INTERACTIVE CREATION OF ANIMATIONS: AN OPTIMIZATION APPROACH TO REAL-TIME INVERSE KINEMATICS

by

Sumeet Bawa

B.E., Electrical and Electronics Engineering

Regional Engineering College

Tiruchirapalli, India, 1990

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the School

of

Computing Science

SIMON FRASER UNIVERSITY

April 1995

APPROVAL

Name: Sumeet Bawa

Degree: Master of Science

Title of thesis: Interactive Creation of Animations: An Optimization Approach to Real-time Inverse Kinematics

Examining Committee: Dr. William Havens
Chair

Dr. Thomas W. Calvert
Senior Supervisor

Dr. John Dill
Supervisor

Dr. F. David Fracchia
Supervisor

Dr. Lou Hafer
External Examiner

Date Approved: April 19, 1995

Abstract

An articulated figure, such as a human skeleton, can be described to a first approximation as a hierarchical collection of rigid segments connected at revolute joints. A wide spectrum of techniques has been tried for animating such figures. At one end of the spectrum is the interpolation of keyframes which are created by explicitly setting the joint angles. This technique gives the animator complete control over the animation, but the task of creating keyframes becomes very difficult and tedious as the number of degrees of freedom of the figure being animated increases. At the other end of the spectrum are procedural animation of specific movement patterns and simulation of models of dynamic behavior of the figures. The former is limited in its application while the latter is intractable, and provides very limited control over the animation.

If keyframe creation can be automated or expedited, it can very effectively aid the task of making animations while still leaving full control with the animator. This can be achieved by automating the inverse kinematic calculation of the joint angle values required to make the figure attain a specific posture.

This thesis presents the development and implementation of a system which uses an iterative nonlinear constrained optimization algorithm for solving the problem of inverse kinematics in the presence of constraints on the figure. A function of the difference in the current and desired posture of the figure, called the objective function, is minimized to allow direct manipulation of the articulated figure. The algorithm displays super-linear convergence and allows interactive manipulation of complex figures. Physical integrity of the figure is maintained and all the constraints imposed on the figure are satisfied at all times. The algorithm requires first and second order partial derivatives of the objective function and constraints, which for a general articulated figure, requires symbolic computation. Algorithms for efficient procedural evaluation of these quantities for any tree structured planar figure are developed.

Each iteration of the optimization algorithm requires inversion and recalculation of matrices. Techniques developed in the field of numerical optimization, which have not been previously used for articulated figure animation, are employed to factorize various matrices. Availability of factors allows efficient inversion and recalculation of the matrices involved.

Previous attempts at direct manipulation of articulated figures have been limited to manipulating one part of the figure at a time to produce the keyframes, which are analogous to snapshots of the complete movement. Our approach extends the concept to allow manipulation of multiple parts of the figure simultaneously by associating a trajectory with each part. The use of the trajectories permits development of complete movement sequences.

dedicated to my father Sh. Amarjit Singh Bawa

Acknowledgments

I wish to convey my gratitude to all those who have helped me at various times during my research and writing of this thesis. I am thankful to my supervisor, Dr. Tom Calvert, for his guidance and unflagging support throughout my graduate studies. A special thanks to Sanjeev Mahajan for being very helpful and encouraging during the early stages of this research. I am also thankful to Frank Henigman who was ever patient and helpful in resolving technical hitches during the system development.

Finally, I will always be indebted to my parents, late Sh. Amarjit Singh Bawa and Smt. Kamla Bawa, and my brother, Vineet Bawa, for their confidence in my abilities. This thesis would not have been possible without their moral encouragement.

Contents

List of Figures

Chapter 1

Introduction

Using the computer to generate physically realistic and visually pleasing animation sequences of objects and figures has been a major goal for computer graphics research. While creating real life interactions between inanimate objects in the presence of external forces is an important field of research, it is the simulation of the behavior of living beings which inarguably evokes maximum interest. The ability to produce a convincing model of a living being on a computer and to simulate its actual behavior and movement patterns finds application in fields as diverse as entertainment and ergonomics.

The most intuitive solution for simulating the movement of a human being is to perform a dynamic analysis of its detailed biomechanical model under the influence of external forces. This approach, however, has a number of problems. Biomechanical models and dynamic and kinematic couplings between various body parts are not very well understood. Moreover, accurate dynamic analysis of such complicated systems is intractable given today's methods. However, research has been directed in this direction and systems have been reported which produce striking animations of articulated figures. But this success is limited to a very small set of movements and very simple figures. Moreover, the animations are not produced in real-time, nor do these systems provide much control to the animator. Thus, physical realism is obtained at the cost of artistic quality in the movement.

The conventional approach to animating complex articulated figures has been by computer interpolation of keyframes which are created manually. This approach gives the animator complete control over the movement. However, most keyframing systems require explicit setting of individual joint angles of the figure to create the desired pose. This process becomes very tedious and error prone as the complexity of the figure increases, resulting in long production times.

If the conventional technique is augmented so as to allow direct manipulation of figures, then the task of keyframe creation will be greatly expedited. With such an approach, the user will be able to retain complete control over the animation. For instance, the user could interactively drag a part of the figure to the goal, and have the computer calculate the values for multiple joint angles to enable the requisite movement of the figure.

1.1 Problem Description

A basic requirement for creating and manipulating a realistic model of a vertebrate animal is the availability of means to represent its body, muscles, bones etc. The skeletal structure shorn of skin and muscles of the living being is the simplest approximation of its body. Then, making the skeleton move in a realistic manner represents the most fundamental problem that needs to be solved before a more complete model can be simulated effectively on the computer.

The skeleton of a vertebrate animal, for instance a human, is a hierarchical collection of rigid segments connected at revolute joints. A joint in the tree shaped skeleton is designated as the *root*. The root remains fixed at its location during the direct manipulation of the figure. The whole figure can be translated, without changing the relative orientation of the rest of the skeleton, by moving the root. Any joint which is used for direct manipulation of the figure is designated as the *end effector*. The part of the skeleton from the root to an end effector is termed as a *kinematic chain*. Calculation of the joint angles required to position the end effector at a desired point in space is an inverse kinematic problem. If $\mathbf{X} = [x_1, x_2, ..., x_m]^T$ is the position vector of the end effector and $\boldsymbol{\theta} = [\theta_1, \theta_2, ..., \theta_n]^T$ are the joint angles between the base of the kinematic chain and the end effector, then $\mathbf{X}$ can be represented as a function of $\boldsymbol{\theta}$ [1]:

$$\mathbf{X} = \mathbf{f}(\boldsymbol{\theta}) \tag{1.1}$$

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{\theta}) \\ f_2(\boldsymbol{\theta}) \\ \vdots \\ f_m(\boldsymbol{\theta}) \end{bmatrix} = \begin{bmatrix} f_1(\theta_1, \theta_2, ..., \theta_n) \\ f_2(\theta_1, \theta_2, ..., \theta_n) \\ \vdots \\ f_m(\theta_1, \theta_2, ..., \theta_n) \end{bmatrix}$$

Given a point P in space, with the position vector $\mathbf{X}_P = [x_{P1}, x_{P2}, ..., x_{Pm}]^T$, the problem of calculating the joint angle vector $\boldsymbol{\theta}_P = [\theta_{P1}, \theta_{P2}, ..., \theta_{Pn}]^T$ such that $\mathbf{X} = \mathbf{X}_P$ is called the *inverse kinematics* problem and requires finding the inverse of Equation 1.1.

1.2 Thesis and Demonstration

The thesis of this dissertation is that the task of articulated figure animation can be expedited without relinquishing any control over the animation by direct manipulation of the figure for the purpose of creating keyframes; direct manipulation of an articulated figure can be facilitated by automating inverse kinematic calculation of the joint angles required to take the end effector to the goal; the inverse kinematic problem can be solved effectively by using techniques from nonlinear optimization; physical integrity of the entity represented by the figure can be ensured and the effort involved in the creation of the keyframes can be reduced by specifying constraints on the figure; simultaneous manipulation of multiple end effectors can be effected and complete movement sequences can be created by associating a trajectory with each end effector.

The inverse kinematic problem of determining the joint angle vector $\boldsymbol{\theta}_P$ such that the end effector is placed at the goal is cast as an optimization problem where a suitable function, called the *objective function*, of the Euclidean distance between the end effector and the goal P, $\|\mathbf{X} - \mathbf{X}_P\|_2$, is minimized. Constraints are allowed to limit the acceptable values of the joint angles and to force any point on the figure

[1]See Appendix A for notation.

to be locked in its position. The joint angles $\boldsymbol{\theta}$ are the variables of the minimization and the solution of the minimization process is the required joint angle vector $\boldsymbol{\theta}_P$. Real-time performance is achieved by using an iterative optimization algorithm which converges rapidly. Techniques are used which improve the efficiency of each iteration of the algorithm.

As a proof of the concept, a simple editor is implemented which allows interactive creation of tree structured, planar articulated figures. The system allows the animator to interactively drag the end effector to the required position. The joint angle values required to attain the pose are determined automatically. Hence, the drudgery involved in the specification of the joint angles is obviated. The system also allows interactive specification of the limits on the acceptable values of the joint angles and the locking of the joints in their position. Multiple end effectors can be specified and a trajectory can be associated with each end effector.

1.3 Nonlinear Optimization

An iterative, nonlinear, constrained optimization algorithm is used to solve the inverse kinematic problem in the presence of constraints on the figure. The algorithm belongs to the class of *Projected Lagrangian Methods* [GMW81]. This class of algorithms poses a linearly constrained subproblem in each iteration, called major iteration. In particular, the algorithm used in this research poses a linearly constrained quadratic programming subproblem which is solved iteratively. Iterations associated with the solution of the subproblem are called minor iterations.

If k is the index of major iteration, the solution of the quadratic programming subproblem provides a search direction vector $\mathbf{p}_k$. A step length α_k along the search direction $\mathbf{p}_k$ which produces *sufficient* decrease in some metric, called the *merit function*, is determined. The joint angle vector $\boldsymbol{\theta}_{k+1}$ for the next iteration is set to $\boldsymbol{\theta}_k + \alpha_k \mathbf{p}_k$. The process continues untill the termination criteria are satisfied.

The minimization problem solved in this dissertation is characterized as having:

1. a non-linear, twice-continuously differentiable objective function,

2. linear equality/inequality constraints,
3. non-linear equality/inequality twice-continuously differentiable constraints and
4. initial values for variables of minimization always feasible with respect to constraints.

1.3.1 Calculation of Derivatives

The minimization algorithm described above requires evaluation of the gradient and the Hessian of the objective function as well as the gradients of the nonlinear constraints imposed on the figure[2]. Calculation of these quantities for a general articulated figure and general nonlinear constraints requires symbolic computation or forward/central difference approximations. These techniques are computationally expensive and can result in serious degradation in the performance. However, simplifications result if the figure is constrained to be planar. Algorithms for the efficient procedural evaluation of these quantities have been developed.

1.3.2 Matrix Factorization

Each iteration of the algorithm requires inversion of the projected Hessian of the objective function. The $O(n^3)$ computation involved in the process can render the algorithm ineffective for interactive applications. This situation is avoided by maintaining *Cholesky Factorization* of the matrix. When such a factorization is available, inversion becomes an $O(n^2)$ process. The matrix in question changes with every iteration and recalculation of the factorization from scratch would again require $O(n^3)$ operations. However, the factorization can be updated in $O(n^2)$ operations using a class of *quasi-Newton* updates called BFGS updates [Bro88], [Fle70], [Gol70], [Sha70]. These updates utilise the old factors and curvature information obtained by the move along the search direction to calculate the new factors.

The implementation of the algorithm also requires the set of vectors which form the basis for the *Null Space* and the *Range Space* of the constraint gradient matrix. These

[2]See Appendix A for definitions.

are computed by maintaining *Orthogonal Factorization* of the constraint gradient matrix. The constraint gradient matrix changes during the course of minimization. But, efficient techniques exist to update the factors and other associated matrices. These techniques are employed in this research to achieve interactive performance.

1.3.3 Treatment of Constraints

Different types of constraints are treated separately so as to allow considerable reduction in the computation. For instance, the gradients of linear constraints are constant vectors while those of nonlinear constraints change with each iteration. Thus. only the nonlinear constraint gradients need to be evaluated in every iteration. The separate treatment of constraints also allows efficient calculation and modification of the matrix factorization mentioned above.

1.4 Organization

The rest of this thesis provides details of our approach to the direct manipulation of articulated figures. Chapter 2 presents an extensive discussion of previous attempts in the field of computer animation, and identifies their drawbacks. Chapter 3 formally casts the inverse kinematics problem into a nonlinear, constrained minimization problem. It also describes the simplifications that result when a planar, tree structured figure is considered and presents algorithms for efficient evaluation of the objective function, its gradient, its Hessian, non-linear constraints and their gradients. Chapter 4 gives a brief survey of relevant nonlinear optimization methods and details of the algorithm used for this research. Implementation of the techniques for animating planar, articulated figures and their applicability to 3-D figures is discussed in Chapter 5. Conclusions and possible directions for future work are presented in Chapter 6.

Chapter 2

Related Work

Computer graphics research of the past two decades in the area of motion control for animation in 3-D can broadly be placed in three categories - *forward kinematics*, *inverse kinematics* and *dynamics.*

2.1 Forward Kinematics

The first approach to animation using forward kinematics involves creation of keyframes by explicitly setting the joint angles to obtain the desired configuration of the object or the figure to be animated. The computer is then used to create intermediate frames by interpolating the motion parameters. The figure is animated by displaying these frames.

Earlier systems used linear interpolation of motion parameters such as joint coordinates, joint angles etc. to generate intermediate frames. This approach, however, often resulted in unnatural and jerky motion and distortion of the figure [SGWM93]. In an attempt to create smooth movements, methods have appeared which use splines to interpolate various motion parameters [Ste83], [Stu84], [KB84]. The use of quaternions for specifying rigid body orientations has also been suggested [Sho85]. This technique removes some of the artifacts introduced by the use of splines for interpolation. These systems provide the animator with complete control over the movement to the animator.

Well understood techniques based on traditional animation [Las87] exist to impart expressive quality to the animation. Some beautiful animations, for example *Luxo Jr.* [Pix86], *Tony de Peltrie* etc. have been made using such systems. In the case of articulated figures, as the degrees of freedom (DOF) increase, this approach becomes very time consuming and error prone. For instance, the human skeleton can have in excess of two hundred DOF. It is a very difficult task to create keyframes for such a complex figure merely by setting the joint angles. For placing the end of a limb at a desired location, one needs to manipulate joint angles starting from the root to the end of the limb. This can require multiple iterations by trial and error to obtain the correct configuration.

2.2 Inverse Kinematics

The second approach employing inverse kinematics attempts to reduce the animator's tedium by providing means for keyframe creation without the explicit setting of joint angles. Typically, the user is allowed to interactively drag the end of a limb to the desired location while the computer calculates the joint angles required to achieve the goal. This approach has the advantage of complete control over the movement, as accorded by systems employing forward kinematics but without the drudgery involved in the creation of keyframes. This approach has also been extensively studied [GM85], [ZB90], [Sur92a], [Wel93]. Since inverse kinematics is the focus of this dissertation, a more detailed analysis of the related work follows.

2.2.1 Jacobian Based Control

Girard and Maciejewski [GM85] describe a system for animating legged figures which incorporates inverse kinematic positioning and a simple model of dynamic behavior of the body to provide realistic animations of a walk. They solve the problem of inverse kinematics by linearizing the set of forward kinematic equations (see Equation 1.1) of the end effector about the current operating point:

$$\text{if}\,\mathbf{X} = \mathbf{f}(\boldsymbol{\theta})$$

$$\begin{aligned} \text{then } \Delta\mathbf{X} &= \nabla\mathbf{f}(\boldsymbol{\theta})\Delta\boldsymbol{\theta} \\ \text{or} \Delta\mathbf{X} &= \mathbf{J}(\boldsymbol{\theta})\Delta\boldsymbol{\theta} \end{aligned} \tag{2.1}$$

where $\mathbf{J}(\boldsymbol{\theta})$ is called the Jacobian of $\mathbf{f}(\boldsymbol{\theta})$.

The linear system of simultaneous equations, given by Equation 2.1, is solved to find incremental changes in the joint angles of the kinematic chain for the incremental changes in the position of the end effector. Joint angles are integrated over time to find the new state of the chain. If the Jacobian matrix is non-square, a generalized inverse called the pseudoinverse is employed which gives the least-square minimum norm solution to Equation 2.1. An additional term can be incorporated in Equation 2.1 which minimizes a certain criterion to provide a unique, desirable configuration for the kinematic chain, from a set of infinitely many possible configurations. The step trajectory is specified using a Catmull-Rom spline which interpolates the end effector positions. The pseudoinverse, which needs to be updated for every time step, involves $O(n^3)$ computation, making it unsuitable for interactive manipulation of complex figures. Moreover, the highly non-linear nature of equations, which generally occur in this kind of applications, requires a small time step for the relationship of Equation 2.1 to remain valid. The time required for computation is inversely proportional to the time step size. Most importantly, this system does not incorporate constraints on the figure or the joint angles and hence cannot prevent generation of physically unrealizable postures.

2.2.2 Optimization Based Methods

Optimization techniques provide an elegant framework for tackling the inverse kinematics problem in the presence of constraints on the figure. Badler et al [ZB90], [PZB90], [BPW93] present one such algorithm for manipulating articulated figures. A potential function, which is a function of the joint angles in the kinematic chain being manipulated, is associated with each end effector. The inverse kinematic problem is formulated as the minimization of the weighted sum of all the potential functions, subject to linear constraints on the joint angles. The optimization algorithm used is Rosen's projection method with BFGS updates. This algorithm belongs to the class of quasi-Newton methods where, instead of calculating the exact second order information about the objective function, an approximation to it is calculated in every step. This technique, in addition to providing super-linear convergence towards the solution, helps eliminate expensive calculation of the Hessian. In particular, the algorithm used by Zhao [ZB90] starts with an identity matrix as an approximation to the inverse of the Hessian of the objective function. This identity matrix is updated in every iteration such that it approximates the actual inverse of the Hessian. They demonstrate very convincingly the applicability of optimization techniques for the figure manipulation. They present a variety of potential functions which can accomplish tasks like placing the end effector at a point in space, making it point in a desired direction, forcing it to move along a line or a plane etc.

In spite of its successful application to spatial tasks, this approach has some drawbacks. First, for optimization to proceed in the correct direction, the Hessian or its approximation must be positive definite. The (Badler et al) approach of maintaining an approximation to the inverse Hessian is known to lose the important property of positive definiteness of the Hessian due to numerical roundoff errors, so that the search direction calculated is not guaranteed to produce any decrease in the objective function. Thus this method is not very robust. Secondly, the identity matrix that is taken as an approximation to the inverse at the beginning of each optimization cycle does not bear any resemblance to the actual inverse. This in most cases makes the algorithm expend extra effort in finding the solution. Finally, since they do not support nonlinear constraints, tasks like pinning an end effector to a point in space cannot be accomplished. Such situations often arise in pose creation when the end effector is at the desired location but internal joints are not. The easiest way to correct the pose is to immobilize the end effector at its current position and then move intermediate joints.

Another very recent and relevant work is that of Surles [Sur92a], [Sur92b] who presents a physically-based modeling system for large proteins where the user can interactively manipulate protein chains. A constrained optimization algorithm is used to arrive at physically valid configurations of the protein chains. The physical constants like bond lengths are modeled as nonlinear equality constraints while comparatively weaker forces like near neighbor interactions are modeled as potential energy which needs to be minimized. The 3-D coordinates of the atoms or nodes in the protein chains are taken as variables for optimization. The problem is characterized by a nonlinear, twice differentiable objective function and nonlinear equality constraints. Surles uses an optimization method which combines first order Lagrange multiplier estimation techniques with the steepest descent method for the calculation of the search direction. Since the constraints in this problem refer to a fixed number of optimization variables and the number of constraints in a user session remains fixed, a one time preprocessing of the constraint Jacobian is done to obtain a band diagonal matrix whose structure remains fixed throughout the user session. This enables each iteration of the optimization algorithm to have linear computational complexity. Close to real-time performance on a higher end workstation is shown even for large proteins.

Though the formulation of the problem used in Surles' work is very appropriate for the intended application, it is not very suitable for figure manipulation. Using 3-D coordinates as the variables for optimization requires one non-linear constraint to be introduced for each link in the figure. Moreover, to support constraints like bounds on the joint angles, inequality constraints would be required. In the presence of inequality constraints, the constraint Jacobian will change whenever an inequality constraint changes state from active to inactive and vice versa. This will substantially degrade the performance of the algorithm making it unsuitable for interactive applications. Moreover, the algorithm cannot have better than linear convergence rate because of the first order characterization of the objective function and constraints.

