

Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures

JIANMIN ZHAO and NORMAN I. BADLER
University of Pennsylvania

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 to operate 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*[™], a human figure simulation system developed at the University of Pennsylvania, and proves to be satisfactorily efficient, controllable, and robust. A spatial constraint in our system involves two parts: one constraint on the figure, the *end-effector*, and one on the spatial environment, the *goal*. These two parts are dealt with separately, so that we 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. If 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.

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

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Articulated figures, inverse kinematics, nonlinear programming

This research is partially supported by ARO grant DAAL03-89-C-0031 including participation by the U.S. Army Research 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 grants CDA88-22719 and USE-9152503.

Authors' addresses: J. Zhao, Bellcore, RRC-4E307, 444 Hose Lane, Piscataway, NJ 08854; N. I. Badler, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104-6389.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1994 0730-0301/94/1000-0313\$03.50

1. INTRODUCTION

In computer animation, an articulated figure is often modeled as a set of rigid segments connected by joints. Abstractly, a joint is a constraint on the geometric relationship between two adjacent segments. This relationship is expressed with a number of parameters called joint angles. With judicious selection of joints so that, for example, segments are connected to form a tree structure, a collection of the joint angles of all the joints corresponds exactly to a configuration of the figure. This correspondence provides an immediate computer representation of articulated figure configurations such that given a set of joint angles, it is straightforward to compute the corresponding configuration; however, the problem of finding a set of joint angles that corresponds to a given configuration—the inverse kinematics problem—is nontrivial.

The inverse kinematics problem is extremely important in computer animation because it is often the spatial appearance, rather than the joint angles, that an animator finds easier to express. Naturally, the problem has received considerable attention in computer animation as well as in robotics (see section 2), 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.

The several applications of inverse kinematics for articulated figures are as follows:

- In interactive *manipulation*, where an animator poses a figure in the spatial environment, joint angles are merely internal (and possibly hidden) representations of postures (configurations) [Phillips et al. 1990]. The joint angles defining the target configuration may be much more important than the process taken by the joint angles to arrive at the target: for manipulation, interactive responsiveness is essential.
- Fast response is also essential for *animation* control of articulated figures where the mapping from spatial configurations to joint angles must be done repeatedly. For example, if the path of an end-effector is described by a space curve [Girard and Maciejewski 1985] or by strength constraints [Lee et al. 1990], the prediction of the next configuration along the path is iteratively transformed to joint angles at regular time steps.
- Inverse kinematics may be used to compute the *reachable workspace* of an end-effector [Alameldin et al. 1990]. The joint angles themselves are not important. Rather, they answer the questions of whether a spatial configuration can be achieved or what the overall shape of the workspace looks like. These issues arise in human factors evaluations [Badler et al. 1993].
- With additional control mechanisms and collision detection, inverse kinematics can be augmented to achieve *collision avoidance* [Zhao 1994]. Combining spatial and joint constraints into collision-free motion is a fundamental goal of robotics, and a variety of exact and heuristic solutions exist. The problem is that the complexity grows exponentially with the number of degrees of freedom, making exact solutions effectively impossible on a figure with the articulation of a simulated human.

We offer a new and feasible approach to the inverse kinematics problem based on nonlinear programming: a numerical method for solving the minimum of a nonlinear function. Our approach searches for the solution in the high-dimensional joint angle space for computational economy. It also deals with joint limits intrinsically rather than as a special case. Last, it has been successfully implemented and has found numerous applications.

Because of the complex nature of nonlinear functions, many efficient nonlinear programming algorithms terminate when they find local minima. The algorithm we chose 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 local minima are encountered during interactive manipulation, users can easily perturb the figure configuration slightly to get around the problem. Often, additional spatial constraints will help guide a solution away from local minima. For example, weak constraints to keep the hand away from the body and the elbow down will often prevent the arm from falling into “gimbal lock” configurations.

2. BACKGROUND

Inverse kinematics for determining mechanism motion is a common technique in mechanical engineering, particularly in robot research [Paul 1981]. In robotics, however, the greatest concern is with the functionality of manipulators; overly redundant degrees of freedom are usually not desired, except when needed for special purposes. Moreover, the computation is usually carried out on particular (fixed) linkage geometries. In contrast, many interesting objects in the computer animation domain, such as human figures and other characters, have many redundant degrees of freedom when viewed as a tree-structured kinematic mechanism.¹ So, it was necessary to look for effective means for solving this problem under circumstances applicable 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 [Korein and Badler 1982; Korein 1985]. Redundancy in the arm linkage was reduced to an “elbow circle” that could be separately constrained by, for example, joint limits or gravity considerations. Badler et al. [1987] used 3D position constraints to specify spatial configurations of articulated figures. A simple recursive solver computed joint angles to satisfy and combine multiple position constraints.

