, because

$$\begin{aligned}\frac{d}{dt}E(\theta(t)) &= \nabla E(\theta)\frac{d\theta}{dt} \\ &= -(\nabla E(\theta))^2.\end{aligned}$$

In the joint angle θ space,

$$E(\theta) = \text{constant}$$

defines a line, called the iso-energy line, on which the energy function E takes an identical value. For any number (energy level), there is such a line. Under this physical meaning of the energy function, Witkin *et al*'s method searches the path from the initial configuration to the target configuration which is, at any point, perpendicular to the iso-energy lines.

Instead of associating energy functions with constraints, Barzel and Barr introduced deviation functions which measure the deviation of two constrained parts [5]. They discussed a variety of constraints in [5], such as point-to-point, point-to-nail, etc., and their associated deviation functions. A segment in their system of rigid bodies is subjected to both external forces, such as the gravity, and constraint forces, which bring the deviations to zero whenever they are greater. Constraint forces are solved from a set of dynamic differential equations which requires that all deviations go to zero exponentially in a certain amount of time. It is worth noting that an approach based on physical modeling and interpretation is also used by Witkin and Welch on nonrigid bodies whose deformations are controlled by a number of parameters [25]. To apply this kind of methods to articulated figures, a joint would be considered as a point-to-point constraint and added to the system as an algebraic equation. This poses some practical problems that render such solutions inappropriate to highly articulated figures. First, it is not unusual to have several dozen joints in a highly articulated figure, which would add to the number of constraint equations substantially. Second, a joint of an articulated figure is meant to be an absolute constraint. In other words, it should not compete with any constraint that relates a point on a segment of the figure to a point in space. This competition often gives rise to numerical instability.

We notice that all those methods have a property in common: the target configuration is the result of a process from a starting one. This process bears some physical meaning. In Girard and Maciejewski's method [11], the process is determined by the end-effector path; in Witkin *et al*'s method [24], it is determined by the energy function (the path in θ space is perpendicular to the family of iso-energy lines); in Barzel and Barr's method [5] or other dynamic methods ([25]), the process is determined by the physical interpretations of each segment, and external and constraint forces exerted on it. Not only can these methods solve the constraints, but also offer a smooth process in which the constraints are satisfied in certain contexts. The achieved target configuration is, therefore, natural in the sense that it results from a process that the user is more or less able to comprehend and control. But this property is not free. If we are only concerned

about the target configuration defined by the spatial constraints, rather than the physical realization, which is true in many circumstances, physical methods could be computationally inefficient, because they add extra burdens to the original geometric problem. For example, in searching for a (local) minimum along a line, one may first choose a small step size and then compute the function value until it rises. Another way to find the solution could be like this. First locate an interval in which the minimum lies, and then use the golden ratio method, a method similar to binary search, to find the minimum. The first method shows a vivid picture of how the function changes to the minimum gradually, whereas the second method is statistically much faster.

Therefore, since a target configuration can be defined by the minimum of an energy function E (see [24]), why don't we look for the minimum directly? As for naturalness of the target configuration, we may give the user more immediate control by allowing the user to specify more constraints, if it remains affordable.

Nonlinear programming is a numerical technique to solve for (local) minima of nonlinear functions. The solution search maintains numerical efficiency and robustness; the intermediate values from the starting state to the final one could be in general fairly "irregular". There are two classes of nonlinear programming problems. One is the unconstrained nonlinear programming, where the variables are free to take any values; the other one is the constrained nonlinear programming, where the variables can only take values in a certain range. The constraints on the variables fit exactly to joint limits of articulated figures. Although the latter problem can be theoretically reduced to the former one, both unconstrained and constrained nonlinear programming problems have been studied extensively, because simple reduction may cause numerical instability.

So we propose a new approach to the inverse kinematics problem based on nonlinear programming methods. Our target application is interactive manipulation of highly articulated figures, such as human figures, where joints and joint limits must not be violated.

3 Spatial Constraints

The basic geometric entity considered here is the articulated figure. The data structure of an articulated figure we used is defined by the *Peabody* language developed at the Computer Graphics Research Lab at

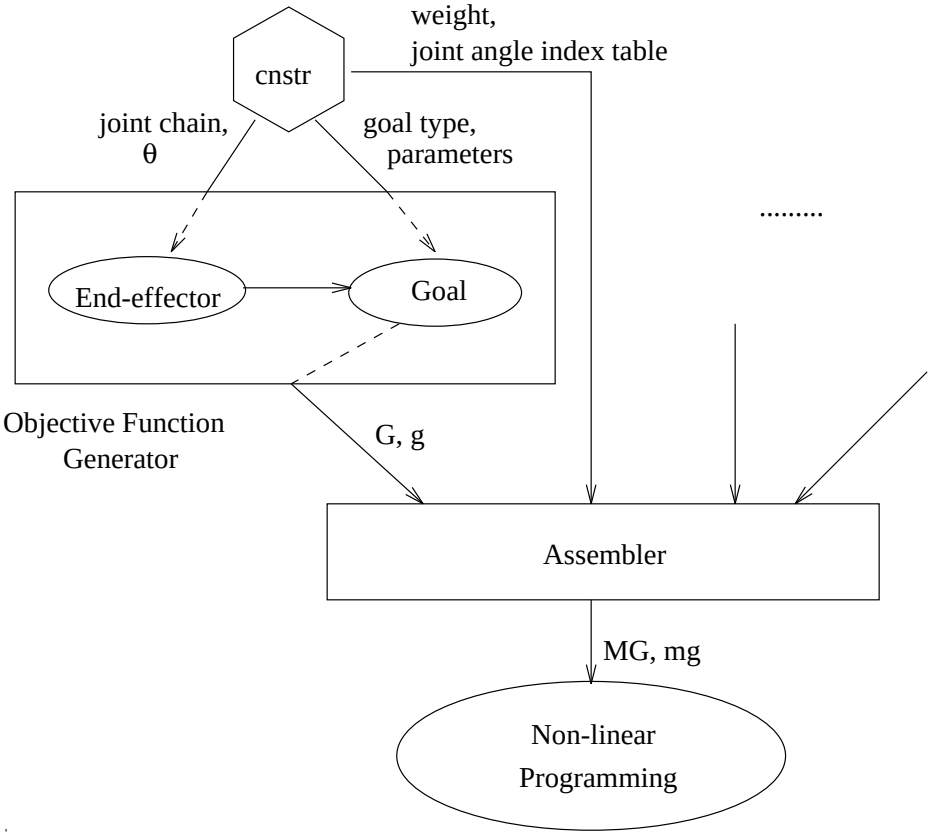

Figure 1: Multiple Spatial Constraint System

the University of Pennsylvania [17]. A *Peabody* figure is composed of rigid segments connected together by joints. Each joint has several rotational and translational degrees of freedom subject to joint limits. The data structure can be viewed as a tree, where nodes represent segments and edges represent joints.

Having decided on the data structure, we need to address the problem of setting a figure to a desired posture. As discussed in the introduction, we wish to be able to adjust the posture directly in the spatial domain. Our spatial constraints are designed for this purpose.

A spatial constraint is simply a demand that the end-effector on a segment of a figure be placed at and/or aligned with the goal in space. To say that a constraint is satisfied is equivalent to saying that the goal is reached. The end-effector's propensity to hold on to the goal persists until the constraint is disabled or deleted.

Figure 1 is a diagram of the multiple spatial constraint system in *Jack*. The system consists of three

major components: Objective Function Generator, Assembler, and Nonlinear Programming solver. They are described in the following sections.