2.2.3 Heuristics Based Control

There have been attempts to solve the inverse kinematics problems using heuristics. Two such algorithms are reported by Welman [Wel93]. One method that he puts forward is similar to that of Girard and Maciejewski [GM85]. After making some simplifying assumptions, he arrives at Equation 2.2 as a solution for Equation 2.1.

$$\Delta\boldsymbol{\theta} = \mathbf{J}^T(\boldsymbol{\theta})\Delta\mathbf{X} \tag{2.2}$$

The control accorded is fast and intuitive and the algorithm has been used as a basis for a powerful animation system called *Lifeforms*. The system also allows specification of simple geometric constraints on the figure. The second algorithm works by traversing the kinematic chain from the end effector towards the root and minimizing the orientational and spatial difference between the end effector and the goal at each joint.

Though Welman's system is effective in its application, some drawbacks remain. The heuristic nature of the first algorithm can lead to unexpected problems while manipulating the figure. For instance, Equation 2.2 suggests that the algorithm will provide a vector $\Delta\boldsymbol{\theta}$ for any vector $\Delta\mathbf{X}$. The system is also prone to singularities in the Jacobian matrix which leads to large joint angle velocities. Moreover, the joint angle vector is first calculated without considering the constraints and then any joint angle value which violates associated constraints is truncated such that the constraints are satisfied. This is an ad hoc way of enforcing constraints and introduces unquantifiable errors in the solution. In a similar manner cases can easily be devised where the second algorithm would fail to work. However, Welman demonstrates the efficacy of even approximate inverse kinematic techniques for creating keyframes. His work and the understanding of the problems involved in inverse kinematic control generated, have provided a major impetus for this research.

2.3 Dynamics

The third approach to creating animations is based on a dynamic model of the object. While kinematics concerns itself with positions, velocities and accelerations of different parts of the object or the figure, the focus of dynamics is on the forces required to cause movement. Given applied forces, the problem of calculating accelerations in the various parts of the figure is called *forward dynamics* while the reverse problem is known as *inverse dynamics.* The model is used to simulate behavior that obeys the physical laws that govern motion in real life. A large body of research in the past decade has been directed towards solving the forward/inverse dynamics problem to produce realistic animations of a variety of objects, including simple articulated figures. Though these systems produce realistic animations, they are mostly restricted to animating simple geometric objects or articulated figures with very few degrees of freedom because of the complexity of the system of equations that needs to be solved to produce dynamically correct behavior. Another drawback is that the control provided to the animator is in terms of the parameters of the dynamic systems. Such control is not very intuitive nor is it easy to produce the desired result in the movement. It is fair to say that some attempts, e.g. [IC87], [BB88], [WK88], [Coh92] have been made to rectify this drawback. Nevertheless, problems still remain which have inhibited the use of these systems for creating general animations of complex figures. Some of the works reported in this area are discussed below.

2.3.1 Control Using Kinematic Constraints

Amongst the earliest attempts to produce dynamically correct movement, without relinquishing complete control are those of Isaacs and Cohen [IC87] and Barzel and Barr [BB88]. They maintain kinematic constraints on the figure/object being simulated by calculating the forces that will be required to keep the constraints satisfied. These constraining forces are then introduced into the simulation to exactly cancel out the components of applied forces, which would otherwise work against the satisfaction of the constraints.

Barzel and Barr [BB88] describe a physically-based modeling system with libraries for primitive bodies such as rods, spheres etc., external forces such as gravity, springs etc. and geometric constraints such as point-to-point constraints and point-to-path constraints. Composite objects are created by specifying component objects which are kept together by specifying constraints. Physically valid simulations are created by specifying external forces acting on the composite object. The building block approach to object modeling facilitates creation of new models and movements quickly.

Isaacs and Cohen [IC87] present a system for dynamic simulation of linked figures which allows specification of kinematic constraints and behavior functions to provide limited control over the movement of the figure. Simultaneous equations of motion are solved to obtain incremental accelerations of DOF, which are integrated in time to obtain the new state. Kinematic constraints help reduce the number of variables in the set of simultaneous equations. However, the computational effort expended is inversely proportional to the time step in the integration process and is exponentially related to the number of DOF. This severely limits the use of their method in interactive applications, even though it introduces a very powerful concept of providing control over the movement in a dynamical setting.

Among the attempts which used optimization techniques to produce dynamically correct behavior, especially noteworthy is the one by Witkin and Kass [WK88]. They introduce the concept of space-time constraints as a higher level control mechanism for influencing the animation of articulated figures. They optimize a user-specified criterion subject to nonlinear equality constraints which can be either kinematic or derived from Newtonian mechanics. The algorithm used for optimization is called Sequential Quadratic Programming (SQP). In addition, they implement a symbolic algebra system which is responsible for symbolically differentiating various functions and setting up modules for evaluating finite difference approximations to the derivatives. These modules can be linked with the program dynamically. They show an impressive animation of 'Luxo Jr.' jumping and skiing, and displaying effects like squash and stretch, anticipation etc.

Since optimization takes place over the entire stretch of animation, the nonlinear system which needs to be solved gets very large even for a relatively simple and short animation sequence. Though the system facilitates creation of complete animation sequences which are visually appealing and dynamically correct, the complexity of the system that needs to be solved inhibits its use for interactive applications or for complex articulated figures. Moreover, the control accorded to the animator, though high level, is not necessarily very intuitive. For instance, varying a spring factor for a muscle need not produce the desired effect in the animation. An optimization criterion must be selected depending upon the goal of the animation. It is not clear what kind of criterion will produce what kind of an effect. Finally, the system requires the user to provide an accurate dynamical model of the object to be animated.

Brotman and Netravali [BN88] adapt techniques from optimal control theory to create natural looking animations of rigid objects. They solve a set of differential equations arising out of the dynamics of the object to make it move. A user can exercise control over the movement by specifying keyframes which impose kinematic constraints over the movement. Smooth control forces are introduced which make the movement satisfy the constraints. Overall control energy is minimized using algorithms from optimal control. However, their approach is applicable to rigid objects with a few degrees of freedom and is not suitable for complex articulated figures. Moreover, the user is expected to provide the models of dynamic behavior of the objects.

Cohen [Coh92] further extends the ideas of space-time constraints [WK88] and optimal control [BN88] to provide support for animating simple articulated figures. He uses the divide and conquer approach to reduce to some extent the complexity inherent in earlier systems by subdividing the space-time into discrete pieces called space-time windows. Each of these windows poses an independent subproblem to the numerical optimization process. The subproblems are solved to obtain sub-sequences of animation. The sub-sequences are smoothly interpolated from window to window to get an integrated animation. The graphic interface to the system facilitates control over the animation by allowing interactive specification of constraints. The user is also given the ability to monitor the progress of the optimization process and guide it if it gets stuck in local minima. The higher order information about the objective function and the constraints is obtained by symbolic computation. However, as is the case with other systems based on the models of dynamic behavior of the objects, the system is quite slow. Even simple animation sequences of figures with a very few DOF take up to several seconds.

2.3.2 Dynamic Programming Based Methods

Girard [Gir91] describes a dynamic programming approach to producing expressive movement of legged figures. This approach embeds resolved motion rate control

[GM85] into a broader system which attempts to produce movements where certain criteria such as jerk in the end effector or energy expended in the movement are minimized. Both kinematics and dynamics based variables are optimized subject to constraints. A recursive Newton-Euler formulation and the concept of gradual refinement are employed to mitigate the complexity of the algorithm.

2.3.3 Control Schemes Derived from Biomechanical Models

Lee et al [LWZB90] extend the earlier work of Zhao and Badler [ZB90] by incorporating a variation of the inverse dynamics algorithm to calculate trajectories for the end effector for tasks such as lifting a load. Heuristics, such as strength, comfort, perceived exertion etc., which are derived from anthropometric data, are used as constraints in the selection of appropriate trajectories. The algorithm is shown to be suitable for task level control for a limited set of tasks.

Another step in the same direction is by Phillips and Badler [PB91] who use kinematic constraints to model behavioral tendencies of humans and provide interactive control over the figure's balance and stability. This work could be grouped along with that of [TBM+88], [BC89], [RG91] as an attempt to provide human figure animation based upon a biomechanical model of the human body. By definition, such systems are very specialized in nature and cannot be used for creating general animations.

2.4 Summary

Animating an articulated figure like a human skeleton and modeling physically correct movement patterns are complex tasks. Animation of a body requires coordinated movement of its various parts, for example arms, legs, torso etc. A simple forward kinematics based system is inadequate for animating complex articulated figures.

We see from the discussion in the previous sections that many researchers have attempted to create realistic animations by simulating the dynamics of the motion. In spite of providing a powerful concept, they are severely restricted in their applicability by their inability to handle all but very simple figures or by the lack of interactive performance. Moreover, most of these systems are very specific in terms of objects that they can animate. Those which are not require the user to provide a model of the dynamic behavior of the figure to be animated. Certainly an animator, though skilled in producing expressive movements, is neither equipped nor expected to understand the myriad details of the mechanics of complex objects. This further inhibits the usability of these systems. The movement produced in these systems can be physically correct. However, it falls short of the requirements for sophisticated animations. These systems make it difficult for the animator to impart expressive quality to the animation because of the limited control that they provide for influencing the animation. Moreover, in the case of articulated figures, where the object of animation is a living being, factors like comfort, objective, culture, etc. play an important role in determining the resulting motion. A system based only on a dynamic model cannot account for all such effects that make a movement look realistic.

We believe that it is not only of paramount importance to leave complete control of the movement with the animator, it also makes pragmatic sense to do so considering the scant existing knowledge of what goes into producing a realistic movement. It will be quite some time before human body movement is completely understood and efficient techniques are developed which can handle such large scale dynamics based problems. To date, inverse kinematics techniques, though leaving the onus of imparting realism to the animation with the animator, have been most successful in providing fast and intuitive means for manipulating articulated figures. Besides, since the dynamics based methods cannot address all the issues involved in the movement of humans/living beings, it appears reasonable to leave the animator in complete control of the not so well understood movement design process, while providing him/her with a fast prototyping system by solving a simpler problem. This is the rationale for this research. The next few sections detail the techniques which are proposed to achieve this end.

Chapter 3

Articulated Figures and Inverse Kinematics

An articulated figure, such as a human skeleton, is a tree structured hierarchy of rigid segments connected by joints. By and large such figures have revolute joints. The joints can be simple, permitting rotation in a plane only, as for instance in a human *knee*. They can be complex, making two or three degrees of freedom possible, as in an *elbow* or a *shoulder* joint. However, complex joints can always be approximated by a succession of simple joints providing rotation in different planes.

The figures considered in this research are planar and are made of rigid segments connected at simple revolute joints, each of which allows rotation in only one plane. One joint in the figure is designated as the *root* (see Figure 3.1). Other segments in the figure move with respect to the root. The root can be used to translate the whole figure without changing the relative position of other parts of the figure with respect to the root. Any point on the figure used for manipulation is referred to as the *end effector*. The set of segments and joints from the root to the end effector constitutes a em kinematic chain. I refer to a kinematic chain and all the branches originating from it as a *kinematic subtree.* A figure can have multiple kinematic subtrees. Manipulation of an end effector affects only its own subtree.

For the purpose of kinematic analysis, only a simple kinematic chain, extending from the root to the end effector will be considered. For an n-segment chain, the

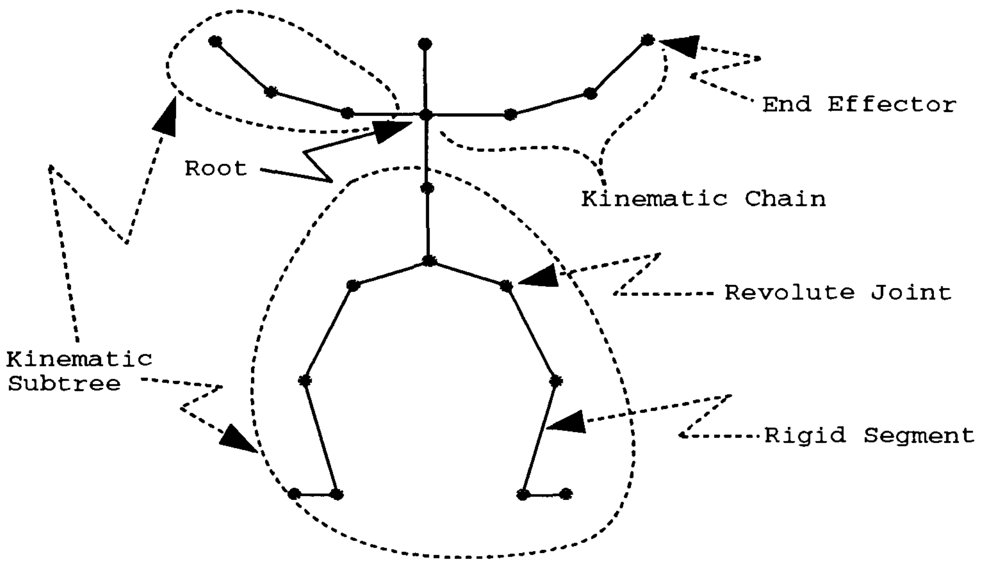

Figure 3.1: Articulated Figure

segments and joints are numbered 1 to n starting from the proximal and going to the distal. The i^{th} joint is the joint connecting the i^{th} and $(i+1)^{th}$ segments. By convention, the root is designated as joint 0 and is assumed to be the joint between a zero length segment and segment one. The end effector is referred to as the joint n.

A *coordinate frame* is a set of three concurrent, orthonormal vectors called *coordinate axes*. The point of concurrence is called the *origin* of the coordinate frame. Any point in 3-D space can be represented as a unique triplet of projections made by the line joining the origin and the point with each of the coordinate axes. This triplet is termed the *coordinate vector* or *coordinates* of the point with respect to the coordinate frame. *Homogeneous* coordinates of a point are the coordinates augmented with a fourth component. Homogeneous coordinates allow a unified treatment of rotations and translations when applied to a vector.

The root is assumed to be placed at the origin of a coordinate frame, called the *base frame* or coordinate frame 0. One coordinate frame is rigidly attached to each of

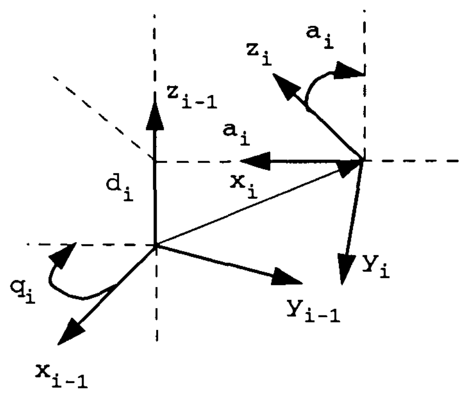

Figure 3.2: Coordinate Frames Using Denavit-Hartenberg Convention

the segments such that the coordinates of the i^{th} segment and the i^{th} joint never vary with respect to the i^{th} coordinate frame. Homogeneous coordinates of points given in the i^{th} coordinate frame can be converted into coordinates in the $(i-1)^{th}$ coordinate frame by a transformation matrix M_i.

3.1 The Denavit-Hartenberg Convention

The Denavit-Hartenberg [DH55], [SV89] convention establishes a framework for systematic specification of coordinate axes for different segments in the chain. The convention uniquely specifies the coordinate axes for each degree of freedom of the articulated figure. The axes obey the right hand rule. So, specification of any two axes results in a unique specification for the third axis. The axes for each segment are fixed such that the axis x_i is perpendicular to and intersect the axis z_{i-1}.

For a three-dimensional figure, the homogeneous transformation M_i between the i^{th} and $(i-1)^{th}$ coordinate frames is characterized by the following four parameters (see Figure 3.2):

1. length a_i of the i^{th} segment: It is the shortest distance between the axes z_i and z_{i-1} measured in the direction of x_i. The corresponding translation matrix is

given by T_{x,a_i}, where:

$$\mathrm{T}_{x,a_i} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. segment twist α_i: It is measured as the angle between the axes z_i and z_{i-1}, in the plane normal to x_i. The corresponding rotation matrix R_{x,α_i} is given as follows:

$$\mathrm{R}_{x,\alpha_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. distance d_i between the segments: It is the distance between the origin of the frame $i-1$ and the intersection of the axes x_i and z_{i-1}. The positive direction is the direction of z_{i-1}. The translation matrix T_{z,d_i} is given by:

$$\mathrm{T}_{z,d_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4. angle θ_i between the segments: It is the angle between x_i and x_{i-1} axes, measured in the plane normal to z_{i-1} axis. The corresponding rotation matrix R_{z,θ_i} is:

$$\mathrm{R}_{z,\theta_i} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The composite transformation matrix M_i is a product of the rotation and translation matrices described above:

$$\mathrm{M}_i = \mathrm{R}_{z,\theta_i} \cdot \mathrm{T}_{z,d_i} \cdot \mathrm{T}_{x,a_i} \cdot \mathrm{R}_{x,\alpha_i} \tag{3.1}$$

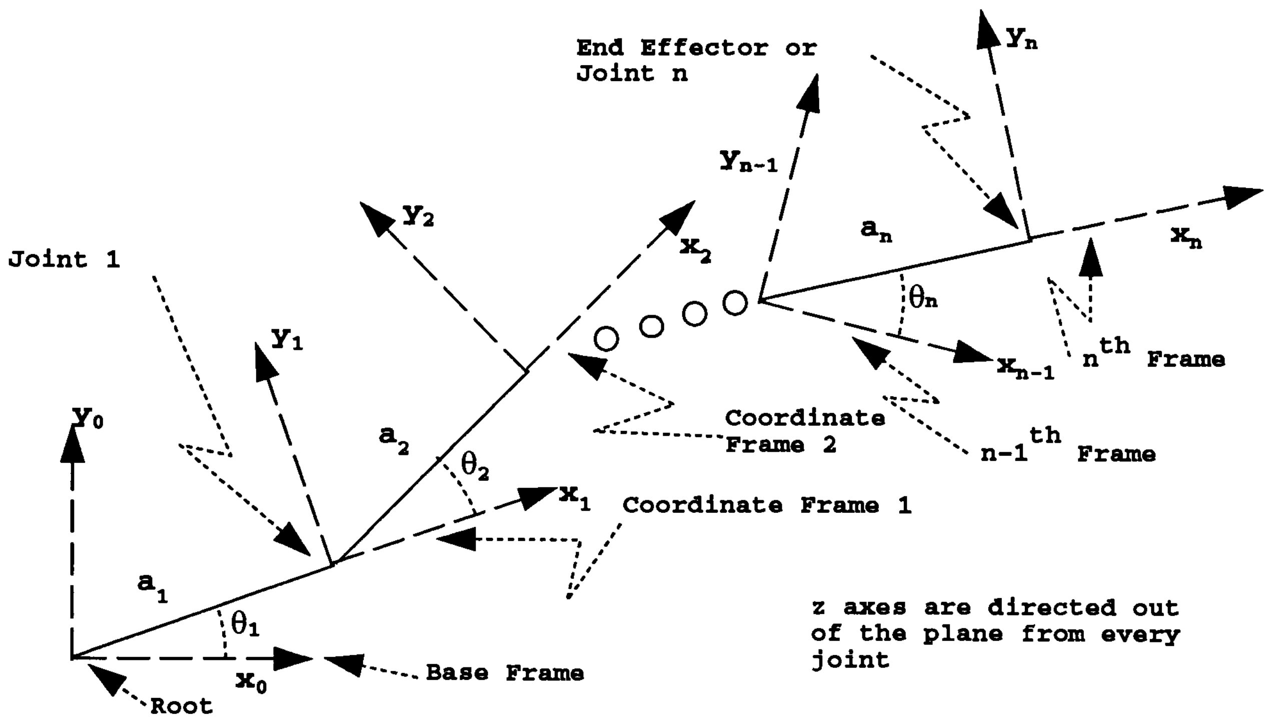

Figure 3.3: Kinematic Chain In xy Plane

Simplifications result for a planar kinematic chain. As shown in Figure 3.3, joint i becomes the origin of the i^{th} coordinate frame. All z axes are parallel to each other and the axis x_i is directed along the i^{th} segment, away from the root. The intersection of the x_i and z_{i-1} axes is also the origin of the $(i-1)^{th}$ coordinate frame. Thus, quantities α_i and d_i are zero such that R_{x,α_i} and T_{z,d_i} become identity matrices. θ_i is the only variable for all $i \in (1, 2, ..., n)$. The matrix M_i reduces to the form:

$$\mathrm{M}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i & 0 & a_i\sin\theta_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

The transformation matrix $\mathrm{M}_{n,0}$ required to convert coordinates of points in the n^{th} coordinate frame into coordinates of the base frame is given by cascaded multiplication of transformation matrices from the root to the end effector:

$$\begin{aligned} \mathrm{M}_{n,0} &= \mathrm{M}_1 \cdot \mathrm{M}_2 \cdots \mathrm{M}_n \qquad (3.3) \\ \mathrm{M}_{n,0} &= \begin{bmatrix} \cos\left(\sum_{i=1}^{n}\theta_i\right) & -\sin\left(\sum_{i=1}^{n}\theta_i\right) & 0 & \sum_{i=1}^{n}\left(a_i\cos\left(\sum_{j=1}^{i}\theta_j\right)\right) \\ \sin\left(\sum_{i=1}^{n}\theta_i\right) & \cos\left(\sum_{i=1}^{n}\theta_i\right) & 0 & \sum_{i=1}^{n}\left(a_i\sin\left(\sum_{j=1}^{i}\theta_j\right)\right) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

The last column of $\mathrm{M}_{n,0}$ gives the homogeneous coordinates of the origin of the n^{th} frame, which coincides with the end effector, with respect to the base frame. Thus for a planar chain, the x and y coordinates of the end effector, which are the first two elements of the last column of $\mathrm{M}_{n,0}$, are given by Equations 3.4 and 3.5.

$$x_E = \sum_{i=1}^{n}\left(a_i\cos\left(\sum_{j=1}^{i}\theta_j\right)\right) \tag{3.4}$$

$$y_E = \sum_{i=1}^{n}\left(a_i\sin\left(\sum_{j=1}^{i}\theta_j\right)\right) \tag{3.5}$$

3.2 Inverse Kinematics

For an n-segment kinematic chain, the coordinates of the end effector with respect to the base frame are completely determined by the joint angles from the root to the end effector (see Equations 3.4 and 3.5). Thus, the relationship between a vector

$\mathbf{X} = [x_1, x_2, ..., x_m]^T$ of end effector coordinates ($m = 2$ for planar case) and a vector of joint angle variables $\boldsymbol{\theta} = [\theta_1, \theta_2, ..., \theta_n]^T$ can be represented by Equation 3.6.