Girard and Maciejewski [1985] adopted a method from robotics. In their work they calculated the pseudoinverse of the Jacobian matrix that relates the increment of the joint angles to the displacement of the end-effector in space. The main formula is

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

¹There is an alternative method of viewing such figures as *deformable* objects, in which case different representational structures and manipulation techniques are applicable.

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 pseudoinverse of the Jacobian $\partial\mathbf{r}/\partial\theta$. To understand this, we can think of \mathbf{r} as a 3D column vector denoting the position of the hand, and θ as an n -dimensional 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 a continuous way. However, to position a human figure simply in a computer-simulated environment, it would not be economical to move the end-effector \mathbf{r} by “small” steps. Also, in making a computer animation sequence, it would not be optimal either to take a step size smaller than necessary. Moreover, the pseudoinverse calculations required for each step in this formula are normally quite expensive, and they did not address joint limits.

Witkin et al. [1982] used energy constraints for position and orientation. The constraints are satisfied if and only if the energy function is zero. Their constraint solution integrates the differential equation:

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

where θ is the parameter (e.g., joint angle) vector that defines the configuration of the system; E is the energy function of θ ; and ∇ is the gradient operator. If $\theta(t)$ is the integral with some initial condition, $E(\theta(t))$ decreases monotonically 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 isoenergy line, on which the energy function E takes identical values. For any number (energy level), there is such a line. Under this physical interpretation of the energy function, Witkin et al.’s method [1982] searches the path from the initial configuration to the target configuration which is, at any point, perpendicular to the isoenergy lines.

Instead of associating energy functions with constraints, Barzel and Barr [1988] introduced deviation functions which measure the deviation of two constrained parts. They discussed a variety of constraints—such as point-to-point, point-to-nail, etc.—and their associated deviation functions. A segment in a system of rigid bodies is subjected to both external forces, such as gravity, and constraint forces, which force the deviations to zero whenever they are found to be positive. Constraint forces are solved from a set of dynamic differential equations that requires that all deviations go to zero exponentially in a certain amount of time.

An approach based on physical modeling and interpretation is also used by Witkin and Welch [1990] on nonrigid bodies whose deformations are controlled by a number of parameters. To apply this method 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, adding to the number of constraint equations substantially. Second, a joint of an articulated figure is meant to be an absolute constraint: it should not compete with any constraint that relates a point on a segment of the figure to a point in space. Such competition leads often 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 start state. This process has some physical meaning. In Girard and Maciejewski's method [1985], the process is determined by the end-effector path; in Witkin et al.'s method [1982], it is determined by the energy function (the path in θ space is perpendicular to the family of isoenergy lines); in Barzel and Barr's method [1988] or other dynamic methods such as those in Witkin and Welsh [1990], the process is determined by the physical interpretations of each segment and by external and constraint forces exerted on it. Not only can these methods solve the constraints, but they 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, 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 curve, one may first choose a small step size and then compute the function value until it rises. Another way to find a solution could be to first locate an interval in which the minimum lies, then to use the golden ratio method (a method similar to binary search) to find the minimum. The first method gives a direct, but gradual picture of how the function changes to the minimum, 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 Witkin et al. [1987]), 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 permitting the specification of additional constraints, as long as the solution 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 unconstrained nonlinear programming, where the variables are free to take any values; the other one is constrained nonlinear programming, where the variables can only take val-

ues 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 result in 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 joint integrity and joint limits must not be violated.

3. SPATIAL CONSTRAINTS

The basic entity considered here is the articulated figure. Such objects are defined in the *Peabody* language developed for the *Jack*[®] software system at the University of Pennsylvania [Badler et al. 1993]. A *Peabody* figure consists 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.