4 End-effectors

4.1 End-effector Mappings

Formally, we can view an end-effector as a mapping:

$$\begin{cases} \mathbf{e}: & \Theta & \rightarrow & L \\ & \theta \in \Theta & \mapsto & \mathbf{e}(\theta) \in L \end{cases} \tag{1}$$

where Θ is the joint angle space, the set consisting of all joint angle vectors, and

$$L = R^3 \times S^2 \times S^2, \tag{2}$$

where R^3 denotes the set of 3-D vectors, and S^2 the set of 3-D unit vectors. Accordingly, $\mathbf{e}(\theta)$ is a 9-D vector, whose first three components form a positional vector, designating the spatial position of a point on the end-effector segment, the second and the third three components form two unit vectors, designating directions of two independent unit vectors on the end-effector segment. Given an instance of the joint angles of all the joints, θ, the end-effector $\mathbf{e}$ associates a 9-D vector $\mathbf{e}(\theta) \in L$ according to the figure definition. Since segments of a figure are rigid, the angle expanded by the last two unit vectors should remain unchanged. A convenient choice is to set it to 90 degrees. These nine numbers uniquely determine the position and orientation of the end-effector segment in space. The first three numbers are independent, but the next six numbers are not. They must satisfy two unity equations and one expanded angle equation. These three equations take away three degrees of freedom from $\mathbf{e}(\theta)$, so that $\mathbf{e}(\theta)$ has only six independent quantities, which are exactly needed to determine the position and orientation of a rigid body in space.

Let's take an example. Let the end-effector segment be the right hand, and the pelvis be fixed temporarily, serving as the root of the figure tree definition. Given joint angles of all the joints from the waist to the right wrist present in vector θ, the location and orientation of the right hand can be computed and the result is put in $\mathbf{e}(\theta)$, provided that a point and two orthonormal vectors attached on the hand have been selected for reference.

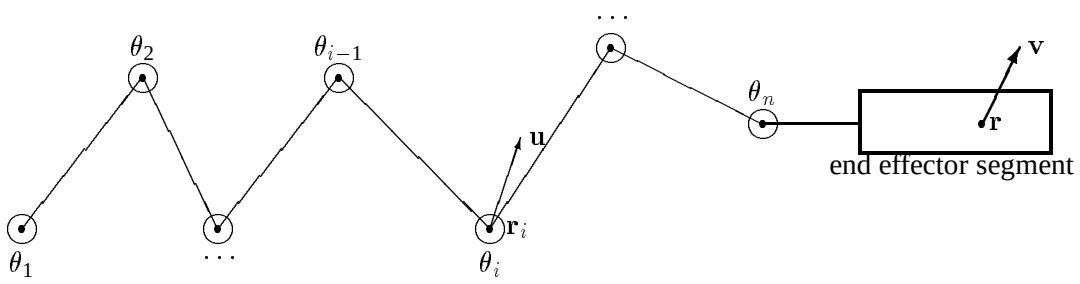

Figure 2: Joint Chain

In practice, not all components in $\mathbf{e}(\theta)$ are always interesting. Sometimes, we are only interested in the position components (the first three components), say for the location of the tip of the middle finger; sometimes, we are only interested in one of the two unit vector components in $\mathbf{e}(\theta)$, say for the direction of the (unit) vector corresponding to the index finger, or two unit vector components in $\mathbf{e}(\theta)$ which define the entire orientation of the end-effector segment. In general, we are interested in certain combinations of those cases. So, the end-effector mapping $\mathbf{e}$ is labeled with a type, and only interesting components are present in $\mathbf{e}(\theta)$.

4.2 End-effector Computational Module

The End-effector module is a part of the Objective Function Generator (see Figure 1). Since the data structure of the figure is a tree, the end-effector depends only on those joints which lie along the path from the root of the figure tree to the distal segment or the end-effector segment. Let us call this path the joint chain (Figure 2). For simplicity, we assume each joint in Figure 2 has only one degree of freedom. A joint with multiple degrees of freedom can be decomposed conceptually into several joints with each having one degree of freedom with zero distance from one joint to another. The length of the joint chain is the total number of joints along the chain; in Figure 2, it is n.

Given $\theta = (\theta_1, \theta_2, \ldots, \theta_n)^T$, where superscript T denotes transposition operator, this module is to compute the end-effector vector $\mathbf{e}(\theta)$. For the sake of computational efficiency, the algorithm we chose to

solve the constraint requires the derivative quantities

$$\frac{\partial \mathbf{e}}{\partial \theta} = \left(\begin{array}{cccc} \frac{\partial \mathbf{e}}{\partial \theta^1} & \frac{\partial \mathbf{e}}{\partial \theta^2} & \cdots & \frac{\partial \mathbf{e}}{\partial \theta^n} \end{array} \right) .$$

The matrix $\partial \mathbf{e}/\partial \theta$ is the Jacobian matrix. Its use will be explained later. Naturally, it is this module's responsibility to compute it.

The vector $\mathbf{e}(\theta)$ is composed of some combination of a point vector and two unit vectors on the end-effector segment. Referring to Figure 2, let $\mathbf{r}$ be a point vector and $\mathbf{v}$ be a unit vector on the end-effector segment. It is clear that in order to compute $\mathbf{e}(\theta)$ and $\partial \mathbf{e}/\partial \theta$, it is sufficient to know how to compute $\mathbf{r}(\theta)$, $\mathbf{v}(\theta)$, $\partial \mathbf{r}/\partial \theta$, and $\partial \mathbf{v}/\partial \theta$.

Because all the joints in our current human figure model are rotational joints, we discuss only rotational joints here. [1] Let the ith joint angle along the chain be θ^i, and the rotation axis of this joint be unit vector $\mathbf{u}$. It turns out that $\mathbf{r}(\theta)$ and $\mathbf{v}(\theta)$ can be easily computed with cascaded multiplications of 4 by 4 homogeneous matrices. The derivatives can be easily computed, too (see [26]):

$$\frac{\partial \mathbf{r}}{\partial \theta^i} = \mathbf{u} \times (\mathbf{r} - \mathbf{r}_i) \tag{3}$$

$$\frac{\partial \mathbf{v}}{\partial \theta^i} = \mathbf{u} \times \mathbf{v} \ . \tag{4}$$

5 Goals

5.1 Goal Potential Functions

A goal can also be viewed as a mapping:

$$\left\{ \begin{array}{rcl} P: \quad L & \rightarrow & R^+ \\ \mathbf{x} \in L & \mapsto & P(\mathbf{x}) \in R^+ \end{array} \right. \tag{5}$$

where the domain L is the same as the range of the end-effector mapping defined in (2), and R^+ is the set of non-negative real numbers. Since the function P assigns a scalar to a combination of position and directions in space, we call it a potential function. When the end-effector vector $\mathbf{e}(\theta)$ is plugged into the potential function P as the argument, it produces a non-negative real number, $P(\mathbf{e}(\theta))$, which is to be understood as

[1] The translational joints can be treated similarly, and actually are even simpler.

the distance from the current end-effector location (position and/or orientation) to the associated goal. For a pair of an end-effector and a goal, the range of the end-effector must be the same as the domain of the potential function.