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} f_1(\theta_1, \theta_2, ..., \theta_n) \\ f_2(\theta_1, \theta_2, ..., \theta_n) \\ \vdots \\ f_m(\theta_1, \theta_2, ..., \theta_n) \end{bmatrix}$$

$$\mathbf{X} = \mathbf{f}(\boldsymbol{\theta}) \tag{3.6}$$

The calculation of the position of a point on a kinematic chain for a given joint angle vector is called the *forward kinematic* problem. Equation 3.6 has a unique solution for any vector $\boldsymbol{\theta}$. The chain can be brought to any desired configuration by specifying appropriate values for joint angles.

The complementary problem of determining the joint angle vector $\boldsymbol{\theta}$ for a given position vector $\mathbf{X}$ of the end effector is an *inverse kinematic* problem. To move the end effector to a desired location, joint angles need to be determined so that the goal is achieved. This requires finding the inverse of Equation 3.6, which is a set of non-linear equations. The simultaneous solution required is not easy to find, and there may not be any solution. Even if a solution exists, it is not guaranteed to be unique. The same end effector position can be achieved by different sets of joint angles (see Figure 3.4). No closed form solution exists for an arbitrary chain.

The inverse kinematic problem of finding the joint angle vector $\boldsymbol{\theta}_P$, which will make the end effector position vector $\mathbf{X} = \mathbf{X}_P$, where $\mathbf{X}_P$ is the vector of coordinates of any point P, can also be stated as a problem of minimizing the *Euclidean* distance between $\mathbf{X}$ and $\mathbf{X}_P$.

3.3 Constrained Minimization

The Euclidean distance between the current position $\mathbf{X}$ of the end effector and the desired position $\mathbf{X}_P$ is given by $\|\mathbf{X} - \mathbf{X}_P\|_2$. Any suitable function of the Euclidean distance can be taken as an objective function for the minimization process. The

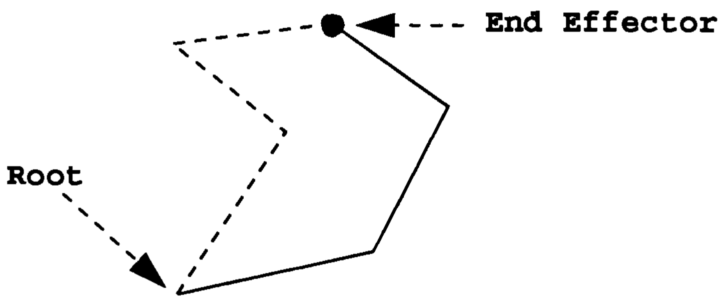

Figure 3.4: Redundant Kinematic Chain

square of the Euclidean distance $\mathcal{F}(\boldsymbol{\theta})$ has been used as the objective function in this research (see Equation 3.7) as it leads to simpler equations.

$$\mathcal{F}(\boldsymbol{\theta}) = \|\mathbf{X} - \mathbf{X}_P\|_2{}^2 \tag{3.7}$$

Indeed, as shown by Zhao and Badler [ZB90] various other kinds of objective functions can be formulated to accomplish different spatial positioning tasks.

A kinematic chain with more DOF than required for a spatial positioning task is called a *redundant* chain. For a kinematically redundant chain, many different configurations are possible to accomplish a spatial positioning goal. Thus constraints need to be incorporated in the optimization procedure so that only a desirable configuration is produced. For instance, the range of operation of a particular joint can be limited by imposing simple bounds (e.g. $l_{Bi} \leq \theta_i \leq u_{Bi}$ where l_{Bi} and u_{Bi} are the lower and upper limits respectively). Linear equality constraints ($l_{Bi} = u_{Bi}$) can be used to fix a joint angle to a specific value.

Functional dependencies between joint angles can be linear or nonlinear in nature. A linear relation between joint angles can be represented by a general linear constraint of type $l_{Li} \leq \mathbf{a}_{Li}^T \cdot \boldsymbol{\theta} \leq u_{Li}$, where $\mathbf{a}_{Li}$ is an n-vector of coefficients of the joint angles for the i^{th} general linear constraint and l_{Li} and u_{Li} are the lower and upper limits respectively. A nonlinear functional dependency can be represented by a constraint of the form $l_{Ni} \leq c_i(\boldsymbol{\theta}) \leq u_{Ni}$, where c_i is the i^{th} nonlinear constraint and l_{Ni} and u_{Ni}

are the associated lower and upper limits respectively.

During the process of keyframe creation, it often happens that the end effector is at the desired location but the pose is not correct. The easiest way to correct the pose is to pin the end effector at its current position and manipulate an intermediate joint. This is typically done by introducing nonlinear equality constraints of the form $l_{Ni} \leq c_i(\boldsymbol{\theta}) \leq u_{Ni}$, where $l_{Ni} = u_{Ni} = 0$. If $\mathbf{X} = [x_1, x_2, ..., x_m]^T$ is the vector of the coordinates of the point which is constrained to be at a point P in space, and $\mathbf{X}_P = [x_{P1}, x_{P2}, ..., x_{Pm}]^T$ is the vector of the coordinates of the point P, the m nonlinear constraints to accomplish the goal can be given as follows:

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} - \begin{bmatrix} x_{P1} \\ x_{P2} \\ \vdots \\ x_{Pm} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.8}$$

The optimization problem can then be stated formally as:

$$\underset{\boldsymbol{\theta} \in \mathcal{R}^n}{\text{minimize}} \quad \mathcal{F}(\boldsymbol{\theta}) \tag{3.9}$$

$$\text{subject to} \quad \begin{array}{ccccc} \mathbf{l}_B & \leq & \boldsymbol{\theta} & \leq & \mathbf{u}_B \\ \mathbf{l}_L & \leq & \mathbf{A} \cdot \boldsymbol{\theta} & \leq & \mathbf{u}_L \\ \mathbf{l}_N & \leq & \mathbf{c}(\boldsymbol{\theta}) & \leq & \mathbf{u}_N \end{array} \tag{3.10}$$

The minimization process, as is detailed in Chapter 4, requires evaluation of the objective function, its gradient and its Hessian. The gradients of the nonlinear constraint also need to be determined. The next section introduces a notation and gives equations for various quantities mentioned above.

3.4 The Definitions and Results

Let P be the point where the end effector is to be placed. For the planar case, the coordinates of P are $\mathbf{X}_P = [x_P, y_P]^T$. Substituting $\mathbf{X}_P$ and Equations 3.4 and 3.5 in Equation 3.7 for an n-segment planar chain and expanding the terms we get:

$$\mathcal{F}(\boldsymbol{\theta}) = \|\mathbf{X} - \mathbf{X}_P\|_2{}^2$$

$$
\begin{aligned}
&= (x_{\text{endeffector}} - x_P)^2 + (y_{\text{endeffector}} - y_P)^2 \\
&= \left(\sum_{i=1}^{n}\left(a_i \cos\left(\sum_{j=1}^{i}\theta_j\right)\right) - x_P\right)^2 + \left(\sum_{i=1}^{n}\left(a_i \sin\left(\sum_{j=1}^{i}\theta_j\right)\right) - y_P\right)^2 \\
&= (a_1\cos(\theta_1) + a_2\cos(\theta_1+\theta_2)\cdots a_n\cos(\theta_1+\theta_2\cdots\theta_n) - x_P)^2 + \\
&\quad (a_1\sin(\theta_1) + a_2\sin(\theta_1+\theta_2)\cdots a_n\sin(\theta_1+\theta_2\cdots\theta_n) - y_P)^2 \qquad (3.11)
\end{aligned}
$$

Similarly, the two nonlinear equality constraints c_x and c_y, required to pin a point on the chain to point P in the plane, are given by Equation 3.12:

$$
\begin{aligned}
c_x &= a_1\cos(\theta_1) + a_2\cos(\theta_1+\theta_2)\cdots a_n\cos(\theta_1+\theta_2\cdots\theta_n) - x_P = 0 \\
c_y &= a_1\sin(\theta_1) + a_2\sin(\theta_1+\theta_2)\cdots a_n\sin(\theta_1+\theta_2\cdots\theta_n) - y_P = 0
\end{aligned}
\qquad (3.12)
$$

The regularity in the structure of Equations 3.11 and 3.12 suggests that there should be some regularity in their derivatives as well. However, this regularity is completely lost and the task of calculating the derivatives becomes very difficult due to proliferation of terms. The notation given below allows concise representation of the objective function, nonlinear constraints and their derivatives.

3.4.1 The Definitions

Definition 1 *SigmaTheta ($\theta_{w,v}$) denotes the angle between the x-axes of the w^{th} and $(v-1)^{th}$ coordinate frames, measured in the plane normal to the z axis of the $(v-1)^{th}$ coordinate frame:*

$$
\theta_{w,v} = \begin{cases} \sum_{i=v}^{w}\theta_i & \text{for } w \geq v \\ 0 & \text{otherwise} \end{cases} \qquad (3.13)
$$

$$
\text{e.g. } \theta_{5,3} = \theta_3 + \theta_4 + \theta_5
$$

Definition 2 *SigmaSin ($S^n_{w,v}$) denotes the distance between the origins of the n^{th} and $(w-1)^{th}$ coordinate frames, measured along the y axis of the $(v-1)^{th}$ coordinate frame:*

$$S^n_{w,v} = \begin{cases} \sum_{i=w}^{n} (a_i \sin(\theta_{i,v})) & \text{for} n \geq w \geq v \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

$$\begin{aligned} \text{e.g. } S^4_{3,1} &= \sum_{i=3}^{4} (a_i \sin(\theta_{i,1})) \\ &= a_3 \sin(\theta_{3,1}) + a_4 \sin(\theta_{4,1}) \\ &= a_3 \sin(\theta_1 + \theta_2 + \theta_3) + a_4 \sin(\theta_1 + \theta_2 + \theta_3 + \theta_4) \end{aligned}$$

Definition 3 *SigmaCos ($C^n_{w,v}$) denotes the distance between the origins of the n^{th} and $(w-1)^{th}$ coordinate frames, measured along the x axis of the $(v-1)^{th}$ coordinate frame:*

$$C^n_{w,v} = \begin{cases} \sum_{i=w}^{n} (a_i \cos(\theta_{i,v})) & \text{for} n \geq w \geq v \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

3.4.2 The Results

The definitions given above can be used to derive the results which are highlighted by the boxes around them. The detailed derivations can be found in Appendix B.

Corollary 1

$$\boxed{\begin{aligned} S^n_{i,1} &= S^n_{n,1} + S^{n-1}_{n-1,1} + \cdots + S^i_{i,1} \\ C^n_{i,1} &= C^n_{n,1} + C^{n-1}_{n-1,1} + \cdots + C^i_{i,1} \end{aligned}} \tag{3.16}$$

Corollary 2

$$\boxed{\begin{aligned} S^n_{i,1} &= S^n_{i+1,1} + S^i_{i,1} \\ C^n_{i,1} &= C^n_{i+1,1} + C^i_{i,1} \end{aligned}} \tag{3.17}$$

Theorem 1 *Partial derivatives of SigmaSin and SigmaCos are given by*

$$\boxed{\begin{aligned} \frac{\partial}{\partial\theta_u}(S^n_{w,v}) &= \begin{cases} C^n_{\max(u,w),v} & \textit{for } n \geq u \geq v \geq 1 \\ 0 & \textit{otherwise} \end{cases} \\ \frac{\partial}{\partial\theta_u}(C^n_{w,v}) &= \begin{cases} -S^n_{\max(u,w),v} & \textit{for } n \geq u \geq v \geq 1 \\ 0 & \textit{otherwise} \end{cases} \end{aligned}} \tag{3.18}$$

It can be shown that using the definitions given above the objective function, its gradient, its Hessian, non-linear constraints and their gradients can be represented as shown in the following sections. For the sake of brevity two additional terms K_x and K_y are defined as follows:

$$K_x = C^n_{1,1} - x_P \tag{3.19}$$

$$K_y = S^n_{1,1} - y_P \tag{3.20}$$

The Objective Function

$$\boxed{\mathcal{F}(\boldsymbol{\theta}) = K_x^2 + K_y^2} \tag{3.21}$$

The Gradient

First partial derivative of $\mathcal{F}(\boldsymbol{\theta})$ with respect to $\theta_i, g_i = \frac{\partial}{\partial\theta_i}(\mathcal{F}(\boldsymbol{\theta}))$ is given by:

$$\boxed{g_i = \frac{\partial}{\partial\theta_i}(\mathcal{F}(\boldsymbol{\theta})) = 2(K_y C^n_{i,1} - K_x S^n_{i,1})} \tag{3.22}$$

The Gradient $\mathbf{g}$ of $\mathcal{F}(\boldsymbol{\theta})$ is given by the following equation:

$$\mathbf{g} = \nabla\mathcal{F}(\boldsymbol{\theta})$$

$$\begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_i \\ \vdots \\ g_n \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial\theta_1}(\mathcal{F}(\boldsymbol{\theta})) \\ \frac{\partial}{\partial\theta_2}(\mathcal{F}(\boldsymbol{\theta})) \\ \vdots \\ \frac{\partial}{\partial\theta_i}(\mathcal{F}(\boldsymbol{\theta})) \\ \vdots \\ \frac{\partial}{\partial\theta_n}(\mathcal{F}(\boldsymbol{\theta})) \end{bmatrix} = \begin{bmatrix} 2(K_yC^n_{1,1} - K_xS^n_{1,1}) \\ 2(K_yC^n_{2,1} - K_xS^n_{2,1}) \\ \vdots \\ 2(K_yC^n_{i,1} - K_xS^n_{i,1}) \\ \vdots \\ 2(K_yC^n_{n,1} - K_xS^n_{n,1}) \end{bmatrix} \tag{3.23}$$

The Hessian

Second partial derivative of $\mathcal{F}(\boldsymbol{\theta})$ with respect to θ_i and θ_j, $h_{i,j}$, where $i \geq j$ is given by:

$$\boxed{h_{i,j} = \frac{\partial^2}{\partial\theta_i\partial\theta_j}(\mathcal{F}(\boldsymbol{\theta})) = 2\Big((C^n_{j,1} - K_x)C^n_{i,1} + ((S^n_{j,1} - K_y)S^n_{i,1}\Big)} \tag{3.24}$$

The Hessian $\mathbf{H}$ of function $\mathcal{F}(\boldsymbol{\theta})$ is an $n \times n$ symmetric matrix of second partial derivatives of $\mathcal{F}(\boldsymbol{\theta})$, such that, if $h_{i,j}$ is the element in the i^{th} row and j^{th} column of $\mathbf{H}$ then $h_{i,j} = h_{j,i}$. To completely determine $\mathbf{H}$ only the lower triangular part of the matrix needs to be determined; i.e., those values of $h_{i,j}$ where i varies from 1 to n and j varies from 1 to i.

The Nonlinear Constraints and Gradients

Using the notation introduced earlier in the chapter, the non-linear constraints c_x and c_y given by Equation 3.12 can be represented by:

$$\boxed{\begin{aligned} c_x &= K_x = 0 \\ c_y &= K_y = 0 \end{aligned}} \tag{3.25}$$

their gradients can be given by:

$$\begin{aligned}\nabla c_x &= -\begin{bmatrix} S^n_{1,1} \\ S^n_{2,1} \\ \vdots \\ S^n_{n,1} \end{bmatrix} \\ \nabla c_y &= \begin{bmatrix} C^n_{1,1} \\ C^n_{2,1} \\ \vdots \\ C^n_{n,1} \end{bmatrix}\end{aligned} \tag{3.26}$$

3.5 Evaluation

In our system an articulated figure is represented as a tree whose i^{th} node contains the length a_i of the i^{th} segment, and the angle θ_i between the x axes of i^{th} and $i-1^{th}$ coordinate frames (see Figure 3.5). Each node has a pointer to its parent node which is returned by the function *parent()*. This allows tree traversal from the end effector to the root. In addition to the fields mentioned above, two more fields, *SigmaCos* and *SigmaSin*, are present in each node. These fields are used for storing the values of $C^i_{i,1} = a_i \cos(\theta_1 + \theta_2 \cdots \theta_i)$ and $S^i_{i,1} = a_i \sin(\theta_1 + \theta_2 \cdots \theta_i)$ respectively. These values are typically updated by the drawing routine which visits each node in the tree during the drawing process.

3.5.1 The Objective Function, the Gradient and the Hessian

Algorithm FGH given below computes the objective function, its gradient and its Hessian by traversing the tree once from the end effector to the root. It takes the following arguments:

real	:	$x_P, y_P, ObjFunc, Grad[1:n], Hess[1:n][1:n]$
integer	:	$mode, n$
treenode	:	$node$

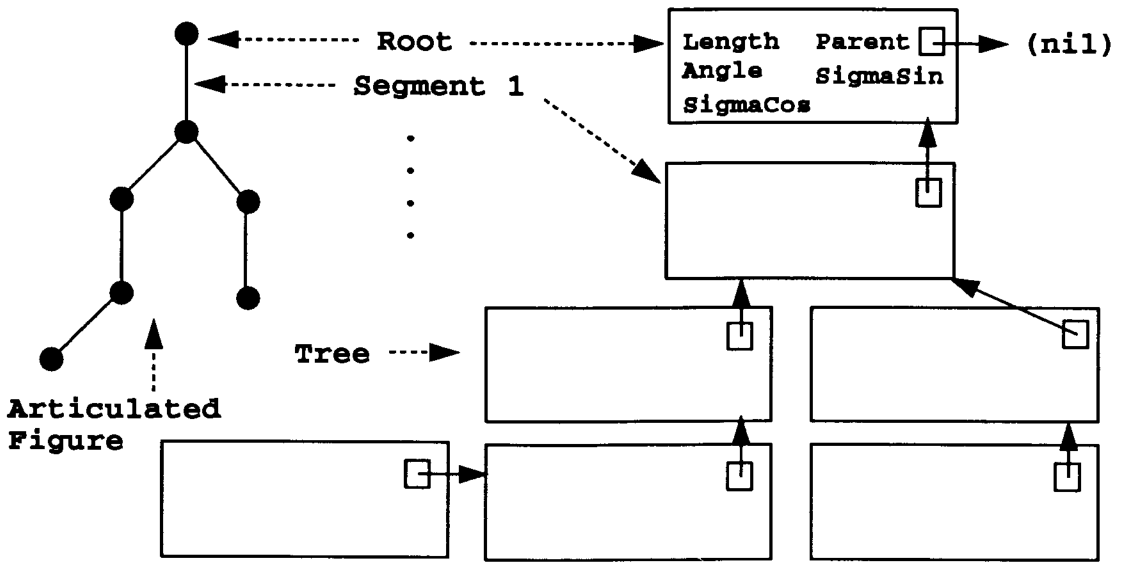

Figure 3.5: Data Structure for an Articulated Figure

x_P, y_P, *mode* and n are the input arguments. x_P and y_P contain the coordinates of the point P. *mode* is a flag for indicating which quantities are to be computed (valid values are *FUNC, GRAD* and *HESS*). If the mode is *FUNC*, only the objective function is calculated. If it is *GRAD*, both the objective function and the gradient are computed. The value *HESS* results in calculation of the objective function, the gradient and the Hessian. n gives the number of segments in the kinematic chain. *ObjFunc, Grad* and *Hess* are used for output of the results of computation. *node* is the pointer of type *treenode* and is used for tree traversal. It is set to *endeffector* in the initialization step (FGH–I). *endeffector* is also of type *treenode* and contains the pointer to the end effector of the kinematic chain for which the calculation is to be carried out.