Given the data structure, we need to address the problem of placing a figure into a desired posture. As discussed in section 1, 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 stick 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:

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

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

$$L = R^3 \times S^2 \times S^2, \quad (2)$$

where R^3 denotes the set of 3D vectors, and S^2 the set of 3D unit vectors. Accordingly, $\mathbf{e}(\theta)$ is a 9D vector, whose first component triple forms a positional vector designating the spatial position of a point on the end-effector segment, and whose second and third component triples form two unit

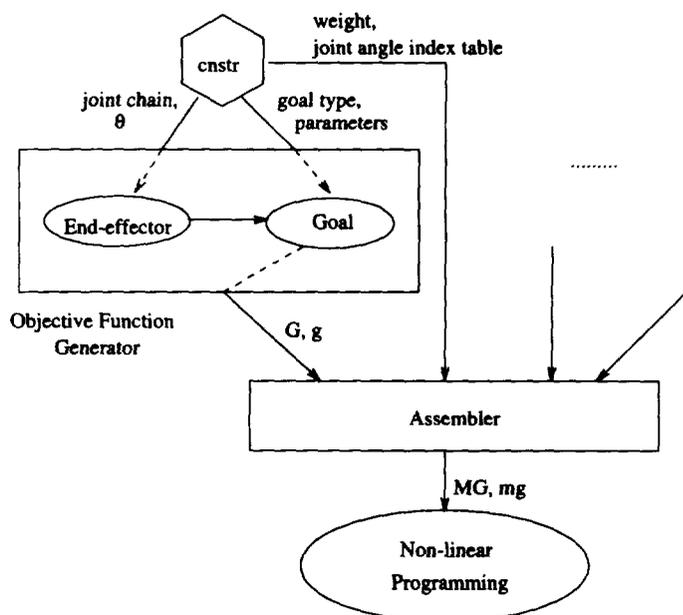


Fig. 1. Spatial multiconstraint system.

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 9D vector $\mathbf{e}(\theta) \in L$ according to the figure definition. Since segments of a figure are rigid, the angle between 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 remove three degrees of freedom from $\mathbf{e}(\theta)$, so that $\mathbf{e}(\theta)$ has only six independent quantities, exactly what is needed to determine the position and orientation of a rigid body in space.

Consider an example. Let the end-effector segment be the right hand, and let the pelvis be fixed temporarily to serve as the root of the figure tree. 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 put in $\mathbf{e}(\theta)$, provided that a point and two orthonormal vectors attached to the hand have been selected for reference.

In practice, the components in $\mathbf{e}(\theta)$ may not be equally important. Sometimes we may be interested only in the position (the first three) components, say, for the location of the tip of the index finger. At other times, we may be interested in only one of the two unit vector components in $\mathbf{e}(\theta)$, say, for the direction (but not the roll angle) of the (unit) vector corresponding to the index finger. We may also want two unit vector components in $\mathbf{e}(\theta)$ to define

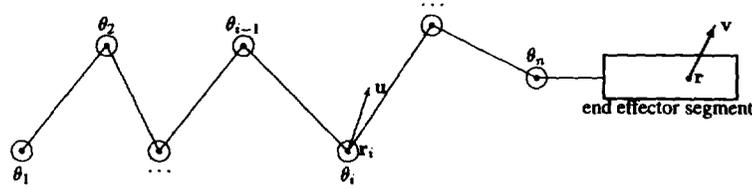


Fig. 2. Joint chain.

the entire orientation of the end-effector segment. In general, we need to permit any combination of these cases. So, the end-effector mapping \mathbf{e} is labeled with a type, and only the desired components are present in $\mathbf{e}(\theta)$.

4.2 End-Effector Computational Module

The End-Effector module is 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 that 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 one-degree-of-freedom joints with zero distance in between. The length of the joint chain is the total number of (one-degree-of-freedom) joints along the chain; in Figure 2, it is n .

Given $\theta = (\theta_1, \theta_2, \dots, \theta_n)^T$, where superscript T denotes the transposition operator, this module computes 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

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

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.² Let the i th 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

²The translational joints can be treated similarly and are actually even simpler.