5.2 Goal Computational Module

The Goal module is the other part of the Objective Function Generator (Figure 1). It is to compute the potential $P(\mathbf{x})$ and its gradient, the column vector formed by all its partial derivatives. Let

$$\nabla_{\mathbf{x}} = \left(\begin{array}{cccc} \frac{\partial}{\partial x^1} & \frac{\partial}{\partial x^2} & \cdots & \frac{\partial}{\partial x^n} \end{array} \right)^T$$

where x^i denotes the ith component of the vector $\mathbf{x}$. The gradient of $P(\mathbf{x})$ can thus be written as $\nabla_{\mathbf{x}} P(\mathbf{x})$.

Listed in the following are potential functions and their gradients for some useful types of goals implemented in *Jack*. Note that this module is completely independent of the data structure of the articulated figure.

Position Goals. The goal is defined by a point $\mathbf{p}$, a 3-D vector, in space. The domain L, which should be the same as the range of the corresponding end-effector mapping, is accordingly R^3.

The potential function:

$$P(\mathbf{r}) = (\mathbf{p} - \mathbf{r})^2, \tag{6}$$

and the gradient:

$$\nabla_{\mathbf{r}} P(\mathbf{r}) = 2(\mathbf{r} - \mathbf{p}) \ . \tag{7}$$

Orientation Goals. The orientation goal is defined by a pair of orthonormal vectors,

$$\{\mathbf{x}_g, \mathbf{y}_g\} \ .$$

Accordingly, the domain

$$L = S^2 \times S^2 \ .$$

Theoretically, the potential function could be:

$$P(\mathbf{x}_e, \mathbf{y}_e) = (\mathbf{x}_g - \mathbf{x}_e)^2 + (\mathbf{y}_g - \mathbf{y}_e)^2 \ .$$

In practice, however, it may not be adequate, because this potential function, when combined with a position goal, would in effect make one unit difference in length as important as about one radian difference in angle, which is not always intended. To make one length unit commensurate with d degrees in angle, we need to multiply the above P by a factor c_d such that

$$\frac{1}{c_d} = \frac{2\pi}{360} d$$

or, explicitly

$$c_d = 360/(2\pi d) \ . \tag{8}$$

To be more flexible, the potential function is chosen to be

$$P(\mathbf{x}_e, \mathbf{y}_e) = c_{dx}^2(\mathbf{x}_g - \mathbf{x}_e)^2 + c_{dy}^2(\mathbf{y}_g - \mathbf{y}_e)^2 \ . \tag{9}$$

The gradient is then

$$\nabla_{\mathbf{x}_e} P(\mathbf{x}_e, \mathbf{y}_e) = 2c_{dx}^2(\mathbf{x}_e - \mathbf{x}_g) \tag{10}$$

$$\nabla_{\mathbf{y}_e} P(\mathbf{x}_e, \mathbf{y}_e) = 2c_{dy}^2(\mathbf{y}_e - \mathbf{y}_g) \ . \tag{11}$$

A goal direction, such as $\mathbf{y}_g$, could be unconstrained by setting c_{dy} to 0. This is useful, for example, to orientationally constrain the normal to the palm of a person holding a cup of water.

Position/Orientation Goals. The position and orientation goal can be treated as two goals, but sometimes it is more convenient to combine them together as one goal. The potential function for the position/orientation goal is chosen to be a weighted sum of the position and orientation components:

$$P(\mathbf{r}, \mathbf{x}_e, \mathbf{y}_e) = w_p(\mathbf{p} - \mathbf{r})^2 + w_o c_{dx}^2(\mathbf{x}_g - \mathbf{x}_e)^2 + w_o c_{dy}^2(\mathbf{y}_g - \mathbf{y}_e)^2 \tag{12}$$

where w_p and w_o are weights assigned to position and orientation, respectively, such that

$$w_p + w_o = 1 \ .$$

The domain

$$L = R^3 \times S^2 \times S^2 \ ,$$

and the gradients $\nabla_{\mathbf{r}} P, \nabla_{\mathbf{x}_e} P$, and $\nabla_{\mathbf{y}_e} P$ can be calculated from (7), (10), and (11) above.

Aiming-at Goals. The goal is defined by a point $\mathbf{p}$ in space; the end-effector is defined by a position vector $\mathbf{r}$ and a unit vector $\mathbf{v}$ on the end-effector segment. The goal is reached if and only if the ray emanating from $\mathbf{r}$ in the direction $\mathbf{v}$ passes through $\mathbf{p}$. The domain of the potential function

$$L = R^3 \times S^2.$$

This type of goal is useful, for example, in posing a human figure facing toward a certain point.

The potential function

$$P(\mathbf{r}, \mathbf{v}) = c_d^2(\frac{\mathbf{p} - \mathbf{r}}{\|\mathbf{p} - \mathbf{r}\|} - \mathbf{v})^2 \tag{13}$$

where c_d is defined in (8), and the gradient is calculated as:

$$\nabla_{\mathbf{r}} P(\mathbf{r}, \mathbf{v}) = 2c_d^2(\|\mathbf{p} - \mathbf{r}\|^2\mathbf{v} - (\mathbf{p} - \mathbf{r})\cdot\mathbf{v}\,(\mathbf{p} - \mathbf{r}))/\|\mathbf{p} - \mathbf{r}\|^3 \tag{14}$$

$$\nabla_{\mathbf{v}} P(\mathbf{r}, \mathbf{v}) = -2c_d^2(\frac{\mathbf{p} - \mathbf{r}}{\|\mathbf{p} - \mathbf{r}\|} - \mathbf{v}) \ . \tag{15}$$

Line Goals. The goal is defined by a line which passes through points $\mathbf{p}$ and $\mathbf{p} + \nu$, where ν is a unit vector. This line is meant for a point $\mathbf{r}$ on the end-effector segment to lie on.

The potential function

$$P(\mathbf{r}) = ((\mathbf{p} - \mathbf{r}) - (\mathbf{p} - \mathbf{r})\cdot\nu\,\nu)^2 \ ; \tag{16}$$

its domain

$$L = R^3,$$

and gradient

$$\nabla_{\mathbf{r}} P(\mathbf{r}) = 2(\nu\cdot(\mathbf{p} - \mathbf{r})\,\nu - (\mathbf{p} - \mathbf{r})) \ . \tag{17}$$

Plane Goals. The goal is defined by a plane with the unit normal ν to and a point $\mathbf{p}$ on it. Similar to the Line Goal, the plane is meant for a point $\mathbf{r}$ on the end-effector segment to lie on.

The potential function

$$P(\mathbf{r}) = ((\mathbf{p} - \mathbf{r})\cdot\nu)^2 \ ; \tag{18}$$

its domain

$$L = R^3 \ ,$$

and the gradient

$$\nabla_{\mathbf{r}} P(\mathbf{r}) = -2\nu \cdot (\mathbf{p} - \mathbf{r})\, \nu \,. \tag{19}$$

Half-space Goals. The goal is defined by a plane specified the same way as in the Plane Goal. The plane is used to divide the space into two halves. A point $\mathbf{r}$ on the end-effector segment "reaches" the goal if and only if it is in the same half-space as the point $\mathbf{p} + \nu$ is.