The following internal variables are defined for storing the intermediate results:

$$\begin{array}{lll} real & : & K_x, K_y, ASigmaCos[1:n], ASigmaSin[1:n] \\ integer & : & depth, j \end{array}$$

K_x and K_y are used for storing the quantities given by Equations 3.19 and 3.20. *ASigmaCos*[1 : n] and *ASigmaSin*[1 : n] store the values $C^n_{i,1}$ and $S^n_{i,1}$ respectively. *depth* indicates the depth of the *current node* in the tree. Both *depth* and j are also used as counters.

The algorithm (*FGH–III*) calculates K_x and K_y using Equation 3.16 and $C^n_{i,1}$ and $S^n_{i,1}$ using Equation 3.17 while traversing the tree from the end effector to the root. The last two quantities are stored in the appropriate elements of arrays *ASigmaCos* and *ASigmaSin* respectively. For the calculation of the Hessian, two additional quantities, $(C^n_{i,1} - K_x)$ and $(S^n_{i,1} - K_y)$ are required. These are calculated once and the appropriate elements of *ASigmaCos* and *ASigmaSin* are overwritten by them (*FGH-VII*), after they are not required any longer.

Algorithm FGH

```
FGH–I Initialization

        $K_x \leftarrow -x_P$
        $K_y \leftarrow -y_P$
        $depth \leftarrow n$
        $node \leftarrow endeffector$

FGH–II Objective Function Pass: if the current node is root, go to FGH–IV

        if $depth = 0$
            $depth \leftarrow 1$
            go to FGH-V
```

FGH–III *Calculate K_x, K_y, SigmaCos ($C^n_{i,1}$) and SigmaSin ($S^n_{i,1}$)*

```
$K_x \leftarrow K_x + node :: SigmaCos$
$K_y \leftarrow K_y + node :: SigmaSin$
if $depth < n$
    $ASigmaCos[depth] \leftarrow ASigmaCos[depth+1] + node :: SigmaCos$
    $ASigmaSin[depth] \leftarrow ASigmaSin[depth+1] + node :: SigmaSin$
else
    $ASigmaCos[depth] \leftarrow node :: SigmaCos$
    $ASigmaSin[depth] \leftarrow node :: SigmaSin$
$node \leftarrow parent(node)$
$depth \leftarrow depth - 1$
go to FGH–II
```

FGH–IV *Calculate Objective Function*

```
    $ObjFunc \leftarrow K_x^2 + K_y^2$
    if $mode$ is $FUNC$    stop
```

FGH–V *Calculate elements of the Gradient*

```
$Grad[depth] \leftarrow 2(K_y \cdot ASigmaCos[depth] - K_x \cdot ASigmaSin[depth])$
```

FGH–VI *Calculate elements of the Hessian:* If the *mode* is not *HESS* then go to *FGH–VIII* else for $j \leftarrow 1$ to *depth* do the following

```
if $j < depth$
    $Hess[depth][j] \leftarrow 2\,(ASigmaCos[j] \cdot ASigmaSin[depth] - ASigmaSin[j] \cdot ASigmaCos[depth])$
    $Hess[j][depth] \leftarrow Hess[depth][j]$
else
    $Hess[depth][depth] \leftarrow 2\,((ASigmaCos[depth] - K_x) \cdot ASigmaSin[depth] - (ASigmaSin[depth] - K_y) \cdot ASigmaCos[depth])$
```

```
FGH–VII Update ASigmaCos and ASigmaSin

        $ASigmaCos[depth] \leftarrow ASigmaCos[depth] - K_x$
        $ASigmaSin[depth] \leftarrow ASigmaSin[depth] - K_y$

FGH–VIII Termination check

        if $depth = n$          then stop
        else
                $depth \leftarrow depth + 1$
                go to FGH–V
```

3.5.2 The Nonlinear Constraints and their Gradients

The nonlinear constraints and their gradients, given by Equations 3.25 and 3.26 respectively, are calculated by traversing the tree from the point to be constrained to the root. To conclude the chapter we present algorithm *CG*, which calculates the two constraint values required to constrain a point in space, and their gradients. The quantities K_x, K_y, $C^n_{i,1}$ and $S^n_{i,1}$ are computed using Equations 3.16 and 3.17, in a manner analogous to the algorithm *FGH*. The algorithm has the following arguments:

real	:	$x_P, y_P, ConstraintX, ConstraintY,$ $ConstrGradX[1:n], ConstrGradY[1:n]$
integer	:	n
treenode	:	*node*

x_P, y_P, *node* and n are the inputs to the algorithm. x_P and y_P are the x and y coordinates of the point P to which the end effector is pinned. *node* is a pointer of type *treenode*. It is set to *endeffector* in the initialization step *CG–I*. *endeffector* is also a pointer of type *treenode* and contains the pointer to the end effector of the kinematic chain for which the computation is to be carried out. n gives the depth of *node*. An internal variable *depth* of type integer is used to store the depth of current node.

ConstraintX, *ConstraintY*, *ConstrGradX* and *ConstrGradY* are the output parameters. *ConstraintX* and *ConstraintY* contain the values K_x and K_y respectively. *ConstrGradX* and *ConstrGradY* are vectors containing the gradients of the two constraints, respectively.

Algorithm CG

```
CG–I Initialization
        $ConstraintX \leftarrow -x_P$
        $ConstraintY \leftarrow -y_P$
        $depth \leftarrow n$
        $node \leftarrow endeffector$

CG–II Termination Check
        $if\ depth = 0\ then\ stop$

CG–III Calculate ConstraintX, ConstraintY, ConstrGradX and ConstrGradY
    $ConstraintX \leftarrow ConstraintX + node :: SigmaCos$
    $ConstraintY \leftarrow ConstraintY + node :: SigmaSin$
    if $depth < n$
        $ConstrGradX[depth] \leftarrow ConstrGradX[depth+1] - node :: SigmaCos$
        $ConstrGradY[depth] \leftarrow ConstrGradY[depth+1] + node :: SigmaSin$
    else
        $ConstrGradX[depth] \leftarrow -node :: SigmaCos$
        $ConstrGradY[depth] \leftarrow node :: SigmaSin$
    $node \leftarrow parent(node)$
    $depth \leftarrow depth - 1$
    go to CG–II
```

Chapter 4

Nonlinear Optimization

The problem of inverse kinematic calculation of the joint angle vector required to place an end effector at a point in space was cast as an optimization problem in the last chapter. This chapter gives a description of the algorithm used for carrying out the optimization. A brief introduction to optimization is given followed by the necessary conditions for optimality. Different classes of algorithms, relevant for solving the type of problem that arises in this research, are discussed. Simplified steps of the specific algorithm used in this research are presented and details of important steps are given.

Most of the discussion in this chapter is adapted from Gill et al [GMW81]. [Fle87] is another important source of information. [Hes75] contains an excellent theoretical discussion on optimization. Details of different aspects of the actual algorithm used can be found in several sources such as [GMSW84], [GMSW85], [GMSW86a], [GMSW86b].

The notation used in this thesis in general, and in this chapter in particular, is recapitulated in Appendix A.

4.1 Introduction to Optimization

An optimization problem is characterized by a vector of independent variables $\boldsymbol{\theta}$, restrictions on acceptable values, called *constraints* ($\mathbf{a}^T\boldsymbol{\theta}$, $\mathbf{c}(\boldsymbol{\theta})$) and a *metric* for comparing one set of values of the independent variable with another, called the *objective*

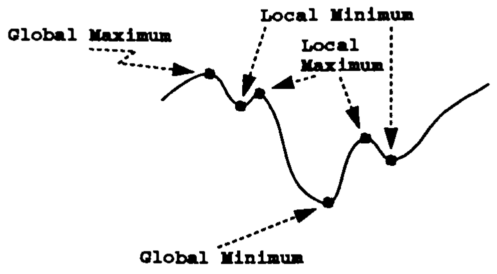

Figure 4.1: Minima and Maxima

function ($\mathcal{F}(\boldsymbol{\theta})$). The theory of optimization concerns itself with making the objective function attain a "good" value, usually a minimum or a maximum. The process is accordingly termed a minimization or a maximization.

A point $\boldsymbol{\theta}^*$ on the graph of the objective function is termed as a *feasible* point if it satisfies all the constraints. Only a feasible point can be optimal. A feasible point $\boldsymbol{\theta}^*$ is called a local minimum (maximum) if the objective function attains a minimum (maximum) value at that point as compared to all other points in the immediate neighborhood of $\boldsymbol{\theta}^*$ (see Figure 4.1). The point $\boldsymbol{\theta}^*$ is a global minimum (maximum) if the function has the lowest (highest) value at $\boldsymbol{\theta}^*$ with respect to all other points. Finding a global minimum is a **hard** problem. Methods like *simulated annealing* [KGV83], can be used to find a global minimum, but are unsuitable for interactive applications because of the enormous computation time involved. The problem addressed in this thesis, however, requires only a strong local minimum, making it possible to use a faster algorithm.

Optimization of general nonlinear functions, subject to general constraints is an open research problem. Most of the popular algorithms in use for optimization impose certain qualifications on the objective functions and the constraints. In particular, both the constraints and the objective function are expected to be at least twice-continuously differentiable. The constraints and the objective functions that arise in this research satisfy these requirements.

4.2 Optimality Conditions

Optimality conditions or *necessary* conditions for a minimum are a set of practical tests that a feasible point must pass for it to be considered as a *minimum.* An inequality constraint of the form $l_{Li} \leq \mathbf{a}_{Li}^T \boldsymbol{\theta}^* \leq u_{Li}$, if present, is termed an *active* constraint if it is exactly satisfied at $\boldsymbol{\theta}^*$, i.e. $\mathbf{a}_{Li}^T \boldsymbol{\theta}^* = l_{Li}$ or $\mathbf{a}_{Li}^T \boldsymbol{\theta}^* = u_{Li}$. If the constraint is satisfied but not at one of the bounds, then the constraint is termed an *inactive* constraint. An equality constraint (where $l_{Li} = u_{Li}$), by definition, is always *active.* Only the constraints active at the optimal point appear in the optimality conditions. However, the optimal point must be feasible with respect to all the constraints.

When specifying optimality conditions, it makes it clearer if only one type of constraint is considered. The necessary conditions for $\boldsymbol{\theta}^*$ to be a minimum of nonlinear programming with only linear constraints and only nonlinear constraints, respectively, are given in the next two subsections [GMW81], [Pow74].

4.2.1 Linear Constraints

It is illustrative to look at the derivation of optimality conditions in detail as they provide the motivation for the various steps of the algorithm described in the later part of this chapter.

A point $\boldsymbol{\theta}^*$ will be a minimum of Equation 4.1 if and only if $\mathcal{F}(\boldsymbol{\theta}^*) \leq \mathcal{F}(\boldsymbol{\theta})$ for every $\boldsymbol{\theta}$ in a *sufficiently* small neighborhood of $\boldsymbol{\theta}^*$.

$$\begin{array}{ll} \underset{\boldsymbol{\theta} \in \mathcal{R}^n}{\textit{minimize}} & \mathcal{F}(\boldsymbol{\theta}) \\ \text{subject to} & \mathbf{A} \cdot \boldsymbol{\theta} = \mathbf{b} \end{array} \tag{4.1}$$

However, only those $\boldsymbol{\theta}$ need be considered which are feasible with respect to the constraints such that $\mathbf{A} \cdot \boldsymbol{\theta} = \mathbf{b}$. If $\mathbf{p}$ is a vector orthogonal to the rows of matrix $\mathbf{A}$ such that Equation 4.2 is true,

$$\mathbf{A} \cdot \mathbf{p} = 0 \tag{4.2}$$

then any point $\boldsymbol{\theta} = \boldsymbol{\theta}^* + \alpha \mathbf{p}$ will be a point feasible with respect to the constraints.

If $\mathbf{Z}$ is a matrix with columns orthogonal to the rows of $\mathbf{A}$ such that Equation 4.3 is satisfied,

$$\mathbf{A} \cdot \mathbf{Z} = 0 \tag{4.3}$$

then $\mathbf{Z}$ represents a basis for the null space of $\mathbf{A}$ and any vector orthogonal to $\mathbf{A}$ can be represented as a linear combination of columns of $\mathbf{Z}$ i.e. $\mathbf{p}$ can be written as in Equation 4.4 for some vector $\mathbf{p}_Z$.

$$\mathbf{p} = \mathbf{Z} \cdot \mathbf{p}_Z \tag{4.4}$$

Using Equation 4.4 a feasible point $\boldsymbol{\theta}$ can be represented as in Equation 4.5:

$$\boldsymbol{\theta} = \boldsymbol{\theta}^* + \alpha \mathbf{Z} \cdot \mathbf{p}_Z \tag{4.5}$$

The behavior of the function $\mathcal{F}$ can be observed at a feasible point in the neighborhood of the minimum $\boldsymbol{\theta}^*$ by expanding $\mathcal{F}(\boldsymbol{\theta})$ using Taylor expansion as shown in Equation 4.6:

$$\begin{aligned} \mathcal{F}(\boldsymbol{\theta}) &= \mathcal{F}(\boldsymbol{\theta}^* + \alpha \mathbf{Z} \cdot \mathbf{p}_Z) \\ &= \mathcal{F}(\boldsymbol{\theta}^*) + \epsilon \mathbf{p}_Z^T \cdot \mathbf{Z}^T \cdot \mathbf{g} + \frac{1}{2}\epsilon^2 \mathbf{p}_Z^T \cdot \mathbf{Z}^T \cdot \mathbf{H} \cdot \mathbf{Z} \cdot \mathbf{p}_Z \end{aligned} \tag{4.6}$$

It is clear from Equation 4.6 that the term $\mathbf{p}_Z^T \cdot \mathbf{Z}^T \cdot \mathbf{g}$ must vanish or else an ϵ can be found such that $\mathcal{F}(\boldsymbol{\theta}) < \mathcal{F}(\boldsymbol{\theta}^*)$, which will imply that $\boldsymbol{\theta}^*$ is not a minimum. Thus, for $\boldsymbol{\theta}^*$ to be a minimum, Equation 4.7 must be true:

$$\mathbf{Z}^T \cdot \mathbf{g} = 0 \tag{4.7}$$

Equation 4.3 and Equation 4.7 imply that the gradient $\mathbf{g}$ can be written as a linear combination of rows of $\mathbf{A}$ (see Equation 4.8), where $\boldsymbol{\lambda}$ is termed as the vector of *Lagrange* multipliers.

$$\mathbf{g} = \mathbf{A}^T \cdot \boldsymbol{\lambda} \tag{4.8}$$

It can also be shown that if $\mathbf{Z}^T \cdot \mathbf{H} \cdot \mathbf{Z}$ is not positive semi definite, the neighborhood of $\boldsymbol{\theta}^*$ will contain points $\boldsymbol{\theta}$ such that $\mathcal{F}(\boldsymbol{\theta}) < \mathcal{F}(\boldsymbol{\theta}^*)$. Thus, $\mathbf{Z}^T \cdot \mathbf{H} \cdot \mathbf{Z}$ must be positive semidefinite for $\boldsymbol{\theta}^*$ to be a minimum of Equation 4.1.

From the preceding discussion it is clear that the necessary conditions for $\boldsymbol{\theta}^*$ to be a minimum of a similar minimization problem given by Equation 4.9 can be stated as:

$$\begin{array}{ll} \underset{\boldsymbol{\theta} \in \mathcal{R}^n}{\textit{minimize}} & \mathcal{F}(\boldsymbol{\theta}) \\ \text{subject to} & \mathbf{l}_L \leq \mathbf{A}_L\boldsymbol{\theta} \leq \mathbf{u}_L \end{array} \tag{4.9}$$

- $\boldsymbol{\theta}^*$ is feasible, i.e. $\mathbf{l}_L \leq \mathbf{A}_L\boldsymbol{\theta}^* \leq \mathbf{u}_L$. $\mathbf{A}_L$ is a matrix with i^{th} row as the gradient $\mathbf{a}_{li}^T$ of the i^{th} constraint.
- *Projected Gradient* is zero ($\mathbf{Z}^T\mathbf{g}(\boldsymbol{\theta}^*) = 0$) or in other words, the gradient $\mathbf{g}(\boldsymbol{\theta}^*)$ is a linear combination of active constraint gradients ($\mathbf{g}(\boldsymbol{\theta}^*) = \hat{\mathbf{A}}_L^T\boldsymbol{\lambda}_L{}^*$), where $\boldsymbol{\lambda}_L{}^*$ is the vector of *Lagrange Multipliers* at $\boldsymbol{\theta}^*$ and $\hat{\mathbf{A}}_L^T$ is the submatrix of $\mathbf{A}_L$ with rows corresponding to active constraints only.
- $\lambda_{Li}^* \geq 0$ if $\mathbf{a}_{Li}^T = l_{Li}$ and $\lambda_{Li}^* \leq 0$ if $\mathbf{a}_{Li}^T = u_{Li}$.
- *Projected Hessian* $\mathbf{Z}^T\mathbf{H}(\boldsymbol{\theta}^*)\mathbf{Z}$ is positive semi-definite.

4.2.2 Nonlinear Constraints

Necessary conditions for $\boldsymbol{\theta}^*$ to be a minimum of a nonlinear programming problem with nonlinear constraints can be stated similarly. The matrices $\mathbf{A}$ and $\mathbf{Z}$ in this case are functions of $\boldsymbol{\theta}^*$:

$$\begin{array}{ll} \underset{\boldsymbol{\theta} \in \mathcal{R}^n}{\textit{minimize}} & \mathcal{F}(\boldsymbol{\theta}) \\ \text{subject to} & \mathbf{l}_N \leq \mathbf{c}(\boldsymbol{\theta}) \leq \mathbf{u}_N \end{array} \tag{4.10}$$

- $\boldsymbol{\theta}^*$ is feasible, i.e. $\mathbf{l}_N \leq \mathbf{c}(\boldsymbol{\theta}^*) \leq \mathbf{u}_N$.
- *Projected Gradient* is zero ($\mathbf{Z}^T(\boldsymbol{\theta}^*)\mathbf{g}(\boldsymbol{\theta}^*) = 0$) or in other words, the Gradient $\mathbf{g}(\boldsymbol{\theta}^*)$ is a linear combinations of constraint Gradients ($\mathbf{g}(\boldsymbol{\theta}^*) = \hat{\mathbf{A}}_N^T(\boldsymbol{\theta}^*)\boldsymbol{\lambda}_N{}^*$).
- $\lambda_{Ni}^* \geq 0$ if $c_i(\boldsymbol{\theta}^*) = l_{Ni}$ and $\lambda_{Ni}^* \leq 0$ if $c_i(\boldsymbol{\theta}^*) = u_{Ni}$.
- *Projected Hessian* of the *Lagrangian* $\mathbf{Z}^T(\boldsymbol{\theta}^*)\mathbf{W}(\boldsymbol{\theta}^*)\mathbf{Z}(\boldsymbol{\theta}^*)$ is positive semi-definite.

4.3 Optimization Methods

Over the years many different classes of algorithms have emerged which can tackle various types of optimization problems. In recent years both the theory and the practice of nonlinear optimization have made tremendous strides and a host of very sophisticated techniques have emerged making the algorithms very efficient. However, these techniques have not yet found extensive use in the Computer Graphics community. This thesis demonstrates the efficacy of these techniques for providing interactive control over an articulated figure.

Typically, these methods transform the original nonlinearly constrained problem into an unconstrained or a linearly constrained problem whose unconstrained/linearly constrained minimum is also the nonlinearly constrained minimum of the original problem.

4.3.1 Penalty Function Methods

These methods [Rya74] add a penalty term to the original function to get a new function to which techniques of unconstrained minimization are applied. The penalty term is such that it increases with the increase in constraint violations and thus tries to guide the solution in the direction which will satisfy the constraints. However, these methods are very sensitive to the choice of penalty parameter and the problem can become defective or ill conditioned for a low or a high value of the parameter [Loo72]. Besides, discontinuities in the function result in the presence of inequality constraints.

4.3.2 Reduced Gradient Methods

The motivation for this class of methods [Ros61] (also known as Gradient Projection Methods) is to exploit the reduction in the search space which results from the presence of constraints. Such algorithms are known to be successful for linearly constrained problems (e.g. [ZB90], [Ros60]). This success prompted attempts to extend them to nonlinearly constrained problems. The search for solutions in these methods is restricted to a subset of points which exactly satisfy the constraints. These methods get their name from the fact that the function to be minimized is "projected" onto a "reduced" subset of points. These methods are suitable for problems where the constraints are nearly linear. In the presence of highly nonlinear constraints, as is the case in our problem, these methods make very slow progress towards the solution.