(see Whitney [1972]):

$$\frac{\partial \mathbf{r}}{\partial \theta^i} = \mathbf{u} \times (\mathbf{r} - \mathbf{r}_1) \quad (3)$$

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

5. GOALS

5.1 Goal Potential Functions

A goal can also be viewed as a mapping:

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

where the domain L is the same as the range of the end-effector mapping defined in (2), and R^+ is the set of nonnegative 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 nonnegative real number, $P(\mathbf{e}(\theta))$, which is understood as the distance from the current end-effector location (position and/or orientation) to the associated goal. For each end-effector and goal pair, 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 computes the potential $P(\mathbf{x})$ and its gradient, the column vector formed by all its partial derivatives. Let

$$\nabla_{\mathbf{x}} = \left(\frac{\partial}{\partial x^1} \quad \frac{\partial}{\partial x^2} \quad \cdots \quad \frac{\partial}{\partial x^n} \right)^T$$

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

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 3D 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

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

and the gradient

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

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, this may not be adequate nor desirable, 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. 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,

$$c_d = 360/(2\pi d). \quad (8)$$

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

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

The gradient is then

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

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

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

Position/Orientation Goals. The position and orientation goals can be treated separately, 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:

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

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 $\nabla_{\mathbf{r}}P$, $\nabla_{\mathbf{x}_r}P$, and $\nabla_{\mathbf{y}_r}P$ 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

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

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

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

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

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. A point \mathbf{r} on the end-effector segment will be constrained to lie on this line.

The potential function is

$$P(\mathbf{r}) = ((\mathbf{p} - \mathbf{r}) - (\mathbf{p} - \mathbf{r}) \cdot \nu \nu)^2. \quad (16)$$

Its domain is

$$L = R^3,$$

and its gradient is

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

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

The potential function is

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

Its domain is

$$L = R^3,$$

and the gradient is

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

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 space into two half-spaces. 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 $\mathbf{p} + \nu$.

The potential function is

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

Its domain is

$$L = R^3.$$

And the gradient is

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

6. SPATIAL CONSTRAINT AS A NONLINEAR PROGRAMMING PROBLEM

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

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

This quantity can be computed by first invoking the end-effector module to compute $\mathbf{e}(\theta)$, 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 because the goal (set) is not always reachable. Thus, the problem would be naturally generalized 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, and the joint angles for the shoulder are confined in a polygon. These limits can all be expressed as linear inequalities. Therefore, we recast the problem as nonlinear programming subject to linear constraints on variables. Formally,

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

where \mathbf{a}_i , $i = 1, 2, \dots, k$ 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 represent the lower limit l^i and upper limit u^i on

θ^i , the i th 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. Research in nonlinear programming is mostly done to solve for local minima. It is still 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 suffice.

For interactive response rates, we chose to compromise for local, rather than global, minima. From years of observation, we have not seen many serious problems. The algorithm we used to solve 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 iterative steps.

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

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

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. This is the case because 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 \mathbf{g} are computed by the Objective

Function Generator at the request of the Nonlinear Programming module 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 Girard and Maciejewski [1985], Witkin et al. [1987], and Barzel and Barr [1988] where the constraint is satisfied as a result of evolution from the initial configuration, this method approaches 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 assume a feasible but undesirable position. An additional constraint for the elbow could be necessary for a satisfactory posture. More constraints would be needed for more complex postures.

Our system, therefore, handles multiple constraints. Since the objective function $G(\theta)$ defined in (22) is nonnegative, multiple constraints are solved by minimizing the sum of the objective functions associated with all the goals

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

where m is the number of constraints; subscript i denotes the association with the i th constraint; w_i is a nonnegative weight assigned to the i th constraint to reflect the relative importance of the constraint; and

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

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

Note that the $G_i(\theta)$ 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 because for the gradient operator ∇_{θ} is additive.

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

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

It is useful for collision avoidance, 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 for multiple constraints can be found by computing separately and independently the

objective functions for individual constraints and then adding them together. In this section, we shall explain how the Assembler works.

The Objective Function Generator module takes a joint chain, an array of corresponding joint angles, a 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. Two gradient vectors so structured for different constraints do not add directly, however: the i th joint angle in one chain may not be the same as the i th 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 i th constraint, and n_i be the number of joint angles in Θ_i . Let

$$\Theta = \bigcup_{i=1}^m \Theta_i, \quad (28)$$

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

$$\left\{ \begin{array}{l} M_i: \{1, 2, \dots, n_i\} \rightarrow \{1, 2, \dots, n\} \\ j \mapsto M_i(j), \end{array} \right. \quad (29)$$

such that the j th 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, is passed to the Assembler so that the effect of the i th constraint on the gradient of the overall objective function G^{all} can be correctly established. Once the g_i^j —the derivative of the objective function of the i th constraint G_i with regard to the j th 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 $j = 1, 2, \dots, n_i$,

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

10. RECONCILIATION OF JOINT CHAINS

It was suggested in (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 to be used as resources for constraint satisfaction. Thus, the joint chain need not go all the way from the end-effector back to the figure definition root: it may be specified when the constraint is defined. Since the constraints may be input sequentially, a joint that may affect the end-effector of one constraint but is not picked for the

joint chain, could 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 may later be selected for the constraint on the left hand. Similar observations were previously made in Badler et al. [1980].