The potential function

$$P(\mathbf{r}) = \begin{cases} 0 & \text{if } (\mathbf{p} - \mathbf{r}) \cdot \nu < 0 \\ ((\mathbf{p} - \mathbf{r}) \cdot \nu)^2 & \text{otherwise} \end{cases} ; \tag{20}$$

its domain

$$L = R^3 \,,$$

and the gradient

$$\nabla_{\mathbf{r}} P(\mathbf{r}) = \begin{cases} 0 & \text{if } (\mathbf{p} - \mathbf{r}) \cdot \nu < 0 \\ -2\nu \cdot (\mathbf{p} - \mathbf{r})\, \nu & \text{otherwise.} \end{cases} \tag{21}$$

6 Spatial Constraint as a Nonlinear Programming Problem

A spatial constraint constrains an end-effector to a goal. From Section 4 and 5, with the current joint angles being θ, the "distance" from the end-effector to the goal is simply

$$G(\theta) = P(\mathbf{e}(\theta)). \tag{22}$$

This quantity can be computed by first invoking the end-effector module to compute $\mathbf{e}(\theta)$, and then invoking the goal module with $\mathbf{e}(\theta)$ as the input argument of the potential function. This process is illustrated in Figure 1. Ideally, we want to solve the algebraic equation,

$$G(\theta) = 0.$$

In reality, however, this equation is not always satisfiable, for the goal is not always reachable. Thus the problem would be naturally to find θ in a feasible region that minimizes the function $G(\theta)$. Most of the joint angles in our figure definition have lower limits and upper limits. The joint angles for the shoulder

are confined in a polygon. They can all be expressed in linear inequalities. Therefore, we formulate the problem as a problem of nonlinear programming subject to linear constraints on variables, that is, formally,

$$\begin{cases} \text{minimize} & G(\theta) \\ \text{subject to} & \mathbf{a}_i^T\theta = b_i, i = 1, 2, \ldots, l \\ & \mathbf{a}_i^T\theta \leq b_i, i = l+1, l+2, \ldots, k \end{cases} \tag{23}$$

where $\mathbf{a}_i, i = 1, 2, \ldots, k$ are column vectors whose dimensions are the same as that of θ's. The equalities allow for linear relationships among the joint angles, and the inequalities admit of the lower limit l^i and upper limit u^i on θ^i, the ith joint angle, as do the inequalities

$$\begin{aligned} -\theta^i &\leq -l^i \\ \theta^i &\leq u^i \, . \end{aligned}$$

The polygonal region for the shoulder joint angles (elevation, abduction, and twist) can be similarly expressed as a set of inequalities.

7 Solving the Nonlinear Programming Problem

The problem posed in (23) to find the minimum of the objective function $G(\theta)$ is intractable without knowledge of the regularity of the objective function. Properties such as linearity or convexity that regulate the global behavior of a function may help to find the global minimum. Otherwise, research in nonlinear programming area is mostly done to solve for local minima. It is worthwhile because, in practice, functions are moderate: the local minimum is often what one wants, or if it fails to be, some other local minimum found by another attempt with a new initial point would quite likely be.

In order to have quick response, we chose to compromise for local minima. From years of observation, we have not seen many serious problems. The algorithm we used to solve the problem (23) is described in the Appendix. It iterates to approach the solution. At each iteration, it searches for a minimum along a certain direction. In order for the search direction to point to the solution more accurately so that fewer iterations will be needed, the direction is determined based on not only the gradient at the current point, but also the gradients at the previous steps of iteration.

Our method is monotonic, namely that after any iterations the value that the objective function takes never increases, and globally convergent, namely that it converges to a (local) minimum regardless of the initial point. These two properties are very attractive to us because the configuration could otherwise diverge arbitrarily, which could cause disaster had the previous posture resulted from substantial effort.

To carry out the computation, we need to compute $G(\theta)$ and its gradient $\nabla_\theta G(\theta)$. It becomes easy now after preparation in Sections 4 and 5. The function value can be computed as in (22), and the gradient can be computed as follows:

$$\begin{aligned} \mathbf{g}(\theta) &\stackrel{\text{def}}{=} \nabla_\theta G \\ &= (\frac{\partial \mathbf{e}}{\partial \theta})^T \nabla_{\mathbf{x}} P(\mathbf{e}) \,. \end{aligned} \tag{24}$$

where $\partial \mathbf{e}/\partial \theta$ and $\nabla_{\mathbf{x}} P(\mathbf{e})$ are readily computed by the end-effector and the goal modules, respectively. It is clear that, as the number of joint angles along the chain n grows, the computational complexity of G and $\mathbf{g}$ is linear for the goals listed in Section 5.2, since the end-effector module needs $O(n)$ time and the goal module needs $O(1)$ time.

Now we are ready to solve a single constraint. Referring to Figure 1, the objective function G and its gradient g are computed by the Objective Function Generator at the request of the module Nonlinear Programming at each iteration. To solve for multiple constraints, we only have to add up all the objective functions and their gradients, and pass the sums to the Nonlinear Programming module. This is explained in the following sections.

8 Multiple Constraints

A single constraint is far from adequate in defining a posture. Unlike methods given in [11, 24, 5] where the constraint is satisfied as a result of evolution from the initial configuration, this method strides toward the solution over the configuration space. Among the infinite number of possibilities due to high redundancy across multiple degrees of freedom, no attempt is made to assure that the solution is a "natural evolution" from the starting configuration. For example, in constraining the hand to the goal, the elbow might result in an undesired position. An additional constraint for the elbow could be necessary for a satisfactory posture. More constraints would be needed for more complex postures.

Therefore, our system handles multiple constraints. Since the objective function $G(\theta)$ defined in (22) is non-negative, the multiple constraints are solved by minimizing the sum of the objective functions associated with all the goals

$$G^{\text{all}}(\theta) = \sum_{i=1}^{m} w_i G_i(\theta) \tag{25}$$

where m is the number of constraints, subscript i denotes the association with the ith constraint, w_i is a non-negative weight assigned to the ith constraint to reflect the relative importance of the constraint, and

$$G_i(\theta) = P_i(\mathbf{e}_i(\theta)). \tag{26}$$

Thus, the multiple constraints can be solved as the problem (23) with G replaced by G^{all} defined in (25).

Note that $G_i(\theta)$'s can be computed independently, and only a number of additions are needed to compute $G^{\text{all}}(\theta)$. This is also true for the gradient, for the gradient operator ∇_θ is additive, too.

Constraints may also be tied together disjunctively, that is, they are considered satisfied if any one of them is satisfied. To solve this problem, we define the objective function as

$$G^{\text{all}}(\theta) = \min_{i \in \{1,\ldots m\}} \{G_i(\theta)\}. \tag{27}$$

It is useful, for example, to constrain an end-effector outside a convex polyhedron, because the outside space can be viewed as the disjunction of the outward half-spaces defined by the polygonal faces.

9 Assembler of Multiple Constraints

As stated in the previous sections, the overall objective function of multiple constraints can be found by computing separately and independently the objective functions of individual constraints and then adding them together. In this section, we shall explain how the Assembler works.

The module Objective Function Generator takes a joint chain, an array of corresponding joint angles, goal type, and other parameters of a constraint as its input and computes the objective function value G and its gradient g. Since the partial derivatives with respect to the joint angles other than those on the joint chain are zero, the gradient determined by this module has to include only the derivatives with respect to the joint angles on the chain. This property lends itself to a clean modular implementation. However, two gradient

vectors so structured for different constraints do not add directly — the ith joint angle in one chain may not be the same as the ith joint angle in another chain. The difference is resolved by the Assembler module.