4.3.3 Augmented Lagrangian Methods

Augmented Lagrangian methods [Hes80], like penalty function methods, are also derived from the idea of converting a nonlinearly constrained problem into an unconstrained problem. However, they avoid the problems of ill conditioning and loss of differentiability, which are inherent in penalty function methods. The objective function is the Lagrangian function of the original function, augmented with a penalty term. Estimation of Lagrange multipliers is required during the course of minimization of the augmented Lagrangian objective function. Though these methods can be applied to the problem in this research, they have a few drawbacks. The convergence of the algorithm is limited by the accuracy of the Lagrange multipliers. If the Lagrange multiplier estimates converge linearly towards true multiplier values, even a quadratic convergence algorithm applied to the unconstrained minimization subproblem will result in only linear convergence. toDetailed discussion of Lagrange multiplier estimates can be found in Gill and Murray [GM79]. Gill et al [GMSW92] should be consulted for convergence properties of the augmented Lagrangian objective function. Since the original problem has linear constraints as well, the advantages which can accrue out of posing an unconstrained subproblem are diminished.

4.3.4 Projected Lagrangian Methods

Algorithms belonging to this class use first order Taylor series approximations of nonlinear constraints to pose a linearly constrained subproblem. The objective function for the subproblem is related to the Lagrangian of the original function. Hence, an estimate of the Lagrange multipliers for the subproblem can also act as an estimate of multipliers for the original problem. Moreover, this class of algorithms is not as sensitive to the accuracy in multiplier estimates as augmented Lagrangian methods. This class of algorithms appears to be most promising for this research since methods developed for minimization in the presence of linear constraints can also be utilized for nonlinear constraints. Different algorithms of this class can be further classified according to the linearly constrained subproblem that they pose.

General Linearly Constrained Subproblem

The linearly constrained subproblem is solved iteratively. These algorithm are quadratically convergent in the neighborhood of a solution. An example of this approach can be found in Rosen and Kreuser [RK72].

A Quadratic Programming Subproblem

The previous case solves an adaptive problem. The effort expended in finding a solution to the subproblem may not be justified if it does not provide a good estimate for a step which will satisfy the constraints. Here, in an attempt to address this problem, a quadratic programming subproblem is posed, the solution to which can be found in one step. Thus, even if the resulting solution to the subproblem is not a good estimate for the right step, not much effort is spent in finding the solution either.

4.4 Proposed System

This research investigates the application of optimization techniques to provide a means for interactively manipulating articulated figures. The figures which come under the ambit of this research are tree-structured hierarchies of planar chains. However, the algorithm is equally applicable to three dimensional figures.

4.4.1 Overview of the Algorithm

As outlined in the previous section, augmented Lagrangian and projected Lagrangian methods are relevant for solving problems arising out of inverse kinematic manipulation of an articulated figure, though both have some drawbacks. Thus the algorithm that has been used is a combination of the methods stated above. It is an active set method where a prediction of the set of constraints active at the solution is made to form a working set of constraints. The search direction is computed so as to remain feasible with respect to the constraints in the working set. Constraints leave and join the working set as the iterations proceed. The "distance" to move along the search direction, or the step length, which produces "sufficient" decrease in an augmented Lagrangian merit function, is computed.

A linearly constrained quadratic programming subproblem is solved to calculate the search direction. A quadratic approximation to the augmented Lagrangian is taken as the objective function, and a subset of the original linear constraints and a linearized version of the original nonlinear constraints is the constraint set in the subproblem. The method can also be categorized as *Sequential Quadratic Programming* (SQP).

Sophisticated matrix factorization techniques are used to improve the efficiency of the algorithm. In addition, simple bounds, general linear constraints and nonlinear constraints are treated separately. This greatly reduces the work involved in factoring and updating the matrices.

The simplified high level steps of the algorithm SQP are given below.

Algorithm SQP

SQP–I *Initialization*: Given a starting point $\boldsymbol{\theta}_0$, form various matrices.

SQP–II *Termination Criteria*: Terminate with $\boldsymbol{\theta}_k$ as the solution if it satisfies optimality conditions

SQP–III *Search Direction (p_k)*: Solve the linearly constrained quadratic programming subproblem

$$\underset{p \in R^n}{\text{minimize}} \qquad \mathbf{g}_k^T \mathbf{p} + \tfrac{1}{2}\mathbf{p}^T \mathbf{W}_k \mathbf{p}$$

$$\text{subject to} \qquad \begin{array}{ccccc} \bar{\mathbf{l}}_B & \leq & \mathbf{p} & \leq & \bar{\mathbf{u}}_B \\ \bar{\mathbf{l}}_L & \leq & \mathbf{A}_L \cdot \mathbf{p} & \leq & \bar{\mathbf{u}}_L \\ \bar{\mathbf{l}}_N & \leq & \mathbf{A}_N \cdot \mathbf{p} & \leq & \bar{\mathbf{u}}_N \end{array} \tag{4.11}$$

$$\text{where} \qquad \begin{array}{ccccc} \bar{\mathbf{l}}_B & = & \mathbf{l}_B & - & \boldsymbol{\theta}_k \\ \bar{\mathbf{l}}_L & = & \mathbf{l}_L & - & \mathbf{A}_L \cdot \boldsymbol{\theta}_k \\ \bar{\mathbf{l}}_N & = & \mathbf{l}_N & - & \mathbf{A}_N \cdot \mathbf{c}(\boldsymbol{\theta}_k) \\ \bar{\mathbf{u}}_B & = & \mathbf{u}_B & - & \boldsymbol{\theta}_k \\ \bar{\mathbf{u}}_L & = & \mathbf{u}_L & - & \mathbf{A}_L \cdot \boldsymbol{\theta}_k \\ \bar{\mathbf{u}}_N & = & \mathbf{u}_N & - & \mathbf{A}_N \cdot \mathbf{c}(\boldsymbol{\theta}_k) \end{array}$$

and set $\mathbf{p}_k$ to the solution. $\mathbf{g}_k$ is the Gradient of the augmented *Lagrangian* function, evaluated at the point $\boldsymbol{\theta}_k$. $\mathbf{W}_k$ is the positive-definite, symmetric, quasi-Newton approximation of the Hessian of the augmented *Lagrangian* at $\boldsymbol{\theta}_k$.

SQP–IV *Step Length (α)*: Calculate α which produces sufficient decrease in the augmented *Lagrangian* $\mathcal{L}(\boldsymbol{\theta}_k, \boldsymbol{\lambda}_k, \mathrm{s})$ where s are the slack variables and $\boldsymbol{\rho}$ is the vector of penalty parameters.

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}_N, \mathrm{s}) \;=\; \mathcal{F}(\boldsymbol{\theta}) - \boldsymbol{\lambda}_N{}^T \cdot (\mathbf{c}(\boldsymbol{\theta}) - \mathrm{s}) + \frac{1}{2}\sum_i^{t_N} \rho_i(c_i(\boldsymbol{\theta}) - s_i) \tag{4.12}$$

SQP–V *Update Values*: Set $\boldsymbol{\theta}_{k+1}$ to the solution of Equation 4.11, $\boldsymbol{\lambda}_{k+1}$ to the Lagrange multiplier estimates of Equation 4.11 at the solution. Calculate the new Hessian. Set k to k+1 and go to SQP–II.

4.4.2 Initialization

Working Set

A working set of constraints is defined by forming a set of constraints which are *exactly* or *nearly* satisfied at the initial point $\boldsymbol{\theta}_0$. Let m be the number of total constraints in the working set. In an active set method, the search direction is computed such that the constraints in the working set remain exactly satisfied. Hence, if there are n_{FX} simple bounds in the working set, the components of the search vector corresponding to the variables with the simple bounds are set to zero. These variables are then said to be *fixed*. Components of $\boldsymbol{\theta}$ are reordered such that the fixed variables come at the end. In addition to the linear bounds, if there are $(t = m - n_{FX})$ general and nonlinear constraints in the working set, then $\hat{\mathbf{A}}_{t\times n}$ denotes the matrix composed of t linear and nonlinear constraint gradients as its rows. Let $(n_{FR} = n - n_{FX})$ denote the number of remaining variables, i.e. the *free* variables. The matrix $\hat{\mathbf{A}}$ can be partitioned as:

$$\begin{aligned}
[\hat{\mathbf{A}}]_{t\times n} &= \left[\begin{array}{ccc|cc} a_{1,1} & \cdots & a_{1,n_{FR}} & \cdots & a_{1,n_{FR}+n_{FX}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{t,1} & \cdots & a_{t,n_{FR}} & \cdots & a_{t,n_{FR}+n_{FX}} \end{array}\right] \\
[\hat{\mathbf{A}}]_{t\times n} &= \left[\begin{array}{c|c} [\mathbf{A}_{FR}]_{t\times n_{FR}} & [\mathbf{A}_{FX}]_{t\times n_{FX}} \end{array}\right] \qquad (4.13)
\end{aligned}$$

If $\hat{\mathbf{C}}$ is the $(m \times n)$ matrix whose rows are Gradients of all the m constraints in the working set, and whose first n_{FX} rows correspond to the linear bounds in the working set, then it can be partitioned as:

$$[\hat{\mathbf{C}}]_{m\times n} = \left[\begin{array}{c|c} [0]_{n_{FX}\times n_{FR}} & [\mathbf{I}_{FX}]_{n_{FX}\times n_{FX}} \\ \hline \mathbf{A}_{FR} & \mathbf{A}_{FX} \end{array}\right] \qquad (4.14)$$

TQ decomposition of the $(t \times n_{FR})$ matrix $\mathbf{A}_{FR}$ is calculated, where $\mathbf{Q}_{FR}$ is an $(n_{FR} \times n_{FR})$ *Orthonormal* matrix which reduces the first $(z = n_{FR} - t = n - m)$ columns of $\mathbf{A}_{FR}$ to zeros and last t columns to a $t \times t$ reverse triangular matrix $\mathbf{T}_{FR}$ such that $t_{i,j} = 0$, if $i + j \leq t$.

$$\mathbf{A}_{FR} \cdot \mathbf{Q}_{FR} = \left[\begin{array}{c|c} [0]_{t\times z} & [\mathbf{T}_{FR}]_{t\times t} \end{array}\right] \qquad (4.15)$$

Clearly, the first z columns of $\mathbf{Q}_{FR}$ are orthogonal to the rows of $\mathbf{A}_{FR}$. Hence these columns can be taken as $\mathbf{Z}_{FR}$ the matrix which spans the null space of rows of $\mathbf{A}_{FR}$. The last t columns are taken as $\mathbf{Y}_{FR}$ whose columns span the range space of $\mathbf{A}_{FR}$. Thus an implicit partitioning of $\mathbf{Q}_{FR}$ takes place as shown in Equation 4.16:

$$[\mathbf{Q}_{FR}] = \left[\ [\mathbf{Z}_{FR}]_{n_{FR} \times z} \ \middle| \ [\mathbf{Y}_{FR}]_{n_{FR} \times t}\ \right] \tag{4.16}$$

An $n \times n$ *Orthonormal* matrix $\mathbf{Q}$ is composed using $\mathbf{Q}_{FR}$ and an $(n_{FX} \times n_{FX})$ *Identity* matrix $\mathbf{I}_{FX}$ as:

$$[\mathbf{Q}]_{n \times n} = \left[\begin{array}{c|c} \mathbf{Q}_{FR} & [0]_{n_{FR} \times n_{FX}} \\ \hline [0]_{n_{FX} \times n_{FR}} & \mathbf{I}_{FX} \end{array}\right] \tag{4.17}$$

The matrix $\mathbf{Q}$ can be partitioned into matrices $\mathbf{Z}$ and $\mathbf{Y}$ which span the null and range space of matrix $\hat{\mathbf{C}}$ respectively:

$$\begin{aligned} [\mathbf{Q}] &= \left[\begin{array}{c|cc} \mathbf{Z}_{FR} & \mathbf{Y}_{FR} & 0 \\ 0 & 0 & \mathbf{I}_{FX} \end{array}\right] \\ &= \left[\ \mathbf{Z} \ \middle| \ \mathbf{Y}\ \right] \end{aligned} \tag{4.18}$$

Postmultiplying $\hat{\mathbf{C}}$ with $\mathbf{Q}$ results in:

$$\hat{\mathbf{C}} \cdot \mathbf{Q} = \left[\ [0]_{m \times z} \ \middle| \ [\mathbf{T}]_{m \times m}\ \right] \tag{4.19}$$

$$\text{where } [\mathbf{T}]_{m \times m} = \begin{bmatrix} [0]_{n_{FX} \times t} & \mathbf{I}_{FX} \\ \mathbf{T}_{FR} & \mathbf{A}_{FX} \end{bmatrix} \tag{4.20}$$

Factorization of the Hessian

If $\mathbf{W}_P$ denotes the permuted Hessian of the Lagrangian, such that the elements corresponding to the fixed variables are in the lower right corner, then $\mathbf{W}_P$ can be partitioned as shown in Equation 4.21. $\mathbf{W}_{FR}$ and $\mathbf{W}_{FX}$ are square symmetric matrices of dimensions $(n_{FR} \times n_{FR})$ and $(n_{FX} \times n_{FX})$ respectively and correspond to free and fixed variables.

$$[\mathbf{W}_P]_{n \times n} = \left[\begin{array}{c|c} [\mathbf{W}_{FR}]_{n_{FR} \times n_{FR}} & [\mathbf{W}_{FRFX}]_{n_{FR} \times n_{FX}} \\ \hline [\mathbf{W}^T_{FRFX}]_{n_{FX} \times n_{FR}} & [\mathbf{W}_{FX}]_{n_{FX} \times n_{FX}} \end{array}\right] \tag{4.21}$$

The permuted Hessian of the Lagrangian $\mathbf{W}_P$ is projected on the orthogonal matrix $\mathbf{Q}$ and the Cholesky factors of the projected Hessian are calculated as shown in Equation 4.22. The matrix $\mathbf{R}_Q$ in Equation 4.22 is an upper triangular matrix such that $r_{Qij} = 0$ if $i > j$.

$$\mathbf{Q}^T \cdot \mathbf{W}_P \cdot \mathbf{Q} = \mathbf{R}_Q^T \cdot \mathbf{R}_Q \tag{4.22}$$

Then, the Cholesky factor $\mathbf{R}_Z$ of the Hessian of the Lagrangian, corresponding to the free variables, projected on the null space of $\mathbf{A}_{FR}$ ($\mathbf{Z}_{FR}$) is the top left $z \times z$ corner of the matrix $\mathbf{R}_Q$ such that:

$$\mathbf{Z}_{FR}{}^T \cdot \mathbf{W}_{FR} \cdot \mathbf{Z}_{FR} = \mathbf{R}_Z{}^T \cdot \mathbf{R}_Z \tag{4.23}$$

The availability of Cholesky factorization of the Hessian allow efficient calculation of the search direction. The advantage of forming $\mathbf{Z}_{FR}$ from a TQ factorization of $\mathbf{A}_{FR}$ is that the columns of $\mathbf{Z}_{FR}$ are orthonormal vectors. Thus the condition number of the projected Hessian $\mathbf{Z}_{FR}^T\mathbf{W}_{FR}\mathbf{Z}_{FR}$ is the same as that of $\mathbf{W}_{FR}$, which would not be the case if the columns of $\mathbf{Z}_{FR}$ were not orthonormal. This has an important implication for the solvability of the problem, as a high condition number of $\mathbf{Z}_{FR}^T\mathbf{W}_{FR}\mathbf{Z}_{FR}$ results in numerical inaccuracies in the solution of a system of equations involving $\mathbf{Z}_{FR}^T\mathbf{W}_{FR}\mathbf{Z}_{FR}$.

4.4.3 Search direction

The search direction $\mathbf{p}_k$ is taken as the solution of the *positive-definite*, quadratic programming problem represented by Equation 4.11. Solution of Equation 4.11 is also an iterative process. The search direction $\mathbf{p}_{QP}$ for the quadratic programming problem is constructed by solving the quasi-Newton equations, such that a move along that direction will be strictly feasible with respect to all the constraints in the working set. A unit step along this direction will minimize the objective function in the subspace of constraints in the active set, if they are assumed to be the only constraints which hold with equality at the solution. However, it is possible that one or more of the inequality constraints, not present in the working set will get violated by a unit step in the search direction. Thus a maximum step length α_{QP_m} is calculated which takes the joint angle vector to the "nearest" inactive constraint. If α_{QP_m} is less than the unit step length, then a step to the corresponding constraint is taken. This constraint enters the working set of constraints for the next iteration. Otherwise a unit step in the search direction is taken to the minimum of the quadratic function in the subspace of constraints in the working set.

At the end of a quadratic programming iteration, Lagrange multipliers for all the constraints in the active set are calculated. The Lagrange multiplier estimates are used for predicting if any of the constraints in the working set will not be active at the solution of the quadratic programming problem. Such constraints are deleted from the working set for the next iteration.

Important steps in the process of minimization of the quadratic programming subproblem are given below.

Search Direction

The search direction is calculated by solving the quasi-Newton Equations given by:

$$\begin{aligned} \mathbf{W}_Z \cdot \mathbf{p}_Z &= -\mathbf{g}_Z \\ \text{or } \mathbf{Z}_{FR}^T \mathbf{W}_{FR} \mathbf{Z}_{FR} \cdot \mathbf{p}_Z &= -\mathbf{Z}_{FR}{}^T \mathbf{g} \end{aligned} \tag{4.24}$$

Solution of Equation 4.24 requires inversion of the $(z \times z)$ matrix $\mathbf{W}_Z$ and hence requires $O(z^3)$ operations. However, availability of a *Cholesky* factorization of $\mathbf{W}_Z$ (see Equation 4.23) permits an efficient solution for Equation 4.24. The system of equations given by Equation 4.24 is better solved by solving two triangular systems of equations given by Equations 4.25 and 4.26. The number of operations required for solving a triangular system is only $O(z^2)$. An intermediate vector $\mathbf{f}_z$ is found by solving Equation 4.25, which is then used in the Equation 4.26 to calculate the vector $\mathbf{p}_Z$. Search vector component $\mathbf{p}_{FR}$ corresponding to the free variables is given by Equation 4.27. $\mathbf{p}_{FR}$ and a zero vector corresponding to the fixed variables taken together constitute the complete search vector $\mathbf{p}$ as shown in Equation 4.28.

$$\mathbf{R}_Z{}^T \cdot \mathbf{f}_z = -\mathbf{g}_Z \tag{4.25}$$

$$\mathbf{R}_Z \cdot \mathbf{p}_Z = \mathbf{f}_z \tag{4.26}$$

$$\mathbf{p}_{FR} = \mathbf{Z}_{FR} \cdot \mathbf{p}_Z \tag{4.27}$$

$$\mathbf{p}_{QP} = \left[\begin{array}{c} \mathbf{p}_{FR} \\ \hline [0]_{n_{FX} \times 1} \end{array}\right] \tag{4.28}$$

Addition of Constraints

In an active set method, the working set constitutes the subset of constraints which are predicted to be active at the solution. As the iterations proceed, the prediction of active constraints changes and some constraints enter the working set and some constraints leave the working set. Changes in the working set $\hat{\mathbf{A}}$ induce changes in other matrices also.

In particular, when a general linear constraint becomes active and enters the working set, the row dimension of $\hat{\mathbf{C}}$, $\hat{\mathbf{A}}$, and $\mathbf{A}_{FR}$ increases by one. As can be seen from Equation 4.15, the row and column dimensions of $\mathbf{T}_{FR}$ increase by one while the column dimension of $\mathbf{Z}_{FR}$ decreases by one. If the new constraint gradient is $\mathbf{a}^T$ and is assumed to join at the end of the list of active constraints, then the relationship given by Equation 4.15 can be represented as in Equation 4.29.

$$\begin{aligned} \left[\begin{array}{c} \mathbf{A}_{FR} \\ \hline \mathbf{a}^T \end{array}\right] \cdot \mathbf{Q}_{FR} &= \left[\begin{array}{c|c} [0]_{t\times z} & [\mathbf{T}_{FR}]_{t\times t} \\ \hline [\mathbf{w}^T]_{1\times z} & [\mathbf{t}^T]_{1\times t} \end{array}\right] \\ \text{where } \left[\begin{array}{c|c} \mathbf{w}^T & \mathbf{t}^T \end{array}\right] &= \mathbf{a}^T \cdot \mathbf{Q}_{FR} \end{aligned} \tag{4.29}$$

An $(n_{FR} \times n_{FR})$ orthonormal matrix $\tilde{\mathbf{Q}}$, is constructed which when postmultiplied on both sides of Equation 4.29, reduces the first $(z-1)$ elements of $\mathbf{w}^T$ to zero by taking linear combinations of the first z columns of the right hand side of Equation 4.29. Then, the last column of $\mathbf{w}^T$ becomes the first column of the last row of the new factor $\bar{\mathbf{T}}$. This transforms the right hand side to the appropriate form.