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

Our nonlinear programming algorithm 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 respectively. 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 spatial multiconstraint system, in which the rigidity of individual degrees of freedom can be controlled, has been implemented in *Jack* [Badler et al. 1993]. 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 maps from spatial specifications of articulated figure configurations to joint angles.

The pose displayed in the left panel of Figure 3 is achieved by using six 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 on 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 body in Figure 3 has 17 segments. A three-degree-of-freedom joint connects each pair of vertebral segments. These joints, however, are grouped together to form a *joint group* that is driven by three independent parameters: forward extension, lateral bending, and axial rotation. So, the number of effective degrees of freedom of the torso is just three. 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

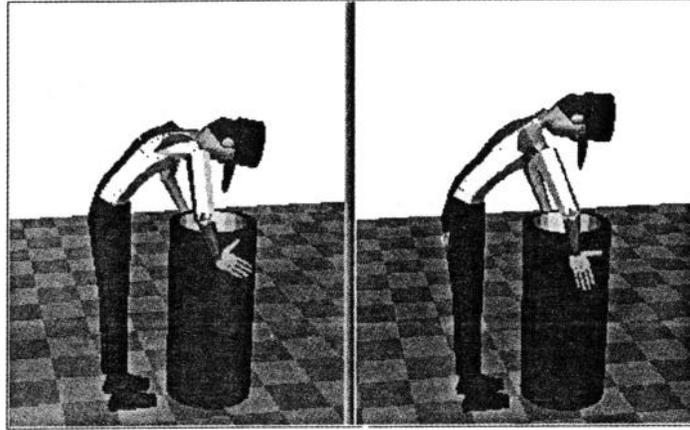


Fig. 3. Looking down toward the end of the tube (left panel: vertebral joints are grouped; right panel: vertebral joints are independent).

three effective degrees of freedom: elevation, abduction, and twist. The new modeling construct *joint group* developed for interdependencies among joints and their incorporation into the spatial multiconstraint system are described in Badler et al. [1993] and Zhao [1993]. In all, there are 22 degrees of freedom: the others 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. Running on a Silicon Graphics workstation 340 VGX, and starting from an upright neutral position, the solution took about five seconds.

For comparison, the right panel of Figure 3 demonstrates the result of exactly the same task as above except that the vertebral joints are not grouped. Since there are 17 vertebral joints, each with three degrees of freedom, there are 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 vertebral joints is interesting. Figure 4 shows the joint angle distribution along the (grouped) vertebral joints corresponding to the pose in the left panel of Figure 3. The 17 segments consist of 12 thoracic and five lumbar vertebrae. They are numbered from the top with the first lumbar vertebra succeeding the twelfth thoracic vertebra. In Figure 4, T1 denotes the joint connecting the second thoracic vertebra to the first thoracic vertebra, and so on. Note that T12 denotes the joint connecting the first lumbar vertebra to the twelfth thoracic vertebra, and L5 denotes the joint connecting the lower torso (the sacrum) to the fifth lumbar. Figure 5 shows the joint angle distribution corresponding to the right panel of Figure 3 (with independent vertebral joints). In comparing Figures 4 and 5, it is clear that although both achieve a solution, the inverse kinematics algorithm based on function optimization methods alone results in irregular joint angle distribution along independent vertebral joints. Figure 4 illustrates more uniform results that are achieved by appropriately grouping these joints together. Our distributions are based on a kinematic model of the

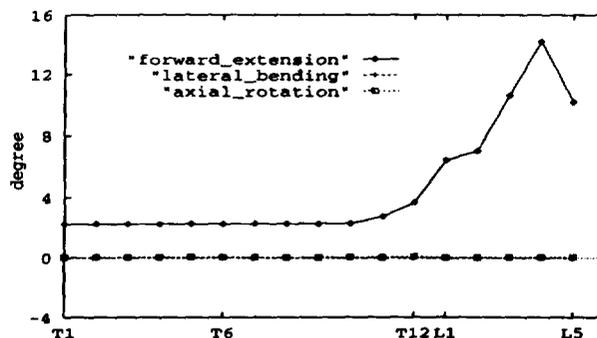


Fig. 4. Joint angle distribution along the grouped vertebral joints.