Suppose there are m constraints. Let Θ_i be the ordered set of joint angles on the joint chain of the ith constraint, and n_i be the number of joint angles in Θ_i. Let

$$\Theta = \cup_{i=1}^{m} \Theta_i, \tag{28}$$

the union of all Θ_i's with the order defined in certain way, and n be the number of joint angles in Θ. In general, $n \leq \sum_{i=1}^{m} n_i$, because of possible overlap among Θ_i's. Let's define the index table as a mapping

$$\begin{cases} M_i : & \{1, 2, \ldots, n_i\} & \rightarrow & \{1, 2, \ldots, n\} \\ & j & \mapsto & M_i(j) \end{cases} \tag{29}$$

such that the jth joint angle in Θ_i corresponds to the $M_i(j)$th joint angle in the overall index system Θ. This index table, along with the weight of the constraint, are passed to the Assembler so that the effect of the ith constraint to the gradient of the overall objective function G^{all} can be correctly accounted. Once the g_i^j's, the derivative of the objective function of the ith constraint G_i with regard to the jth joint angle in Θ_i, are available, the Assembler does:

For $i = 1$ to m do

$g^{M_i(j)} \leftarrow g^{M_i(j)} + w_i g_i^j$, for $j = 1, 2, \ldots, n_i$,

where g^j stands for the partial derivative of G^{all} with regard to the jth joint angle in Θ. They are initially set to zero.

10 Reconciliation of Joint Chains

It was suggested in Expression (28) that only a union was needed to combine all the joint chains. In fact, it is slightly more complicated, because we allow the user to specify the set of joints in the joint chain as the resource for the constraint satisfaction. The joint chain does not have to go from the end-effector segment back to the root in the figure definition, and is specified by the user when he or she defines the constraint. Since the constraints may be input one by one, a joint which may affect the end-effector of one constraint but is not picked for the joint chain could well be picked for the joint chain of another constraint. For

example, the waist joint might not be selected for the constraint on the right hand, but later may be selected for the constraint on the left hand. Similar observations were made in [2].

So, some reconciliation is necessary to unite Θ_i's into Θ. It is done by possible extension of a joint chain. For instance, if a joint in Θ_A but not in Θ_B affects the end-effector of constraint B, it will be added to Θ_B. When a constraint is added to or deleted from the system, the reconciliation must be redone. However, by careful deliberation, this operation does not have to be done from scratch.

11 Rigidities of Individual Degrees of Freedom

The nonlinear programming algorithm we use utilizes gradient quantities. Given the same error tolerance for termination, a variable would undergo more displacement if the value of the objective function changes relatively more due to the unit increment of that variable (partial derivative). This property can be used to control the rigidity of individual degrees of freedom by assigning scaling factors to the joint angles. The scaling factor in effect changes the unit of the joint angle, and hence scales the derivative with respect to it. The greater a partial derivative is compared to the others, the closer the search direction is to the direction of the corresponding variable.

12 Implementation

A multiple spatial constraint system, in which the rigidity of individual degrees of freedom can be controlled, has been implemented in *Jack* [17]. The kernel algorithm used to solve the nonlinear programming problem is presented in the Appendix. A constraint may be of any type, or a set of disjunctively combined constraints of any type, listed in Section 5.2. The system sets up an avenue from spatial specifications of articulated figure configurations to joint angle resolutions.

The pose displayed in the left panel of Figure 3 is achieved by using 6 constraints. Two position/orientation constraints on the two hands were used to hold the tube, where one direction of the orientation component is suppressed so that only the normals to the palms and the tube are aligned. Two plane constraints on the elbows were used to stretch the elbows on two side planes. To have the figure look down toward the bottom of the tube, we used two more constraints – a line constraint to constrain the view

point (the point at the middle of the two eyes) to the central axis of the tube, and an aiming-at constraint to point the viewing vector toward the bottom of the tube. The torso of the figure has 17 segments. Between two consecutive segments is a joint of 3 degrees of freedom. These joints, however, are grouped together to form a *joint group* which is driven by three independent parameters: the forward extension, lateral bending, and axial rotation. So the number of effective degrees of freedom of the torso is just 3. The joint connecting the sternum to the clavicle and the joint connecting the clavicle to the upper arm are similarly grouped to form the shoulder complex, which has three effective degrees of freedom : elevation, abduction, and twist. The new modeling construct *joint group* for interdependencies among joints and its incorporation into the spatial constraint system are described in [4, 27]. The other degrees of freedom in this case are: two at the two elbows, six at the two wrists, three from the torso to the neck, and two from the neck to the head. The total number of degrees of freedom are thus 22. Running on a Silicon Graphics workstation 340 VGX, and starting from an upright neutral position, the solution took about 5 seconds.

As a comparison, the right panel of Figure 3 demonstrates the result of exactly the same task as above except that the spinal joints are not grouped in this task. There are 17 spinal joints, of which each has 3 degrees of freedom. So it involves 48 ($17 \times 3 - 3$) more degrees of freedom than the previous task, or 70 degrees of freedom in total. As expected, it took about 10 seconds longer. The joint angle distribution along the spinal joints is interesting. Figure 4 shows the joint angle distribution along the (grouped) spinal joints corresponding to the pose in the left panel of Figure 3. As mentioned, the spine consists of 17 vertebral segments, 12 thoracic vertebrae and 5 lumbar vertebrae. They are numbered from the top with the 1^{st} lumbar vertebra succeeding the 12^{th} thoracic vertebra. In Figure 4, T1 denotes the joint connecting the 2^{nd} thoracic vertebra to the 1^{st} thoracic vertebra, and so on. Note that T12 denotes the joint connecting the 1^{st} lumbar vertebra to the 12^{th} thoracic vertebra and L5 denotes the joint connecting the lower torso (the sacrum) to the 5^{th} lumbar. Figure 5 shows the joint angle distribution corresponding to the right panel of Figure 3 (with independent spinal joints). In comparing Figure 4 and 5, it is clear that the inverse kinematics algorithm based on function optimization methods alone results in irregular joint angle distribution along the spinal joints if they are treated independently, but it can be regulated by appropriately grouping these joints together. Figure 4 gives an example of regulated distribution. It is beyond the scope of this paper to find realistic spinal joint angle distributions. Issues regarding the kinematic model of the human spine can

be found in [15].

The human models in Figures 6 to 8 are older *Jack* bodies having just five segments for the torso. Interdependencies among the joints were not modeled. Figure 6 is a situation where the goal is not reachable if the joint chain includes only the joints from the shoulder to the hand. The goal is the position plus the direction of the normal to the right face of the box. If we add more joints to this task, so that the joint chain starts at the waist, the goal becomes reachable but all joint angles along the torso segments are treated equally. This leads to an awkward pose, as shown in Figure 7. To make it more natural, we set the rigidities in lateral bending and axial rotation of the torso segments to 0.5 (middle of the range [0,1]). The result is shown in Figure 8. The task, with joint chain starting at the waist, involves 22 degrees of freedom. It took about 2 seconds on a Silicon Graphics workstation 4D-25TG.

The task in the left panel of Figure 3 and Figure 8 involve equal number of degrees of freedom; it is worth explaining why the task in Figure 3 took about twice as long as the task in Figure 8, despite the clear superiority in speed that the Silicon Graphics 340 VGX enjoys to the 4D-25TG. The exact number of computational steps for each iteration (computational complexity) can be counted; it is $O(n^2)$ if the total number of degrees of freedom is n and the number of constraints is $O(n)$ (see Appendix). However, the algorithm is iterative and the number of iterations depends in part on the spatial complexity: the spatial relationship between the starting configuration to the target configuration, and the nonlinearity of the objective function, which can be affected by the functions used to produce joint angles of the grouped joints from a number of parameters. (It is impossible to count exactly the total number of iterations with a given tolerance; the "computational complexity" in this dimension is usually measured with convergence rate.) Obviously, the task in Figure 3 is much more complicated than that in Figure 8. We have not quantitatively analyzed the time efficiency of this algorithm. To do this, one should take into account the number of degrees of freedom involved, the number of constraints solved, and the "spatial complexity" of the target. (In analyzing time efficiency of nonlinear programming algorithms, the algorithms are often tested on some typical and reasonably involved functions. Consensus on a test suite of typical challenging inverse kinematics tasks has yet to be developed.)