Postmultiplying $\mathbf{Q}_{FR}$ with $\tilde{\mathbf{Q}}$ results in a new orthonormal matrix $\bar{\mathbf{Q}}_{FR}$ which has the first z columns as linear combinations of the first z columns of $\mathbf{Q}_{FR}$ and the last t columns the same as those of $\mathbf{Q}_{FR}$. Reduction in the column dimension of $\mathbf{Z}_{FR}$ is reflected by partitioning $\bar{\mathbf{Q}}_{FR}$ such that the z^{th} column of $\bar{\mathbf{Q}}_{FR}$ becomes the first column of $\bar{\mathbf{Y}}_{FR}$. Since $\mathbf{Z}_{FR}$ and $\mathbf{R}_Z$ are related by Equation 4.23, reduction in the column dimension of $\mathbf{Z}_{FR}$ results in reduction in the row and column dimension of $\mathbf{R}_Z$.

Post-multiplication of $\mathbf{Q}_{FR}$ by $\tilde{\mathbf{Q}}$ results in post-multiplication of $\mathbf{R}_Z$ with $\tilde{\mathbf{Q}}$. The new matrix $\mathbf{R}_Z \cdot \tilde{\mathbf{Q}}$ has an extra subdiagonal element in each column. An orthonormal matrix $\check{\mathbf{Q}}$ is constructed such that premultiplying it with $\mathbf{R}_Z$ results in elimination of subdiagonal elements in columns 1 to $(z-1)$ by taking linear combinations of rows. The matrix $\check{\mathbf{Q}} \cdot \mathbf{R}_Z \cdot \tilde{\mathbf{Q}}$ without its z^{th} row and column is the new *Cholesky* factor $\bar{\mathbf{R}}_Z$ such that $\bar{\mathbf{Z}}_{FR}^T \cdot \mathbf{W}_{FR} \cdot \bar{\mathbf{Z}}_{FR} = \bar{\mathbf{R}}_Z^T \cdot \bar{\mathbf{R}}_Z$

When a simple bound enters the working set, the number of fixed variables increases by one. The variable corresponding to the new constraint in the working set is assumed to be the last variable in the list of free variable and the new constraint Gradient becomes the first row of the active constraint matrix $\hat{\mathbf{C}}$. The addition of a bound constraint in the working set thus results in reduction in the column dimension of $\mathbf{A}_{FR}$. Thus the row and column dimension of $\mathbf{Q}_{FR}$ and the row dimension of $\mathbf{Z}_{FR}$ is reduced by one. Reduction in the row dimension of $\mathbf{Z}_{FR}$ results in reduction in the row and column dimension of $\mathbf{R}_Z$. Various matrices are updated in a manner similar to when a general constraint enters the working set.

Deletion of Constraints

When a general constraint is deleted from the working set of constraints, the row dimension of the matrix $\mathbf{A}_{FR}$ decreases by one resulting in a decrease in the row and column dimension of the matrix $\mathbf{T}_{FR}$ and an increase in the column dimension of the matrix $\mathbf{Z}_{FR}$. If the i^{th} row of $\mathbf{A}_{FR}$ is deleted because of the constraint leaving the active set and $\bar{\mathbf{A}}_{FR}$ is the new matrix of constraint gradients, then Equation 4.15 results in

$$[\bar{\mathbf{A}}_{FR}]_{(t-1)\times n_{FR}} \cdot \mathbf{Q}_{FR} = \left[\ [0]_{(t-1)\times z} \ \middle|\ [\tilde{\mathbf{T}}_{FR}]_{(t-1)\times t}\ \right] \tag{4.30}$$

where columns 1 to $(t-i)$ of $\tilde{\mathbf{T}}_{FR}$ have one extra element on the top. An orthonormal matrix $\tilde{\mathbf{Q}}$ is constructed such that it reduces the extra elements to zero when post-multiplied to the right hand side of Equation 4.30, by taking the linear combination of columns $(t-i+1)$ to 1, while leaving all other columns untouched. The resulting matrix without its first column is the required $((t-1)\times(t-1))$ matrix $\bar{\mathbf{T}}_{FR}$. The result of the same multiplication on the left hand side of the equation is an orthonormal matrix $\bar{\mathbf{Q}}_{FR}$ whose first z and last $(i-1)$ columns are same as that of $\mathbf{Q}_{FR}$ while the $(z+1)^{th}$ to $(z+t-i)^{th}$ columns are linear combinations of appropriate columns of $\mathbf{Q}_{FR}$. The increase in the column dimension of the new $\bar{\mathbf{Z}}_{FR}$ is reflecting by partitioning the of $\bar{\mathbf{Q}}_{FR}$ where the $(z+1)^{th}$ column of $\bar{\mathbf{Q}}_{FR}$ becomes the last column of $\bar{\mathbf{Z}}_{FR}$.

An increase in the column dimension of $\bar{\mathbf{Z}}_{FR}$ results in an increase in the row and column dimension of $\mathbf{W}_Z$ and hence in the row and column dimension of $\mathbf{R}_Z$. The new *Cholesky* factor $\bar{\mathbf{R}}_Z$ has the form given by Equation 4.31 where the vector $\mathbf{r}$ and element γ are determined by a step of *Cholesky* factorization [GGMS74].

$$\bar{\mathbf{R}}_Z = \left[\begin{array}{c|c} \mathbf{R}_Z & r \\ 0 & \gamma \end{array}\right] \tag{4.31}$$

Similar methods for updating various matrices are employed when a simple bound leaves the working set of constraints.

4.4.4 Step Length

The solution of the quadratic programming subproblem is used as the search vector for the original nonlinearly constrained problem. A scalar α is calculated by a linesearch algorithm which produces a sufficient decrease in the augmented Lagrangian such that Equation 4.32 holds. The penalty parameter vector ρ is increased, if required, to produce sufficient decrease in the merit function. Details of various schemes to select the *sufficient decrease* parameter δ can be found in [GMW81].

$$\mathcal{L}(\boldsymbol{\theta}_k + \alpha\mathbf{p_k}, \boldsymbol{\lambda}_k, \mathbf{s}_k) - \mathcal{L}(\boldsymbol{\theta}_k + \alpha\mathbf{p_k}, \boldsymbol{\lambda}_k, \mathbf{s}_k) \geq \delta \tag{4.32}$$

4.4.5 Update Values

The solution of the quadratic programming subproblem provides the search vector. The set of constraints in the final working set of the subproblem corresponding to the nonlinear constraints give a prediction of the set of active nonlinear constraints. Corresponding Lagrange multipliers $\boldsymbol{\lambda}_N$ at the solution of the subproblem are taken as estimates of the Lagrange multipliers for the nonlinear constraints. The estimate of the solution vector $\boldsymbol{\theta}$ is updated as $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \alpha \mathbf{p_k}$. The change in $\boldsymbol{\theta}$ makes it necessary to recalculate the Hessian of the Lagrangian, and hence the projected Hessian. Calculation of the Hessian from scratch is an expensive operation. However, it is possible to obtain an approximation of the new Hessian from the old Hessian and curvature information obtained from the move along the search vector. Various schemes exist (DFP [FP63], PSB [Pow70], BFGS [Bro88], [Fle70], [Gol70], [Sha70]) which allow construction of the new Hessian by addition of a low rank matrix to the old Hessian. The BFGS update scheme, which has been used here, is known to be more effective than other earlier schemes, and makes a rank two update to the old Hessian to obtain the new Hessian. The process can be carried out in $O(n^2)$ operations. BFGS updates ensure that the new Hessian is positive definite, a property which is very important for calculating the correct search direction. These updates also have the property of preserving the symmetry of the new Hessian. The updates for the projected Hessian can be represented by Equation 4.33, where $\bar{\mathbf{W}}_Q$ is the updated projected Hessian, $\mathbf{W}_Q$ is the old projected Hessian, $\mathbf{y}_Q = \mathbf{Q}^T(\mathbf{g}_{k+1} - \mathbf{A}_{Nk+1}\boldsymbol{\lambda}_N - \mathbf{g}_k + \mathbf{A}_{Nk}\boldsymbol{\lambda}_N)$, $\mathbf{s}_Q = \mathbf{Q}^T(\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k)$, $\mathbf{A}_{Nk+1}$ is the matrix of the nonlinear constraint gradients evaluated at $\boldsymbol{\theta}_{k+1}$ and $\mathbf{A}_{Nk}$ is the corresponding matrix evaluated at $\boldsymbol{\theta}_k$.

$$\bar{\mathbf{W}}_Q = \mathbf{W}_Q - \frac{1}{\mathbf{s}_Q^T\mathbf{W}_Q\mathbf{s}_Q}\mathbf{W}_Q\mathbf{s}_Q\mathbf{s}_Q^T\mathbf{W}_Q + \frac{1}{\mathbf{y}_Q^T\mathbf{s}_Q}\mathbf{y}_Q\mathbf{y}_Q^T \tag{4.33}$$

The rank-two update given by Equation 4.33 can be translated into a rank-one update to the Cholesky factor $\mathbf{R}_Q$ of $\mathbf{W}_Q$ to obtain the updated Cholesky factor $\bar{\mathbf{R}}_Q$ of $\bar{\mathbf{W}}_Q$ as given by Equation 4.34,

$$\bar{\mathbf{R}}_Q = \mathbf{R}_Q + \mathbf{u}\mathbf{v}^T \tag{4.34}$$

where $\mathbf{u} = \frac{1}{\sqrt{\mathbf{s}_Q^T\mathbf{W}_Q\mathbf{s}_Q}}\mathbf{R}_Q\mathbf{s}_Q$ and $\mathbf{v} = \frac{1}{\sqrt{\mathbf{y}_Q^T\mathbf{s}_Q}}\mathbf{y}_Q - \frac{1}{\sqrt{\mathbf{s}_Q^T\mathbf{W}_Q\mathbf{s}_Q}}\mathbf{W}_Q\mathbf{s}_Q$

Chapter 5

The Animation System

A simple system was created for demonstrating the efficacy of algorithms for providing interactive and intuitive control over articulated figures. This chapter describes the system and rationale behind various design decisions. Many interesting issues, which emerged from the experience gained by the development and use of the system, are analyzed.

The system has a built-in interactive editor for creating articulated figures. A figure can be directly manipulated for the purpose of creating keyframes. Constraints can be specified on the figure to maintain the physical integrity of the underlying entity. Ability to create complete motion sequences is also provided. This is done by specifying a trajectory for an end effector. The end effector moves along the trajectory over a period of time resulting in a movement sequence. The use of trajectories allows for the manipulation of multiple parts of the figure simultaneously. The ability to create complete movement sequences allows fast prototyping of animations. However, the system described is not intended to be a comprehensive package for animation. Rather, it consists of a set of services which provide a means for testing techniques for articulated figure animation, and will be a powerful set of tools for direct manipulation within the framework of a keyframe animation system.

5.1 Planar Articulated Figures

5.1.1 Interactive Figure Editor

Articulated figures such as human skeletons or vertebrate animals are best represented as trees. A simple interactive editor has been implemented to allow the user to create such figures on the fly (see Figure 5.1). The system has a work area window where the figures are drawn and animated, labeled *Animator*, and a control panel window which allows the user to issue commands to draw or animate the figure, labeled *Figure Menu*. The user session is started by pressing the *Edit* button, which results in a dot representing the root to appear on the screen.

The figure is drawn by selecting the root or an end point of a segment and pressing the *Add Segment* button. This causes a new segment to appear at the end of the selected segment. The new segment is a child of the selected segment. The angle that the new segment makes with its parent segment and its length can be changed by dragging the endpoint of the segment away from the root using the mouse, or by the associated sliders on the control panel. Each segment has associated with it an upper and lower limit on the angle it makes with its parent. When a segment is added, these limits appear on the screen as two lines by its sides. The limits can be changed either by dragging or from the control panel. By default the free endpoint of the most recently drawn segment is the selected joint. Thus pressing the *Add Segment* button again results in a new segment appearing at that point.

In order to delete a segment, the user first selects the segment by clicking on it, and then presses the *Remove Segment* button from the control panel. If the segment being deleted has any children, then they become the children of its parent segment.

Any joint can be made the root by first selecting the joint and then pressing the *Make Root* button on the panel. All the segments connected at that joint become children of the root. Parent-child relationships are reversed along the chain from the old root to the new root. The process is illustrated in the Figure 5.2.

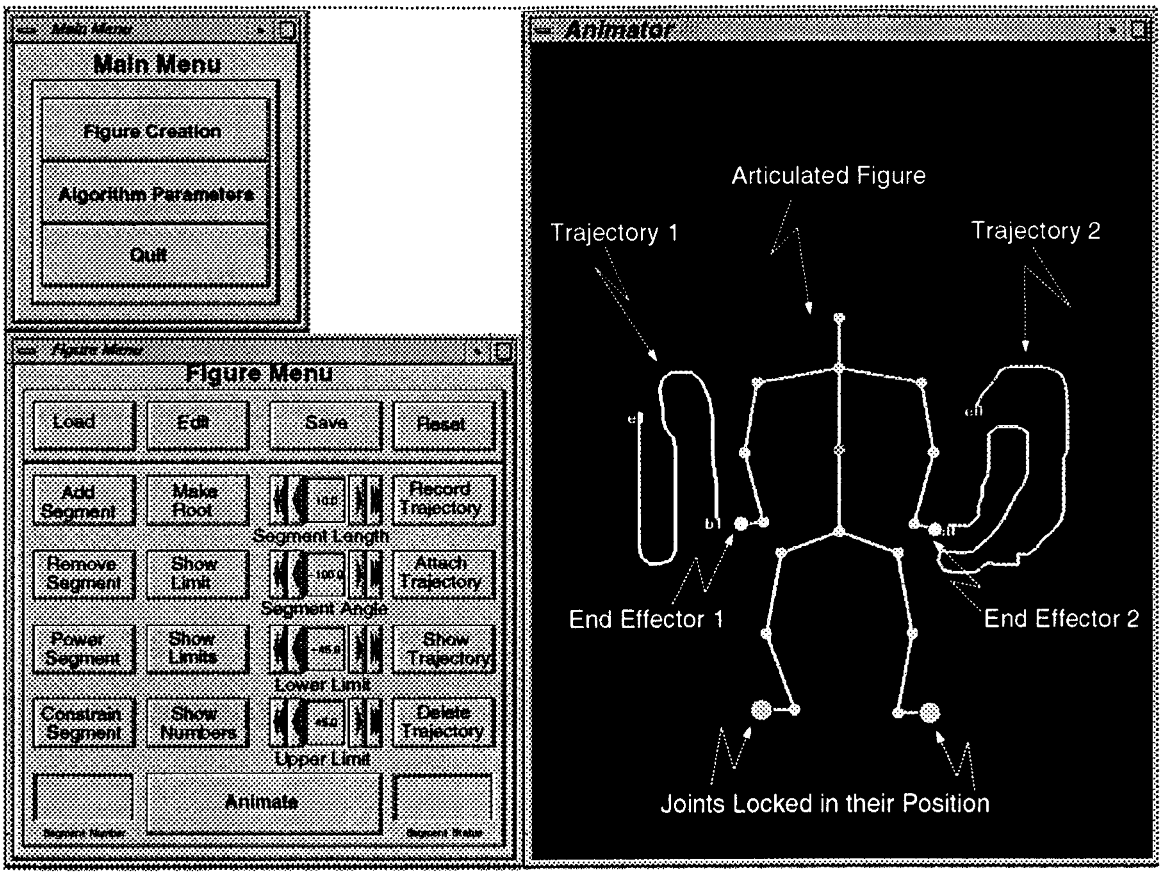

Figure 5.1: The Screen

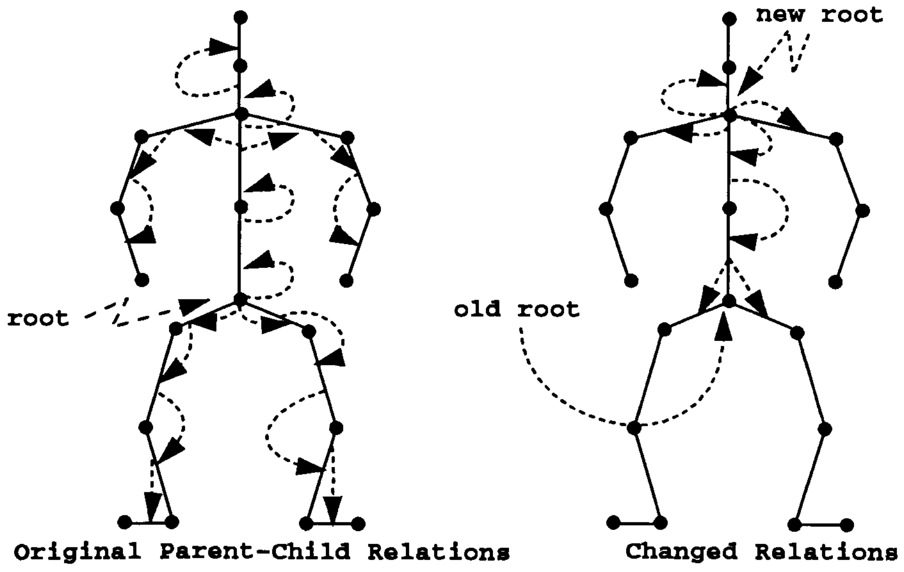

Figure 5.2: Changing the Root

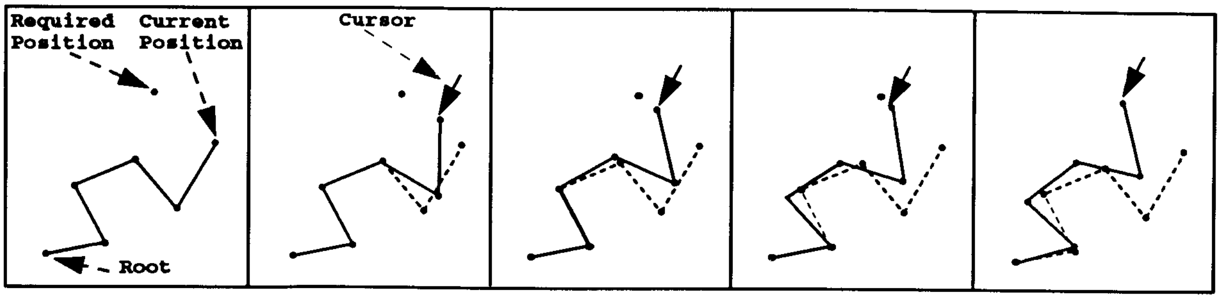

Figure 5.3: Direct Manipulation Of A Movable Kinematic Chain

5.1.2 Keyframing and Animation

When manipulating a figure, the root remains fixed and other segments move with respect to the root. The figure can be translated as a whole by dragging the root.

With inverse kinematic control, this system allows the user to directly manipulate a part of the figure for creating a keyframe. The part may consist of more than one segment. This is a marked improvement over a system without inverse kinematic control, which would require setting of individual joint angles. For a part to be manipulated, there has to be an end effector, which is used for dragging the part. Any joint can be made an end effector by selecting the joint by clicking on it. The system then determines the movable part by traversing the figure hierarchy from the end effector to the root. A joint locked in its position or the root, whichever is encountered first, becomes the *de facto* root. The set of segments between the *de facto* root and the end effector constitutes the movable part. The user can manipulate the part by dragging the end effector to the goal. Different parts of the figure are manipulated in this manner to get a desired keyframe. Figure 5.3 shows snapshots of a movable kinematic chain during a user session of dragging the end effector to the goal.

The keyframes created are analogous to the snapshots of a movement sequence

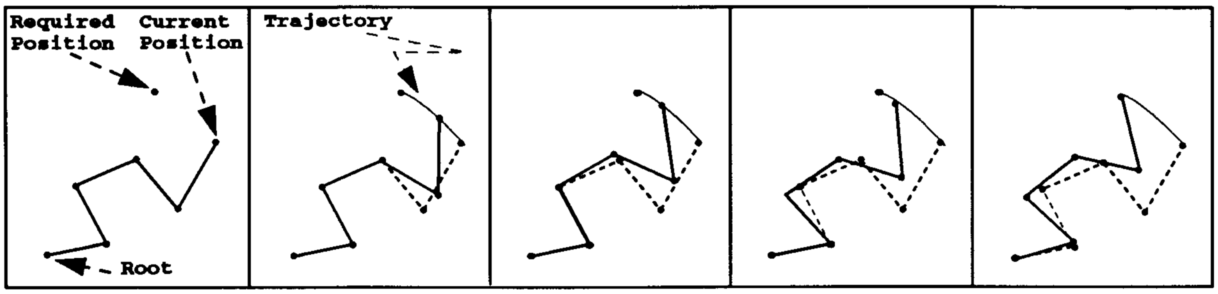

Figure 5.4: Manipulation Of A Movable Kinematic Chain Using A Trajectory

which is constructed by suitable interpolation of the keyframes. However, the ability to interactively create a complete movement sequence can markedly reduce the production time. The system incorporates this feature by associating a trajectory, which is just a curve in space, with the end effector. Pressing the *Animate* button on the control panel results in animation of the movable part such that the end effector follows the trajectory. Figure 5.4 shows the manipulation of a movable chain by associating a trajectory with the end effector.

A trajectory is created by pressing the *Record Trajectory* button and tracing out a curve in the work area with the left mouse button depressed. The *Delete Trajectory* button on the control panel can be used to delete a trajectory.