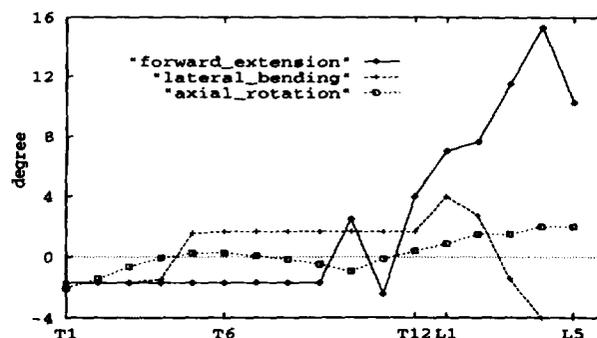


Fig. 5. Joint angle distribution along the independent vertebral joints.

human spine constructed by Monheit and Badler [1991] and Badler et al. [1993].

The human models in Figures 6 to 8 are more primitive *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 the 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 involves an equal number of degrees of freedom; it is worth explaining why the task in Figure 3

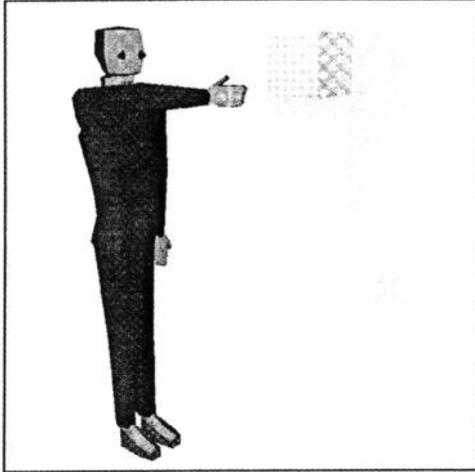


Fig. 6. Goal not reachable without activating the torso.



Fig. 7. Successful but awkward reach.

took about twice as long as the task in Figure 8, despite the clear superiority in speed that the Silicon Graphics 340 VGX enjoys over 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 spatial complexity (determined by 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 by convergence rate. Obviously, the task in Figure 3 is much more

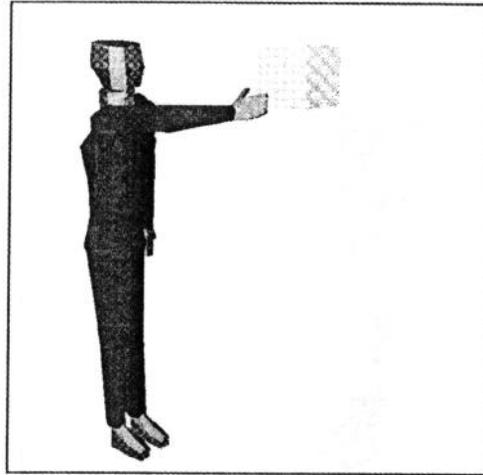


Fig. 8. Natural reach with controlled rigidities in the vertebral joints.

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 or challenging inverse kinematics tasks has yet to be developed.)

Over years of use, this inverse kinematics algorithm has proved to be robust and highly interactive, especially when constraints are manipulated on-screen [Phillips et al. 1993]. The algorithm has been highly successful in modeling balance “behaviors” that take the static mass distribution of a figure into account during interactive manipulation of animation and move various body joints to restore balance [Phillips and Badler 1991; Badler et al. 1993]. Finally, inverse kinematics has been used to generate interactive collision and self-collision avoidance in the *Jack* figure [Zhao 1994]. While it can be faulted for occasionally producing unnatural joint angle transitions into a posture, it is controllable with additional spatial constraints, joint rigidities, or combined joint groups. The result is a tool of great versatility and ease of use for interactive articulated-figure manipulation.