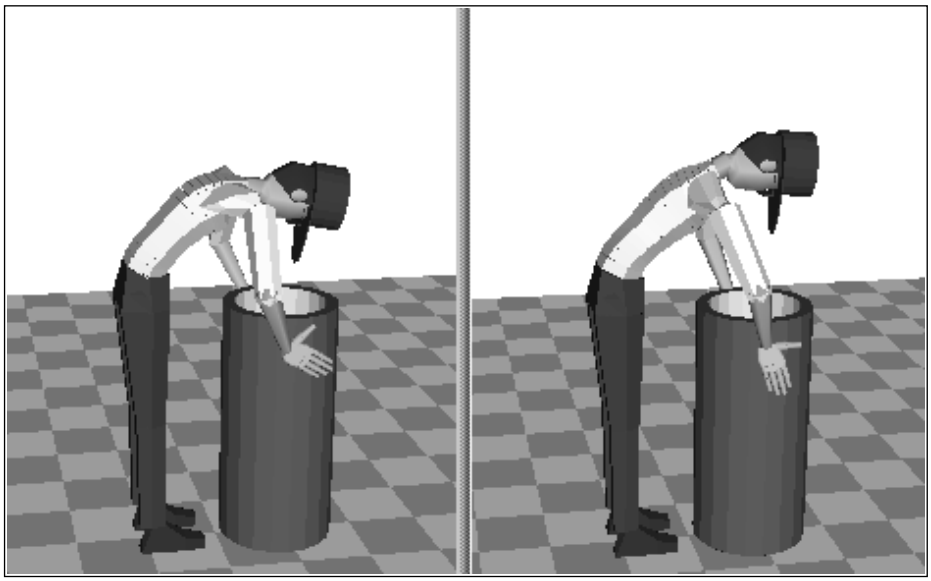

Figure 3: Looking down towards the end of the tube (Left panel: spinal joints are grouped. Right panel: spinal joints are independent)