Multiple parts of the figure can be animated by making two or more joints end effectors. This is done by first selecting a joint and then pressing the *Power Segment* button. However, the mouse cannot be used to animate multiple parts of the figure simultaneously. Hence, a trajectory is associated with each end effector. Pressing the *Animate* button results in the movement of each part corresponding to an end effector such that the end effector follows the associated trajectory.

These abilities demonstrate the efficacy of inverse kinematic control for facilitating the task of articulated figure animation. However, many fruitful directions still remain to be explored. This implementation stores a trajectory as a list of points, hence subsequent modification of the trajectory is not possible. The use of splines can provide considerable flexibility in the creation and modification of trajectories. Presently we allow only the synchronized movement of end effectors along their trajectories with constant speed. A control over the speed of different end effectors is highly desirable.

5.1.3 Constraints

When animating an articulated figure, it becomes imperative to impose some constraints on it so that the physical integrity of the entity being represented by the figure is maintained. For example, if the figure represents a human skeleton, the segment representing the lower arm can only make an angle within the limits of 0 and 180 degrees with the segment representing the upper arm. Such constraints on joint angles are effected by the limits associated with the segments.

Other useful constraints are the ones which constrain a point on the figure at a point in space. This type of constraint can be specified by pressing the *Constrain Segment* button, which results in two constraints, one for each coordinate. The end of the selected segment farthest from the root then becomes locked at its current position. However, the orientation of the segment can still change. This type of constraint is useful for situations such as keeping the feet of the figure fixed to the ground. They are also very effective for the task of keyframe creation. Normally, the process of creating keyframes involves bringing an extremity of a *limb* to a goal with the limb in a particular pose. For such a task the end effector of the articulated chain representing the limb can first be dragged to the goal and then constrained to remain at that position. Any of the interior segments can then be manipulated to attain the desired pose without dislodging the end effector from the goal. In addition, constraints which lock the orientation of a segment or make an end effector stay in a plane can also be incorporated.

Normally for an entity being modeled by the articulated figure, variations in joint angles display some functional relationship. For example, in a human walk, a relationship can be deduced between the variations in angles at the knee and the heel during a step cycle. Such relationships can be linear or nonlinear. A library of such entity-specific constraints can be defined which can then be the default constraints when animating a figure modeling a particular entity. A library of entity specific constraints and a library of predefined figures for different entities can be a useful addition to any animation system. The system presented in this thesis can handle such constraints. However, the techniques developed are for the whole class of articulated figures and no attempt has been made to define any entity-specific constraints.

5.1.4 Data Structures

An articulated figure is represented as a tree which consists of a set of nodes, a pointer to the root node and a linked list of pointers to the leaf nodes. Each node of the tree stores the *length, angle, lower limit, upper limit* and *type* attributes of a segment. In addition, each node has a pointer to its parent and a linked list of pointers to its children. The value of the *type* field determines if the endpoint of the associated segment away from the root is an end effector or is locked in its position.

For tasks involving manipulation of a part of the figure using an end effector, angles at the joints from the root to the end effector need to be determined by solving an inverse kinematic problem. The corresponding kinematic chain of segments, from the root to the end effector is termed a *movable* chain. Any subtrees from the *movable* chain only change in their position and orientation, while the angles at the joints of these subtrees remain fixed. However, if a subtree emanating from a *movable* chain has a constrained joint, then the joint angles of the kinematic chain from the movable chain to the constrained joint also become a part of the inverse kinematic problem, but the subtrees rooted at the constrained joint become impervious to any changes in the configuration of rest of the tree. Such subtrees can then be taken as independent subproblems for animation. Any subtree which does not have an end effector does not change its configuration during the course of the animation. Such subtrees, called *static*, are effectively discarded from the problem of inverse kinematic calculation of

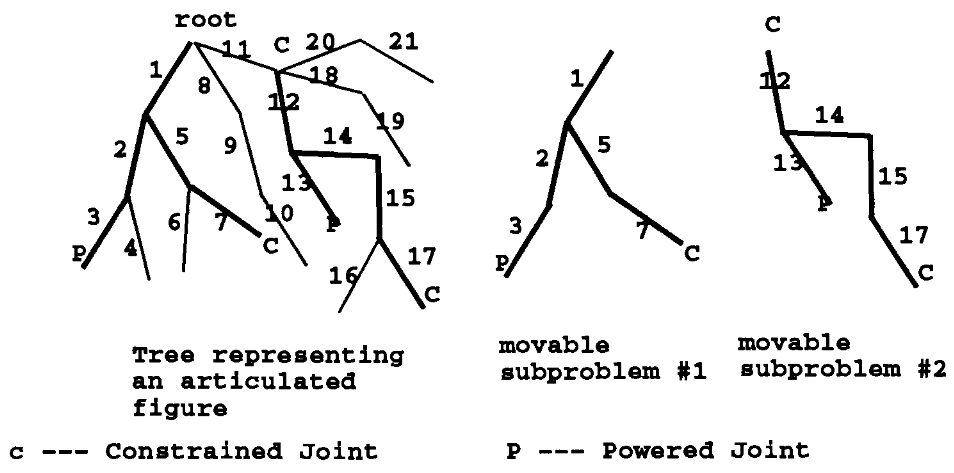

Figure 5.5: Partition of a Tree into Subproblems

the joint angles.

We see from the description given above that end effectors and constrained joints induce a partitioning of the tree into subtrees, some of which are *movable* and the rest *static* subtrees. The movable subtrees are treated as independent subproblems by the inverse kinematic control algorithm. A linked list of *movable* subtrees is created before every animation session. Nodes in the list represent independent problems, which are solved sequentially. After all nodes in the problem tree list are visited, the figure on the screen is updated to reflect the new state. Figure 5.5 shows the process of partitioning the tree into subtrees representing independent problems.

5.2 Three-Dimensional Figures

Throughout this thesis emphasis has been laid on planar figures. However, the techniques presented earlier can be applied to three-dimensional figures as well. This section gives a brief outline of how this can be accomplished.

One way of manipulating a three dimensional figure is by calculating the Gradient and the Hessian of the associated distance function. The technique for procedural evaluation of these quantities, as described in Chapter 3, is based on the observation that the end effector position with respect to the base frame is given by the cascaded multiplication of matrices, all of which display a particular structure. This observation also holds for a three dimensional figure. Thus, there is a possibility of arriving at a similar procedural formulation of the gradient and the Hessian for a three-dimensional figure.

The use of the Denavit-Hartenberg convention for specifying coordinate frames also assumes more importance in the context of three dimensional figures. Routines have been developed in the field of Robotics which can symbolically evaluate the gradient and the Hessian for a figure specified using the convention. Thus extension of the system is possible where a module constructs the Denavit-Hartenberg specification of a figure and another module symbolically evaluates the necessary higher order information. Such a system however is likely to have a substantially degraded performance.

However, a simpler technique can be developed by working with the planar projections of a three dimensional figure. It is instructive to note that an articulated figure can be made to move in space by allowing rotation at the joints to be in two planes instead of one. Moreover, a planar projection of a three dimensional figure is of the type handled in this research, and all the techniques described earlier apply. From this it follows that a three dimensional figure can be animated by sequentially applying the inverse kinematic algorithm to its projections in two orthogonal planes.

Let F be a three dimensional chain, C be a fixed coordinate frame. Let E be the coordinate vector of the end effector and P be the coordinate vector of the goal, both given with respect to C. As shown in Figure 5.6, F_{xy}, E_{xy} and P_{xy} are the projections of F, E and P respectively on the x-y plane of C and F_{yz}, E_{yz} and P_{yz} are the corresponding projections on y-z plane of C. Animating E to P involves the following steps. The inverse kinematic control algorithm for planar chains is invoked on F_{xy}. The joint angles of F_{xy} change so that E_{xy} is at P_{xy} at the end of the iterations. Let F'_{xy} and E'_{xy} be the updated projections of F_{xy} and E_{xy} on the x-y plane.

Lengths of the segments of F_{xy} do not change if the joints of F rotate only in the

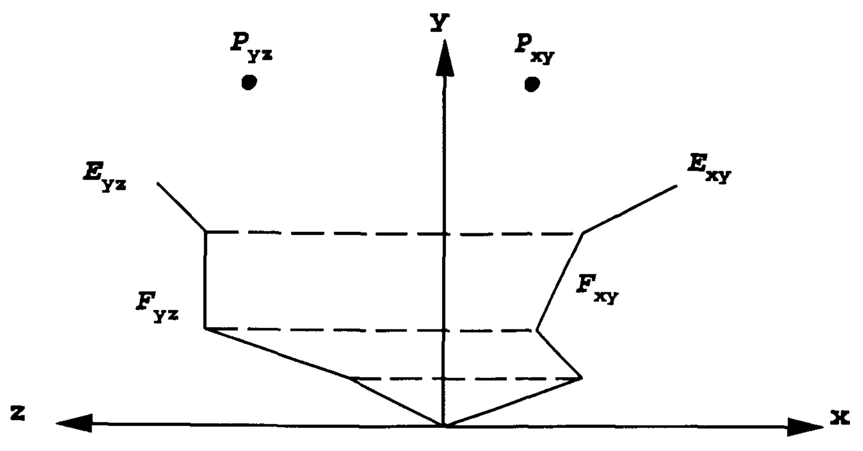

Figure 5.6: Initial Planar Projections of F

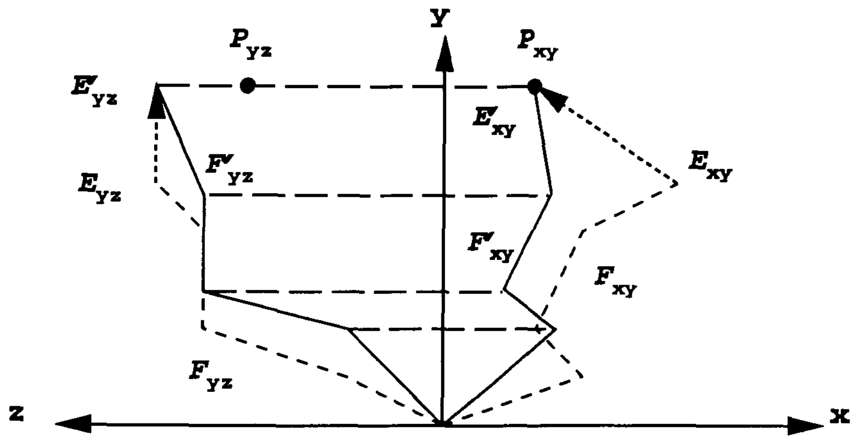

Figure 5.7: Projections of F after Manipulation of F_{xy}

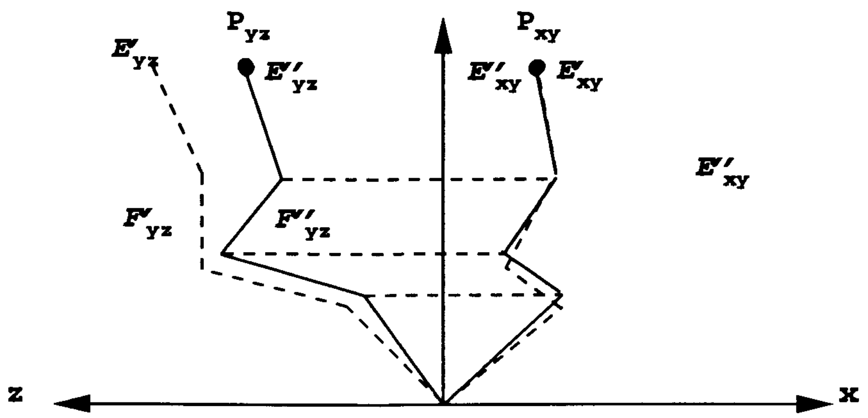

Figure 5.8: Projections of F after Manipulation of F_{yz}

x-y plane. Thus lengths of corresponding segments of F_{xy} and F'_{xy} are the same. The changes when applied to F, however, do change the projection F_{yz}. Lengths of the segments as well as joint angle values of F_{yz} change, but the z coordinates of all joints remain the same. At the end of the inverse kinematic manipulation of F_{xy}, the projection of F on the y-z plane is recalculated. Let the new projections of F_{yz} and E_{yz} be F'_{yz} and E'_{yz} respectively (see Figure 5.7). The y coordinate of E'_{yz} is the same as P_{yz}.

Now, the inverse kinematic control algorithm is invoked on F'_{yz}. At the end of this step the end effector is at P_{yz}. The change in joint angle values of F'_{yz} induces changes in the projection F'_{xy}, but the x coordinates of all joints and x and y coordinates of the end effector E'_{xy} remain fixed (see Figure 5.8). The end effector of F is at the desired point P at the end of this step. The projections F'_{xy} and E'_{xy} are updated to F''_{xy} and E''_{xy} respectively. The figure F is drawn by a traversal of the chain from the root to the end effector. F''_{yz} and E''_{yz} denote the final configuration of F'_{yz} and E'_{yz} respectively.

There is a clear computational advantage in this apparently circuitous way of animating a three dimensional figure. Each iteration of the inverse kinematic control algorithm involves $O(m^2)$ operations, where m is the number of degrees of freedom. The technique outlined above reduces the problem with $2n$ degrees of freedom into two smaller problems with n degrees of freedom, where n is the number of segments in the chain. However, in situations where the point P is near the surface of the envelope of the accessible space of the end effector, more than one iteration of this technique may be required to get the end effector to the goal.

5.3 Implementational Details

The notion of trees, subtrees and lists recurs many times in the preceding discussion. Indeed, the implementation of the system described earlier contains different types of trees, lists and lists of trees. This lends itself to an object oriented design. A set of generic container classes pertaining to lists and trees was created to allow reusability of the code and to make interaction with various program entities uniform. These generic classes provide very basic services to support functionality of lists and trees such as data insertion and retrieval and different types of list and tree traversal. Specialized classes supporting the notion of an articulated figure, movable trees, lists of movable trees and so on were derived from the generic classes. The derived classes provide services which support the functionality of the underlying entity. These services were built on top of services provided by the generic classes. Objects, which are instances of the derived classes, were used for program development.

The articulated figure class provides services which allow partitioning of the tree into movable and static subtrees and for drawing the figure. These services are invoked to obtain a set of independent subproblems each element of which is an instance of the movable tree class. The movable tree class provides services for querying the following information: joint angle vector, objective function value, the gradient, the Hessian, constraint values, constraint bounds and constraint gradients. In addition there is a service to update the joint angle values.

The driver program for the optimizer gets the object representing the list of movable subtrees. It retrieves a node from the list, retrieves from the node the information required by the optimizer and invokes the optimizer. The optimizer returns the solution to the problem, which is an updated joint angle vector corresponding to the movable subtree represented by the node. The driver program then updates the movable subtree. Updating of the joint angle vector in the movable subtree object is also reflected in the corresponding parts of the articulated figure. After all the nodes in the list of movable subtrees are visited by the driver program, it invokes the drawing routine of the articulated figure, which results in the updated picture on the screen. These iterations are carried out untill all the subproblems are solved.

As is clear from the description given above, the optimizer is completely insulated from the design of the application. Thus future changes in the application will not require any modification to the optimizer. The class design has been done using C++. The drawing routines embed Silicon Graphic Graphics Library calls in the C++ code. The FORMS library is used for the user interface design.

The optimizer uses routines from packages like BLAS [LHKK79], LAPACK [ABB+92], LSSOL [GMSW86a] and NPSOL [GMSW86b]. BLAS and LAPACK provide general linear algebra routines and NPSOL and and LSSOL are a collection of optimization related routines. All packages are written in FORTRAN–77 and are an excellent repository of stable and robust code for numerical computations.

Chapter 6

Conclusions

This thesis has presented the development and implementation of a system which allows direct manipulation of articulated figures, represented as planar, hierarchical, skeletal structures, for the purpose of creating animations. Applicability for animating three-dimensional figures has been discussed. The system allows for the interactive creation of keyframes (which can later be interpolated suitably to make an animation) and complete animation sequences. Various techniques have been tried in the past for animating articulated figures. A survey of these techniques has been presented and their limitations have been identified. The new approach to inverse kinematics presented here addresses some of these limitations.

Direct manipulation of articulated figures is supported by automating the inverse kinematic calculation of joint angles for placing the end effector at a point in space. Reformulation of the inverse kinematic problem as an optimization problem, where a function of the distance between the end effector and the goal is the objective function to be minimized, has been described. A particular contribution of this work is the range of constraints which can be incorporated. Constraints can be used to impose limits on the possible values for the joint angles; Another class of constraints reduces the effort required for the creation of keyframes. Previous attempts at articulated figure animation have been limited to providing direct manipulation of the figure with only one end effector at a time. Our approach takes the concept of direct manipulation a step further by making it possible to manipulate multiple end effectors simultaneously by associating trajectories with them. The end effectors follow associated trajectories over a period of time, thereby creating an animation sequence.

The process of optimization of a function in the presence of constraints requires higher order information about the function and the constraints. Algorithms for efficient procedural evaluation of these quantities have been developed.

Optimization of nonlinear functions is an iterative process where each iteration involves potentially expensive steps of matrix inversion and recalculation of different matrices and vectors. Techniques from the field of nonlinear optimization, which have not been previously used for articulated figure animation, allow efficient inversion and modification of relevant matrices in each iteration by maintaining suitable factors of different matrices. Real-time direct manipulation of articulated figures has been ensured by the use of these techniques and an optimization algorithm which displays super-linear convergence.

6.1 Future Work

We have chosen only a function of the distance between the end effector position and the goal as the objective function which is minimized to provide direct manipulation of an articulated figure. However, a variety of other functions can be defined to accomplish different tasks. For instance a function of the difference in the orientation of the end effector and a user specified vector can be minimized to make the end effector point in a desired direction. A weighted sum of the distance function and the orientation function can be used if both the end effector position and the orientation are important. A kinematic subtree with more than one end effector can be manipulated by using an objective function which is a weighted sum of objective functions associated with each end effector.

Similarly, many different kinds of constraints in addition to those discussed can be incorporated in the system. For example types of constraints which constrain a joint on the articulated figure to lie on a twice-continuously differentiable curve can be defined.

It is easy to show that higher order information about the types of objective functions and constraints discussed above can be derived from the material presented in Chapter 3. Any set of functions and constraints for which the higher order information can be derived are suitable for the optimization algorithm used in this research. Thus, an optimization approach to inverse kinematics is a very powerful paradigm which can very conveniently and uniformly handle various kinds of situations that can arise in the direct manipulation of articulated figures.

However, for any set of goals and constraints, an articulated figure can have more than one solution. An optimization algorithm selects the *nearest* solution in the solution space. The nearest solution however, may not result in the *best* posture for the articulated figure. The best posture for a given set of goals and constraints depends on the entity which the figure represents. Incorporation of entity specific knowledge in the form of constraints, as mentioned in Chapter 6, can greatly enhance the functionality of the system.

This implementation represents trajectories as a list of points. This, however, makes it difficult to modify trajectories. Use of splines as a representation for trajectories offers a very attractive and flexible solution to this problem.

Appendix A

Notation

Vectors and matrices are shown in bold with lower case and upper case letters respectively. All vectors are column vectors. Superscript T is used to denote the transpose of a vector or a matrix. An element of a vector is represented by the same letter as the vector, and is italicized. An element of a matrix is represented with a lower case, italicized letter which is the same as that used for the matrix. Subscripts are used to denote the order of the element within a vector or a matrix. For example, g_i is the i^{th} element of vector $\mathbf{g}$ and $h_{i,j}$ is the element in the i^{th} row and j^{th} column of matrix $\mathbf{H}$.