APPENDIX: 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 problem (23). Without constraints on joint angles, the variable metric method (or conjugate gradient method) is considered a good choice. To deal with linear equality and inequality constraints in (23), Rosen [1960] proposed the projection method, by which the search direction determined from the corre-

sponding unconstrained problem is orthogonally projected to the subspace defined by those constraints on variables. Goldfarb [1969] combined DFP's method (a variable metric algorithm) [Fletcher and Powell 1963] with Rosen's projection method. After that, the variable metric method underwent further improvements. The BFGS method [Broyden 1970; Fletcher 1970; Goldfarb 1970; Shanno 1970] has been considered most successful. One motivation for the improvement is to get the best conditioning of the approximate inverse Hessian matrix [Shanno 1970]. The algorithm we present 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, Broyden [1970], Fletcher [1970], Goldfarb [1970], and Shanno [1970] 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 in (23) are unit vectors. We say that point θ is feasible if it satisfies all the equalities and inequalities in (23). The i th 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 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 , 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 i th iteration.

Step 1. 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 , and the first l columns of A_q are $\{\mathbf{a}_i : i = 1, 2, \dots, l\}$. H_q^0 is computed by applying (32) q times; $g^0 = g(\theta^0)$.

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

Step 3. If the algorithm did not terminate at Step 2, either $\|H_q^i g^i\| > \max\{0, \frac{1}{2}\alpha_q \alpha_{qq}^{-1/2}\}$ or $\|H_q^i g^i\| \leq (1/2)\alpha_q \alpha_{qq}^{-1/2}$, where it is assumed that $\alpha_q \alpha_{qq}^{1/2} \geq \alpha_i \alpha_{ii}^{-1/2}$, $i = l + 1, \dots, q - 1$, and α_{ii} is the i th diagonal element of $(A_q^T A_q)^{-1}$. They are all positive. (See Goldfarb [1969].) If the former holds, proceed to Step 4.

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

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

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 q th column of A_q ; and A_{q-1} is the n -by- $(q-1)$ matrix obtained by taking off the q th column from A_q .

Let $q \leftarrow q - 1$, and go to Step 2.

Step 4. Let the search direction $s^i = -H_q^i g^i$, and compute

$$\lambda_j = \frac{b_j - \mathbf{a}_j^T \theta^i}{\mathbf{a}_j^T s^i}, \quad j = q + 1, q + 2, \dots, k$$

$$\lambda^i = \min_j \{\lambda_j > 0\}.$$

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

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

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

$$H_{q+1}^{i+1} = H_q^i - \frac{H_q^i \mathbf{a}_j \mathbf{a}_j^T H_q^i}{\mathbf{a}_j^T H_q^i \mathbf{a}_j}. \quad (32)$$

Set $q \leftarrow q + 1$ and $i \leftarrow i + 1$, and go to Step 2.

Step 6. 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:

$$H_q^{i+1} = H_q^i + \left(\left(1 + \frac{(y^i)^T H_q^i y^i}{(\sigma^i)^T y^i} \right) \sigma^i (\sigma^i)^T - \sigma^i (y^i)^T H_q^i - H_q^i y^i (\sigma^i)^T \right) / (\sigma^i)^T y^i. \quad (33)$$

Else use the DFP formula:

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

Set $i \leftarrow i + 1$, and go to Step 2.

The inexact line search strategy (31) in Step 4 was proposed by Powell [1976] 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, which we feel is fairly effective. (The switch between the BFGS formula and the DFP formula in Step 6 was suggested by Fletcher [1970].)

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 Goldfarb [1969] 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 [1969] for exact line search in Step 4 and the DFP formula in Step 6. But, it is not hard to follow the proof in Goldfarb [1969] to show the correctness of our algorithm, being careful that his algorithm was for maximum while our's is for minimum. We tried both the BFGS and DFP formula and found that BFGS is really better. Shanno [1970] compared them for many functions, and the results are generally in favor of the BFGS formula.