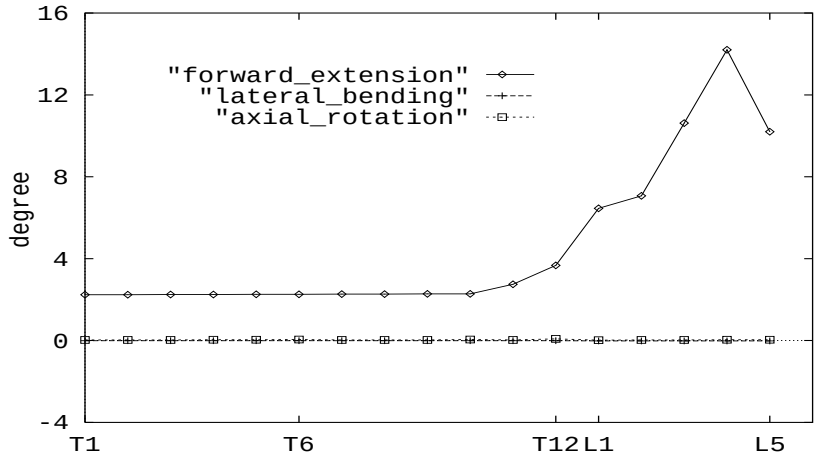

Figure 4: Joint angles distribution along the grouped spinal joints

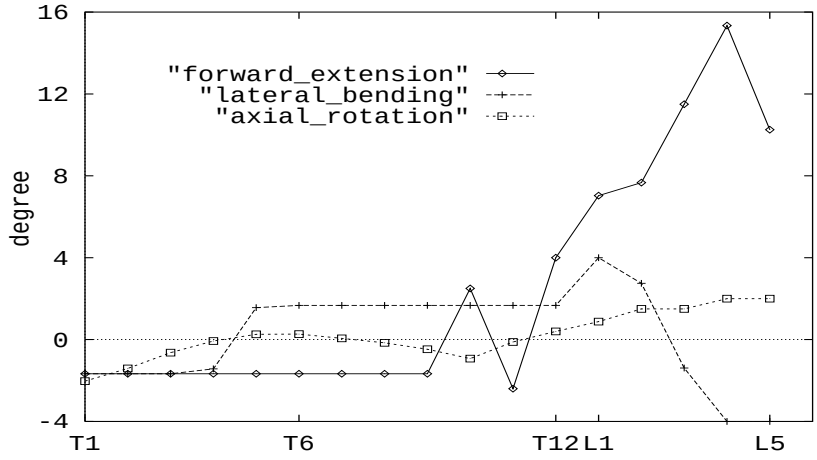

Figure 5: Joint angles distribution along the independent spinal joints

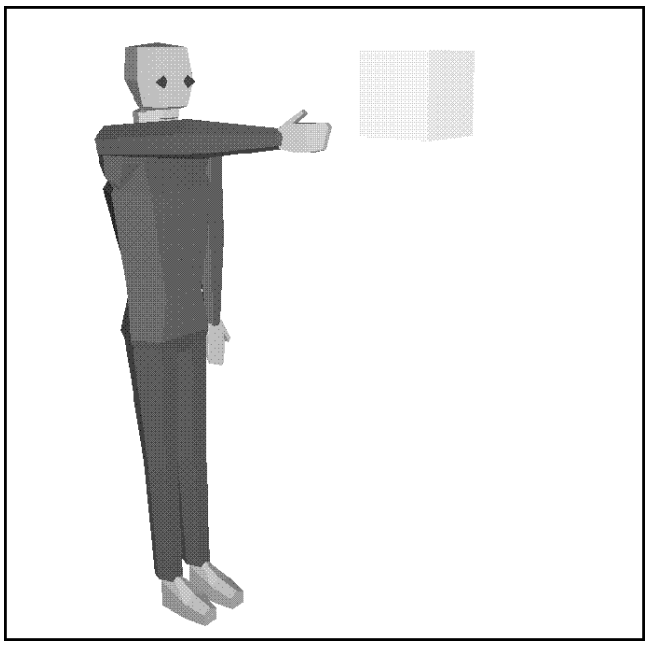

Figure 6: Goal not reachable without activating the torso

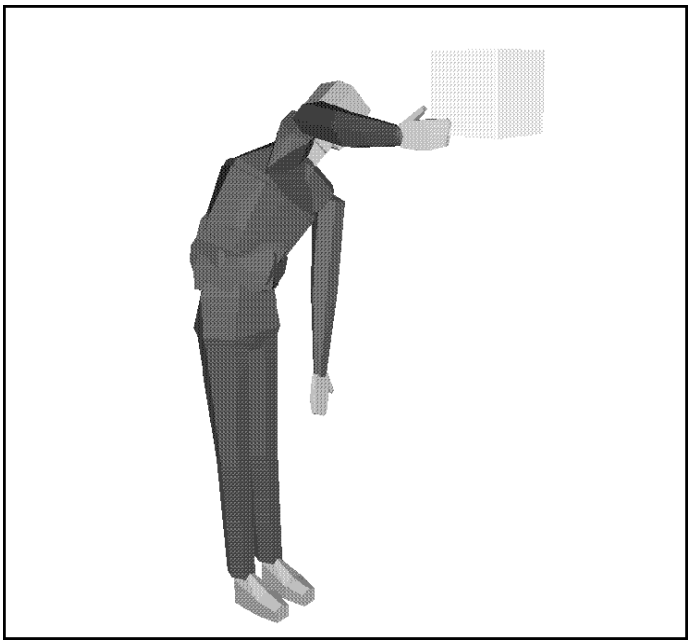

Figure 7: Successful but awkward reach

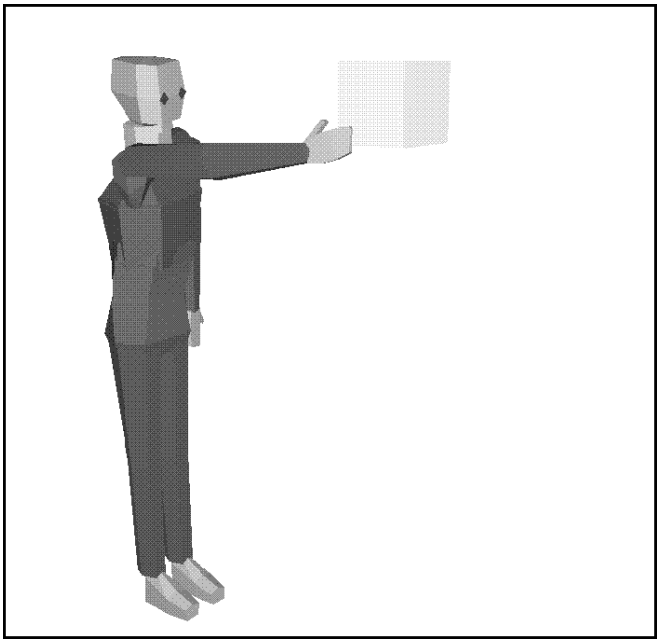

Figure 8: Natural reach with controlled rigidities of the spinal joints

13 Acknowledgments

This research is partially supported by ARO Grant DAAL03-89-C-0031 including participation by the U.S. Army Human Engineering Laboratory, Natick Laboratory, and NASA Ames Research Center; U.S. Air Force DEPTH contract through Hughes Missile Systems F33615-91-C-0001; MOCO Inc.; and NSF CISE Grant CDA88-22719. (Partial support for this work was provided by the National Science Foundation's Instrumentation and Laboratory Improvement Program through Grant #USE-9152503.)

Appendix

A Algorithm for Nonlinear Programming Subject to Linear Constraints

From Sections 4.2-5.2 and Section 9, we can compute $G(\theta)$ and $g(\theta) = \nabla_\theta G(\theta)$ in $O(nm)$ steps, where m is the number of constraints and n the total number of degrees of freedom. Many algorithms have been developed to solve the problem (23). Without constraints on joint angles, the variable metric method (or conjugate gradient method) is considered a good one. To deal with linear equality and inequality constraints in (23), Rosen proposed the projection method, by which the search direction determined from corresponding unconstrained problem is orthogonally projected to the subspace defined by those constraints on variables [22]. Goldfarb combined DFP's method (a variable metric algorithm) [7] with Rosen's projection method in [9]. But after that, the variable metric method enjoyed further improvements. The BFGS method [6, 8, 10, 23] has been considered most successful. One of the motivations of the improvement is to get the best conditioning of the approximate inverse Hessian matrix [23]. The algorithm we are presenting here is the combination of the BFGS method and Rosen's projection method. The overall framework is similar to Goldfarb's method. We just give an algorithmic description. For the full rationale, [9, 8, 10, 23] should be consulted. The algorithm, like others, only finds a Kuhn-Tucker point, that is, a point at which the objective function satisfies necessary conditions of a constrained local minimum.

Without loss of generality, we assume that all the $\mathbf{a}_i$'s in (23) are unit vectors. We say that point θ is feasible if it satisfies all the equalities and inequalities in (23). The ith constraint is said to be active at θ if $\mathbf{a}_i^T\theta = b_i$. So an equality constraint is always active at a feasible point. We assume further that at any point,

the $\mathbf{a}_i$'s for active constraints are linearly independent. Let A_q denote an n by q matrix derived by lumping together q vectors from $\mathbf{a}_i$'s, i.e. ,

$$A_q = (\ \mathbf{a}_{i_1} \quad \mathbf{a}_{i_2} \quad \cdots \quad \mathbf{a}_{i_q}\ ) \ .$$

In the following description of the algorithm, the superscript i denotes the association with the ith iteration.

Step 0. Let θ^0 be an initial feasible point, and H_0^0 an initially chosen n by n positive definite symmetric matrix. Suppose there are q constraints active at point θ^0. A_q is composed of these $\mathbf{a}_i$'s and first l columns of A_q are $\{\mathbf{a}_i : i = 1, 2, \ldots, l\}$. H_q^0 is computed by applying (32) q times. $g^0 = g(\theta^0)$.

Step 1. 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 $\alpha_j \leq 0, j = l+1, l+2, \ldots, q$, then stop. θ^i is a Kuhn-Tucker point.

Step 2. If the algorithm did not terminate at Step 1, either $\|H_q^i g^i\| > \max\{0, \frac{1}{2}\alpha_q a_{qq}^{-1/2}\}$ or $\|H_q^i g^i\| \leq \frac{1}{2}\alpha_q a_{qq}^{-1/2}$, 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, see [9])

If the former holds, proceed to Step 3.

Otherwise, drop the qth constraint from A_q and obtain H_{q-1}^i from

$$H_{q-1}^i = H_q^i + \frac{P_{q-1}\mathbf{a}_{i_q}\mathbf{a}_{i_q}^T P_{q-1}}{\mathbf{a}_{i_q}^T P_{q-1}\mathbf{a}_{i_q}} \tag{30}$$

where $P_{q-1} = I - A_{q-1}(A_{q-1}^T A_{q-1})^{-1} A_{q-1}^T$ is a projection matrix, $\mathbf{a}_{i_q}$ is the qth column of A_q, and A_{q-1} is the n by $q-1$ matrix obtained by taking off the qth column from A_q.

Let $q \leftarrow q-1$ and go to Step 1.

Step 3. Let the search direction $s^i = -H_q^i g^i$ and compute

$$\begin{aligned} \lambda_j &= \frac{b_j - \mathbf{a}_j^T \theta^i}{\mathbf{a}_j^T s^i}, j = q+1, q+2, \ldots, k \\ \lambda^i &= \min_j\{\lambda_j > 0\} \ . \end{aligned}$$

Use any line search technique to obtain the biggest possible γ^i such that $0 < \gamma^i \leq \min\{1, \lambda^i\}$, and

$$\begin{cases} P(\theta^i + \gamma^i s^i) & \leq & P(\theta^i) + \delta_1 \gamma^i (g^i)^T s^i \\ g(\theta^i + \gamma^i s^i)^T s^i & \geq & \delta_2 (g^i)^T s^i \end{cases} \tag{31}$$

where δ_1 and δ_2 are positive numbers such that $0 < \delta_1 < \delta_2 < 1$ and $\delta_1 < 0.5$. Let $\theta^{i+1} = \theta^i + \gamma^i s^i$ and $g^{i+1} = g(\theta^{i+1})$.

Step 4. If $\gamma^i = \lambda^i$, add to A_q the $\mathbf{a}_j$ corresponding to the $\min\{\lambda_j\}$ in Step 3. Then compute

$$H_{q+1}^{i+1} = H_q^i - \frac{H_q^i \mathbf{a}_j \mathbf{a}_j^T H_q^i}{\mathbf{a}_j^T H_q^i \mathbf{a}_j} \tag{32}$$

Set $q \leftarrow q + 1$ and $i \leftarrow i + 1$ and go to Step 1.

Step 5. Otherwise, set $\sigma^i = \gamma^i s^i$ 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:

$$H_q^{i+1} = H_q^i + ((1 + \frac{(y^i)^T H_q^i y^i}{(\sigma^i)^T y^i})\sigma^i (\sigma^i)^T - \sigma^i (y^i)^T H_q^i - H_q^i y^i (\sigma^i)^T)/(\sigma^i)^T y^i \tag{33}$$

else use the DFP formula:

$$H_q^{i+1} = H_q^i + \frac{\sigma^i (\sigma^i)^T}{(\sigma^i)^T y^i} - \frac{H_q^i y^i (y^i)^T H_q^i}{(y^i)^T H_q^i y^i} \tag{34}$$

Set $i \leftarrow i + 1$ and go to Step 1.

The inexact line search strategy (31) in Step 3 was proposed by Powell [20] who also suggested $\delta_1 = 0.0001$ and $\delta_2 = 0.5$. Since s^i is a descent direction, i. e. , $(g^i)^T s^i < 0$, this strategy guarantees that the function value is decreased and $(\sigma^i)^T y^i \geq (1 - \delta_2)|(g^i)^T s^i| > 0$. Because, as we pointed out in Section 7, the gradient $g(\theta)$ is almost as expensive as the function $P(\theta)$, we used a cubic Hermite interpolation method in linear searching for the sake of speed. We feel it is fairly effective.

The switch between the BFGS formula and the DFP formula in Step 5 was suggested by Fletcher [8].

Notice that all matrix multiplications are performed as an n by n matrix and a vector, or an n by 1 matrix and a 1 by n matrix. For example, matrix multiplication $H_q^i \mathbf{a}_j \mathbf{a}_j^T H_q^i$ can be grouped as $(H_q^i \mathbf{a}_j)(H_q^i \mathbf{a}_j)^T$. The inverse of a matrix might take much time but, fortunately, for $(A_q^T A_q)^{-1}$ there is an efficient recursive

relation of $(A_q^T A_q)^{-1}$ to $(A_{q+1}^T A_{q+1})^{-1}$ and $(A_{q-1}^T A_{q-1})^{-1}$ (see [9] for details). So the complexity of one iteration is $O(n^2)$, provided that the number k of equalities and inequalities is $O(n)$.

The correctness of the algorithm was proved by Goldfarb [9] for exact line search in Step 3 and the DFP formula in Step 5. But it is not hard to follow the proof in [9] to show the correctness of our algorithm, being careful that [9] was for maximum while our algorithm is for minimum. We tried both the BFGS and DFP formula and found that BFGS is really better. Shanno compared them in [23] for many functions, and the results are generally in favor of the BFGS formula.

References

[1] Alameldin, T., Palis, M., Rajasekaran, S., and Badler, N.I. On the Complexity of Computing Reachable Workspaces for Redundant Manipulators. *OE/Boston'90 Symposium on Advances in Intelligent Systems. Intelligent Robots and Computer Vision IX: Algorithms and Complexity*, 1990.

[2] Badler, N. I., O'Rourke, J., and Kaufman,B. Special problems in human movement simulation. *ACM Computer Graphics* 14, 3 (1980), 189-197.

[3] Badler, N. I., Manoochehri, K. H., and Walters, G. Articulated figure positioning by multiple constraints. *IEEE Computer Graphics and Applications* 7, 6 (June 1987), 28-38.

[4] Badler, N. I., Phillips, C. B., and Webber, B. L. *Simulating Humans: Computer Graphics Animation and Control*. Oxford University Press, New York, 1993.

[5] Barzel, R., and Barr, A. H. A modeling system based on dynamic constraints. *ACM Computer Graphics* 22, 4 (1988), 179-188.

[6] Broyden, C. G. The Convergence of a Class of Double-rank Minimization Algorithms, 2. the New Algorithm. *Journal of the Institute of Mathematics and Its Applications* 6 (1970), 222-231.

[7] Fletcher, R., and Powell, M. J. D. A rapidly convergent descent method for minimization. *Computer J.* 6 (1963), 163-168.

[8] Fletcher, R. A new approach to variable metric algorithms. *Computer J.* 13 (1970), 317-322.

[9] Goldfarb, D. Extension of Davidon's variable metric method to maximization under linear inequality and equality constraints. *SIAM J. Appl. Math.* 17 (1969), 739-764.

[10] Goldfarb, D. A family of variable metric methods derived by variational means. *Math. Computation* 24 (1970), 23-26.

[11] Girard, M., and Maciejewski, A.A. Computational modeling for the computer animation of legged figures. *ACM Computer Graphics* 19, 3 (1985), 263-270.

[12] Korein, J., and Badler, N. I. Techniques for goal directed motion. *IEEE Computer Graphics and Applications* 2, 9 (1982), 71-81.

[13] Korein, J. *A Geometric Investigation of Reach*. MIT Press, 1985.

[14] Lee, P., Wei, S., Zhao, J., and Badler, N. I. Strength guided motion. *ACM Computer Graphics* 24,4 (1990), 253-262.

[15] Monheit, G. and Badler, Norman I. A Kinematic Model of the Human Spine and Torso. *IEEE Computer Graphics and Applications* 11, 2 (1991), 29-38.

[16] Paul, R. *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, 1981.

[17] Phillips, C. B., and Badler, N. I. Jack: A toolkit for manipulating articulated figures. *ACM/SIGGRAPH Symposium on User Interface Software*, Banff, Canada, October 1988.

[18] Phillips, C. B., Zhao, J., and Badler, N. I. Interactive real-time articulated figure manipulation using multiple kinematic constraints. *ACM Computer Graphics* 24, 2 (1990), 245-250.

[19] Powell, M. J. D. A hybrid method for nonlinear equations. *Numerical Methods for Nonlinear Algebraic Equations*, Rabinowitz, P. Ed., Gordon and Breach Science Publishers, 1970.

[20] Powell, M. J. D. Some global convergence properties of a variable metric algorithm for minimization without exact line searches. *Nonlinear Programming, SIAM-AMS Proc.* IX (1976), 53-72.

[21] Ritter, K. A variable metric method for linearly constrained minimization problems. *Nonlinear Programming 3*, Mangasarian, O. L., Meyer, R. R., and Robinson, S. M.(Eds), Academic Press, 1978.

[22] Rosen, J. B. The gradient projection method for nonlinear programming, Part I, Linear Constraints. *SIAM J. Appl. Math.* 8 (1960), 181-217.

[23] Shanno, D. F. Conditioning of quasi-Newton methods for function minimization. *Math. Computation* 24 (1970), 647-664.

[24] Witkin, A., Fleischer, K., and Barr, A. Energy constraints on parameterized models. *ACM Computer Graphics* 21, 4 (1987), 225-232.

[25] Witkin, A., and Welch, W. Fast animation and control of nonrigid structures. *ACM Computer Graphics* 24, 4 (1990), 243-252.

[26] Whitney, D. E. The mathematics of coordinated control of prostheses and manipulators. *J. Dynamic Systems, Measurement, and Control, Transaction ASME* 94 (1972), Series G, 303-309.

[27] Zhao, J. Moving Posture Reconstruction from Perspective Projections of Jointed Figure Motion. Ph.D. dissertation, University of Pennsylvania, Philadelphia, 1993.