REFERENCES

- ALAMELDIN, T., PALIS, M., RAJASEKARAN, S., AND BADLER, N. I. 1990. On the complexity of computing reachable workspaces for redundant manipulators. In *OE/Boston '90 Symposium on Advances in Intelligent Systems. Intelligent Robots and Computer Vision IX: Algorithms and Complexity*. SPIE, Bellingham, Wash.
- BADLER, N. I., MANOOCHEHRI, K. H., AND WALTERS, G. 1987. Articulated figure positioning by multiple constraints. *IEEE Comput. Graph. Appl.* 7, 6 (June), 28-38.
- BADLER, N. I., O'ROURKE, J., AND KAUFMAN, B. 1980. Special problems in human movement simulation. *ACM Comput. Graph.* 14, 3, 189-197.
- BADLER, N. I., PHILLIPS, C. B., AND WEBBER, B. L. 1993. *Simulating Humans: Computer Graphics Animation and Control*. Oxford University Press, New York.
- BARZEL, R. AND BARR, A. H. 1988. A modeling system based on dynamic constraints. *ACM Comput. Graph.* 22, 4, 179-188.
- BROYDEN, C. G. 1970. The convergence of a class of double-rank minimization algorithms, 2. The new algorithm. *J. Inst. Math. Appl.* 6, 222-231.
- FLETCHER, R. AND POWELL, M. J. D. 1963. A rapidly convergent descent method for minimization. *Comput. J.* 6, 163-168.
- FLETCHER, R. 1970. A new approach to variable metric algorithms. *Comput. J.* 13, 317-322.
- GOLDFARB, D. 1970. A family of variable metric methods derived by variational means. *Math. Comput.* 24, 23-26.
- GOLDFARB, D. 1969. Extension of Davidson's variable metric method to maximization under linear inequality and equality constraints. *SIAM J. Appl. Math.* 17, 4, 739-764.
- GIRARD, M. AND MACIEJEWSKI, A. A. 1985. Computational modeling for the computer animation of legged figures. *ACM Comput. Graph.* 19, 3, 263-270.
- KOREIN, J. AND BADLER, N. I. 1982. Techniques for goal directed motion. *IEEE Comput. Graph. Appl.* 2, 9, 71-81.
- KOREIN, J. 1985. *A Geometric Investigation of Reach*. MIT Press, Cambridge, Mass.
- LEE, P., WEI, S., ZHAO, J., AND BADLER, N. I. 1990. Strength guided motion. *ACM Comput. Graph.* 24, 4, 253-262.
- MONHEIT, G. AND BADLER, N. I. 1991. A kinematic model of the human spine and torso. *IEEE Comput. Graph. Appl.* 11, 2, 29-38.

- PAUL, R. 1981. *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, Cambridge, Mass.
- PHILLIPS, C. B. AND BADLER, N. I. 1991. Interactive behaviors for bipedal articulated figures. *ACM Comput. Graph.* 25, 4, 359-362.
- PHILLIPS, C. B., ZHAO, J., AND BADLER, N. I. 1990. Interactive real-time articulated figure manipulation using multiple kinematic constraints. *ACM Comput. Graph.* 24, 2, 245-250.
- POWELL, M. J. D. 1976. Some global convergence properties of a variable metric algorithm for minimization without exact line searches. In *Nonlinear Programming, SIAM-AMS Proc. IX*. SIAM, Philadelphia, Pa., 53-72.
- POWELL, M. J. D. 1970. A hybrid method for nonlinear equations. In *Numerical Methods for Nonlinear Algebraic Equations*, P. Rabinowitz, Ed. Gordon and Breach Science Publishers, New York.
- RITTER, K. 1978. A variable metric method for linearly constrained minimization problems. In *Nonlinear Programming*, O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, Eds. Vol. 3. Academic Press, New York.
- ROSEN, J. B. 1960. The gradient projection method for nonlinear programming, Part I, Linear Constraints. *SIAM J. Appl. Math.* 8, 1, 181-217.
- SHANNO, D. F. 1970. Conditioning of quasi-Newton methods for function minimization. *Math. Comput.* 24, 647-664.
- WITKIN, A. AND WELCH, W. 1990. Fast animation and control of nonrigid structures. *ACM Comput. Graph.* 24, 4, 243-252.
- WITKIN, A., FLEISCHER, K., AND BARR, A. 1987. Energy constraints on parameterized models. *ACM Comput. Graph.* 21, 4, 225-232.
- WHITNEY, D. E. 1972. The mathematics of coordinated control of prostheses and manipulators. *J. Dynam. Syst. Meas. Control Trans. ASME* 94, 303-309.
- ZHAO, X. 1994. Near real-time body awareness. *Comput.-Aided Des.* In press.
- ZHAO, J. 1993. Moving posture reconstruction from perspective projections of jointed figure motion. Ph.D. dissertation, Computer and Information Science, MS-CIS-93-74, Univ. of Pennsylvania, Philadelphia, Pa.

Received August 1990; revised December 1992 and August 1994; accepted August 1994