$\|\| \cdot \|\|_2$	2-norm of a vector
n	number of variables
$\boldsymbol{\theta}$	$n \times 1$ vector of variables of optimization
$\boldsymbol{\theta}^*$	$n \times 1$ vector of variables at the optimal point
$\mathcal{F}(\boldsymbol{\theta})$	the objective function

$\mathbf{g}(\boldsymbol{\theta}) = \nabla \mathcal{F}(\boldsymbol{\theta})$	$n \times 1$ vector of first partial derivatives of $\mathcal{F}(\boldsymbol{\theta})$, called the *gradient*

$$\mathbf{g}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial}{\partial \theta_1}(\mathcal{F}(\boldsymbol{\theta})) \\ \frac{\partial}{\partial \theta_2}(\mathcal{F}(\boldsymbol{\theta})) \\ \vdots \\ \frac{\partial}{\partial \theta_n}(\mathcal{F}(\boldsymbol{\theta})) \end{bmatrix}$$

$\mathbf{H}(\boldsymbol{\theta}) = \nabla^2 \mathcal{F}(\boldsymbol{\theta})$	$n \times n$ symmetric matrix of second partial derivatives of $\mathcal{F}(\boldsymbol{\theta})$ called the *Hessian*

$$\mathbf{H}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial}{\partial \theta_1 \partial \theta_1}(\mathcal{F}(\boldsymbol{\theta})) & \frac{\partial}{\partial \theta_1 \partial \theta_2}(\mathcal{F}(\boldsymbol{\theta})) & \cdots & \frac{\partial}{\partial \theta_1 \partial \theta_n}(\mathcal{F}(\boldsymbol{\theta})) \\ \frac{\partial}{\partial \theta_2 \partial \theta_1}(\mathcal{F}(\boldsymbol{\theta})) & \frac{\partial}{\partial \theta_2 \partial \theta_2}(\mathcal{F}(\boldsymbol{\theta})) & \cdots & \frac{\partial}{\partial \theta_2 \partial \theta_n}(\mathcal{F}(\boldsymbol{\theta})) \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial}{\partial \theta_n \partial \theta_1}(\mathcal{F}(\boldsymbol{\theta})) & \frac{\partial}{\partial \theta_n \partial \theta_2}(\mathcal{F}(\boldsymbol{\theta})) & \cdots & \frac{\partial}{\partial \theta_n \partial \theta_n}(\mathcal{F}(\boldsymbol{\theta})) \end{bmatrix}$$

n_B	number of simple bounds in the constraint set
n_{FX}	number of simple bounds in the working set
$n_{FR} = n - n_{FX}$	number of variables with no corresponding simple bound in the working set
n_L	number of general linear constraints
t_L	number of general linear constraints active at the optimal point

$\mathbf{a}_{Li}^T\boldsymbol{\theta}$	i^{th} general linear constraint
$\mathbf{A}_L$	$n_L \times n$ matrix with $\mathbf{a}_{Li}^T$'s as its rows
$\hat{\mathbf{A}}_L$	$t_L \times n$ matrix corresponding to active linear constraints only
n_N	number of nonlinear constraints
t_N	number of nonlinear constraints active at the optimal point
$c_i(\boldsymbol{\theta})$	i^{th} nonlinear constraint
$\mathbf{c}(\boldsymbol{\theta})$	$n_N \times 1$ vector of nonlinear constraint
$\hat{\mathbf{c}}(\boldsymbol{\theta})$	$t_N \times 1$ vector of active nonlinear constraint only
$\mathbf{a}_{Ni}(\boldsymbol{\theta}) = \nabla c_i(\boldsymbol{\theta})$	$n \times 1$ vector of first partial derivatives of i^{th} nonlinear constraint, called *constraint Gradient*

$$\mathbf{a}_{Ni}(\boldsymbol{\theta}) = \begin{bmatrix} \dfrac{\partial}{\partial\theta_1}(c_i(\boldsymbol{\theta})) \\ \dfrac{\partial}{\partial\theta_2}(c_i(\boldsymbol{\theta})) \\ \vdots \\ \dfrac{\partial}{\partial\theta_n}(c_i(\boldsymbol{\theta})) \end{bmatrix}$$

$\mathbf{A}_N(\boldsymbol{\theta})$	$n_N \times n$ matrix with nonlinear constraint gradients $\mathbf{a}_{Ni}^T$ as its rows
$\hat{\mathbf{A}}_N(\boldsymbol{\theta})$	$t_N \times n$ matrix corresponding to $\hat{\mathbf{c}}(\boldsymbol{\theta})$
$t = t_L + t_N$	number of active general constraints

$m = n_{FX} + t_L + t_N$	total number of constraints in the working set
$\hat{\mathbf{C}}$	$m \times n$ matrix of active constraints
$\hat{\mathbf{A}}(\boldsymbol{\theta})$	$t \times n$ matrix of active general constraint gradients
$\mathbf{A}_{FX}$	$t \times n_{FX}$ partition of $\hat{\mathbf{A}}$ corresponding to n_{FX} variables with simple bounds in the working set
$\mathbf{A}_{FR}$	$t \times n_{FR}$ matrix of rest of the columns of $\hat{\mathbf{A}}$
$z = n_{FR} - t = n - m$	dimension of Null space of $\mathbf{A}_{FR}$ and $\hat{\mathbf{C}}$
$\mathbf{Z}$	$n \times z$ matrix whose columns are orthogonal to the rows of $\hat{\mathbf{C}}$
$\mathbf{Z}_{FR}$	$n_{FR} \times z$ matrix whose columns are orthogonal to the rows of $\hat{\mathbf{A}}$
$\mathbf{Y}$	$n \times m$ matrix whose columns are basis for the rows of $\hat{\mathbf{C}}$
$\mathbf{Y}_{FR}$	$n_{FR} \times t$ matrix whose columns are are basis for rows of $\hat{\mathbf{A}}$
$\mathbf{Q}_{FR} = \left[\ \mathbf{Z}_{FR} \mid \mathbf{Y}_{FR}\ \right]$	$n_{FR} \times n_{FR}$ matrix composed of $\mathbf{Z}_{FR}$ and $\mathbf{Y}_{FR}$
$\mathbf{Q} = \left[\ \mathbf{Z} \mid \mathbf{Y}\ \right]$	$n \times n$ matrix composed of $\mathbf{Z}$ and $\mathbf{Y}$
$\boldsymbol{\lambda}$	$m \times 1$ vector of *Lagrange multipliers*
$\boldsymbol{\lambda}_L$	$t_L \times 1$ vector of *Lagrange multipliers* corresponding to active general linear constraints
$\boldsymbol{\lambda}_N$	$t_N \times 1$ vector of *Lagrange multipliers* corresponding to active nonlinear constraints

$\boldsymbol{\lambda}^*$	$m \times 1$ vector of *Lagrange multipliers* at the optimal point
$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda})$	*Lagrangian* of the objective function
$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \mathbf{s})$	*Augmented Lagrangian* of the objective function
$\mathbf{W}$	Hessian of the Lagrangian function
$\mathbf{W}_Q = \mathbf{Q}^T\mathbf{W}\mathbf{Q}$	Hessian of the Lagrangian function projected on the matrix $\mathbf{Q}$
$\mathbf{W}_{FR}$	$n_{FR} \times n_{FR}$ submatrix of $\mathbf{W}$ corresponding to variables not with simple bounds in the working set
$\mathbf{W}_Z = \mathbf{Z}_{FR}^T\mathbf{W}_{FR}\mathbf{Z}_{FR}$	$z \times z$ matrix resulting from projecting $\mathbf{W}_{FR}$ on $\mathbf{Z}_{FR}$

Appendix B

Proofs

B.1 The Notation

Formulae for *SigmaTheta, SigmaCos, SigmaSin*, K_x and K_y defined in Chapter 3 are repeated here for ease of reference.

$$\theta_{w,v} = \begin{cases} \sum_{i=v}^{w} \theta_i & \text{for } w \geq v \\ 0 & \text{otherwise} \end{cases} \tag{B.1}$$

$$S^n_{w,v} = \begin{cases} \sum_{i=w}^{n} (a_i \sin(\theta_{i,v})) & \text{for} n \geq w \geq v \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{B.2}$$

$$C^n_{w,v} = \begin{cases} \sum_{i=w}^{n} (a_i \cos(\theta_{i,v})) & \text{for} n \geq w \geq v \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{B.3}$$

$$K_x = C^n_{1,1} - x_P \tag{B.4}$$

$$K_y = S^n_{1,1} - y_P \tag{B.5}$$

B.2 Proofs

B.2.1 Proof of Corollary 1

By definition of $C^n_{i,1}$

$$\begin{aligned} C^n_{i,1} &= \left(\sum_{k=i}^{n} a_k \cos(\theta_{k,1})\right) \\ &= a_i \cos(\theta_{i,1}) + a_{i+1}\cos(\theta_{i+1,1}) + \cdots + a_n \cos(\theta_{n,1}) \\ &= \sum_{k=i}^{i} a_k \cos(\theta_{k,1}) + \sum_{k=i+1}^{i+1} a_k \cos(\theta_{k,1}) + \cdots \sum_{k=n}^{n} a_k \cos(\theta_{k,1}) \end{aligned}$$

$$\boxed{C^n_{i,1} = C^i_{i,1} + C^{i+1}_{i+1,1} + \cdots + C^n_{n,1}}$$

The proof for $S^n_{i,1}$ is given in exactly the same way.

B.2.2 Proof of Corollary 2

By definition of $C^n_{i,1}$

$$\begin{aligned} C^n_{i,1} &= \left(\sum_{k=i}^{n} a_k \cos(\theta_{k,1})\right) \\ &= a_i \cos(\theta_{i,1}) + a_{i+1}\cos(\theta_{i+1,1}) + \cdots + a_n \cos(\theta_{n,1}) \\ &= \sum_{k=i}^{i} a_k \cos(\theta_{k,1}) + \sum_{k=i+1}^{n} a_k \cos(\theta_{k,1}) \end{aligned}$$

$$\boxed{C^n_{i,1} = C^i_{i,1} + C^n_{i+1,1}}$$

The proof for $S^n_{i,1}$ is given in exactly the same way.

B.2.3 Proof of Theorem 1

Case I: $u > w$

$$\begin{aligned}
\frac{\partial}{\partial\theta_u}(S^n_{w,v}) &= \frac{\partial}{\partial\theta_u}\Bigg(\overbrace{a_w \sin(\theta_v + \cdots + \theta_w) + \cdots + a_{u-1}\sin(\theta_v + \cdots + \theta_{u-1})}^{\text{terms independent of } \theta_u} \\
&\quad + \underbrace{a_u \sin(\theta_v + \cdots + \theta_u) + \cdots + a_n \sin(\theta_v + \cdots + \theta_u + \cdots + \theta_n)}_{\text{terms containing } \theta_u}\Bigg) \\
&= 0 + a_u \cos(\theta_v + \cdots + \theta_u) + \cdots + a_n \cos(\theta_v + \cdots + \theta_u + \cdots + \theta_n) \\
&= a_u \cos(\theta_{u,v}) + \cdots + a_n \cos(\theta_{n,v}) \\
&= \sum_{i=u}^{n}(a_i \cos(\theta_{i,v})) \\
&= C^n_{u,v}
\end{aligned}$$

If $w > u$, all terms of $S^n_{w,v}$ will contain θ_u and hence all will be differentiable with respect to θ_u

Case II: $w > u$

$$\begin{aligned}
\frac{\partial}{\partial\theta_u}(S^n_{w,v}) &= \frac{\partial}{\partial\theta_u}(a_w \sin(\theta_v + \cdots + \theta_u + \cdots + \theta_w) + \cdots + \\
&\quad a_n \sin(\theta_v + \cdots + \theta_u + \cdots + \theta_w + \cdots + \theta_n)) \\
&= a_w \cos(\theta_{w,v}) + \cdots + a_n \cos(\theta_{n,v}) \\
&= \sum_{i=w}^{n}(a_i \cos(\theta_{i,v}))
\end{aligned}$$

$$= C^n_{w,v}$$

Combining Case I and II, we get

$$\boxed{\frac{\partial}{\partial \theta_u}(S^n_{w,v}) = C^n_{\max(u,w),v}} \tag{B.6}$$

Similarly, it can be proved that

$$\boxed{\frac{\partial}{\partial \theta_u}(S^n_{w,v}) = C^n_{\max(u,w),v}} \tag{B.7}$$

B.2.4 The Objective Function

The objective function as given by Equation 3.11 is shown below:

$$\begin{aligned}\mathcal{F}(\boldsymbol{\theta}) &= (a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) \cdots a_n \cos(\theta_1 + \theta_2 \cdots \theta_n) - x_P)^2 + \\ &\quad (a_1 \sin(\theta_1) + a_2 \sin(\theta_1 + \theta_2) \cdots a_n \sin(\theta_1 + \theta_2 \cdots \theta_n) - y_P)^2\end{aligned}$$

Using the definition of SigmaTheta (Equation B.1) in the equation given above we get:

$$\begin{aligned}\mathcal{F}(\boldsymbol{\theta}) &= (a_1 \cos(\theta_{1,1}) + a_2 \cos(\theta_{2,1}) + \cdots a_n \cos(\theta_{n,1}) - y_P)^2 \\ &\quad (a_1 \sin(\theta_{1,1}) + a_2 \sin(\theta_{2,1}) + \cdots a_n \sin(\theta_{n,1}) - y_P)^2 \\ &= (\sum_{i=1}^{n}(a_i \cos(\theta_{i,1})) - x_P)^2 + (\sum_{i=1}^{n}(a_i \sin(\theta_{i,1})) - y_P)^2\end{aligned}$$

Furthermore using the definitions of SigmaSin and SigmaCos (Equations B.2 and B.3) in the equation above, we get:

$$\boxed{\mathcal{F}(\boldsymbol{\theta}) = (C^n_{1,1} - x_P)^2 + (S^n_{1,1} - y_P)^2} \tag{B.8}$$

Substituting K_x for $C^n_{1,1} - x_P$ and K_y for $S^n_{1,1} - y_P$ in the equation above we get:

$$\boxed{\mathcal{F}(\boldsymbol{\theta}) = K_x^2 + K_y^2} \tag{B.9}$$

B.2.5 The Gradient

The Gradient $\mathbf{g}(\boldsymbol{\theta})$ of $\mathcal{F}(\boldsymbol{\theta})$ is a vector of first partial derivatives of $\mathcal{F}(\boldsymbol{\theta})$ and is given by the following equation:

$$\mathbf{g}(\boldsymbol{\theta}) = \nabla\mathcal{F}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial}{\partial\theta_1}(\mathcal{F}(\boldsymbol{\theta})) \\ \frac{\partial}{\partial\theta_2}(\mathcal{F}(\boldsymbol{\theta})) \\ \vdots \\ \frac{\partial}{\partial\theta_i}(\mathcal{F}(\boldsymbol{\theta})) \\ \vdots \\ \frac{\partial}{\partial\theta_n}(\mathcal{F}(\boldsymbol{\theta})) \end{bmatrix} \tag{B.10}$$

First partial derivative of g_i of $\mathcal{F}(\boldsymbol{\theta})$ (Equation B.8) with respect to θ_i, $g_i = \frac{\partial}{\partial\theta_i}(\mathcal{F}(\boldsymbol{\theta}))$, is given as follows:

$$\begin{aligned} \frac{\partial}{\partial\theta_i}(\mathcal{F}(\boldsymbol{\theta})) &= \frac{\partial}{\partial\theta_i}((C^n_{1,1} - x_P)^2 + (S^n_{1,1} - y_P)^2) \\ &= 2(C^n_{1,1} - x_P)\frac{\partial}{\partial\theta_i}(C^n_{1,1} - x_P) + 2(S^n_{1,1} - y_P)\frac{\partial}{\partial\theta_i}(S^n_{1,1} - y_P) \end{aligned}$$

using Equations B.6 and B.7 we get

$$= 2(C^n_{1,1} - x_P)(-S^n_{i,1} - 0) + 2(S^n_{1,1} - y_P)(C^n_{i,1} - 0)$$

rearranging terms in the Equation above we get:

$$\boxed{\frac{\partial}{\partial\theta_i}(\mathcal{F}(\boldsymbol{\theta})) = 2((S^n_{1,1} - x_P)C^n_{i,1} - (C^n_{1,1} - y_P)S^n_{i,1})} \tag{B.11}$$

substituting K_x and K_y, we get

$$\boxed{\frac{\partial}{\partial\theta_i}(\mathcal{F}(\boldsymbol{\theta})) \;=\; 2(K_y C^n_{i,1} - K_x S^n_{i,1})} \tag{B.12}$$

Substituting Equation B.12 in Equation B.10 we get:

$$\mathbf{g}(\boldsymbol{\theta}) = \nabla\mathcal{F}(\boldsymbol{\theta}) = \begin{bmatrix} 2(K_y C^n_{1,1} - K_x S^n_{1,1}) \\ 2(K_y C^n_{2,1} - K_x S^n_{2,1}) \\ \vdots \\ 2(K_y C^n_{w,1} - K_x S^n_{w,1}) \\ \vdots \\ 2(K_y C^n_{n,1} - K_x S^n_{n,1}) \end{bmatrix} \tag{B.13}$$

B.2.6 The Hessian

The Hessian $\mathbf{H}(\boldsymbol{\theta})$ of function $\mathcal{F}(\boldsymbol{\theta})$ is an $n \times n$ matrix of second partial derivatives of $\mathcal{F}(\boldsymbol{\theta})$ and is given by:

$$\mathbf{H}(\boldsymbol{\theta}) = \nabla^2\mathcal{F}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial^2}{\partial{\theta_1}^2}(\mathcal{F}(\boldsymbol{\theta})) & \frac{\partial^2}{\partial\theta_1\partial\theta_2}(\mathcal{F}(\boldsymbol{\theta})) & \cdots & \frac{\partial^2}{\partial\theta_1\partial\theta_n}(\mathcal{F}(\boldsymbol{\theta})) \\ \frac{\partial^2}{\partial\theta_2\partial\theta_1}(\mathcal{F}(\boldsymbol{\theta})) & \frac{\partial^2}{\partial\theta_2^2}(\mathcal{F}(\boldsymbol{\theta})) & \cdots & \frac{\partial^2}{\partial\theta_2\partial\theta_n}(\mathcal{F}(\boldsymbol{\theta})) \\ \vdots & \vdots & \vdots\cdots & \vdots \\ \frac{\partial^2}{\partial\theta_n\partial\theta_1}(\mathcal{F}(\boldsymbol{\theta})) & \frac{\partial^2}{\partial\theta_n\partial\theta_2}(\mathcal{F}(\boldsymbol{\theta})) & \cdots & \frac{\partial^2}{\partial\theta_n^2}(\mathcal{F}(\boldsymbol{\theta})) \end{bmatrix} \tag{B.14}$$

If $h_{i,j}$ is the element in i^{th} row and j^{th} column of $\mathbf{H}(\boldsymbol{\theta})$, then $h_{i,j}$ is the second partial derivative of $\mathcal{F}(\boldsymbol{\theta})$ with respect to θ_i and θ_j. Formula for $h_{i,j}$ given by Equation 3.24 can be proved as follows:

$$\begin{aligned} h_{i,j} &= \frac{\partial^2}{\partial\theta_i\partial\theta_j}(\mathcal{F}(\boldsymbol{\theta})) \\ & \frac{\partial}{\partial\theta_i}(\frac{\partial}{\partial\theta_j}(\mathcal{F}(\boldsymbol{\theta}))) \end{aligned}$$

Substituting for $\frac{\partial}{\partial\theta_j}(\mathcal{F}(\boldsymbol{\theta}))$ from Equation B.11, we get:

$$\begin{aligned}
h_{i,j} &= 2\left(\frac{\partial}{\partial\theta_i}\left(2((S^n_{1,1} - x_P)C^n_{j,1} - (C^n_{1,1} - y_P)S^n_{j,1})\right)\right) \\
h_{i,j} &= 2\left(\left(\frac{\partial}{\partial\theta_i}(S^n_{1,1} - x_P)\right)C^n_{j,1} + (S^n_{1,1} - x_P)\left(\frac{\partial}{\partial\theta_i}(C^n_{j,1})\right) - \right. \\
&\qquad \left.\left(\frac{\partial}{\partial\theta_i}(C^n_{1,1} - y_P)\right)S^n_{j,1} - (C^n_{1,1} - x_P)\left(\frac{\partial}{\partial\theta_i}(S^n_{j,1})\right)\right) \\
&= 2(C^n_{i,1}C^n_{j,1} - (S^n_{1,1} - x_P)S^n_{i,1} + S^n_{i,1}S^n_{j,1} - (C^n_{1,1} - y_P)C^n_{j,1})
\end{aligned}$$

substituting K_x and K_y above

$$= 2(C^n_{i,1}C^n_{j,1} - K_yS^n_{i,1} + S^n_{i,1}S^n_{j,1} - K_xC^n_{j,1})$$

rearranging the terms results in:

$$\boxed{h_{i,j} = 2\left((C^n_{j,1} - K_x)C^n_{i,1} + ((S^n_{j,1} - K_y)S^n_{i,1}\right)} \tag{B.15}$$

B.2.7 Non-linear Constraints

Constraints required to pin the end effector of an n-segment kinematic chain to a point P in space, with x_P and y_P as its x and y coordinates respectively, are given as follows:

$$c_x = \begin{array}{l} a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) + \cdots + \\ a_n \cos(\theta_1 + \theta_2 + \cdots + \theta_n) - x_P \end{array} = 0$$

$$c_y = \begin{array}{l} a_1 \sin(\theta_1) + a_2 \sin(\theta_1 + \theta_2) + \cdots + \\ a_n \sin(\theta_1 + \theta_2 + \cdots + \theta_n) - x_P \end{array} = 0$$

using the definition of SigmaTheta (see Equation B.1) in Equations above:

$$\begin{aligned} c_x &= a_1 \cos(\theta_{1,1}) + a_2 \cos(\theta_{2,1}) + \cdots + \cos(\theta_{n,1}) - x_P &= 0 \\ c_x &= a_1 \sin(\theta_{1,1}) + a_2 \sin(\theta_{2,1}) + \cdots + \sin(\theta_{n,1}) - x_P &= 0 \end{aligned}$$

using the definitions of SigmaCos and SigmaSin in the Equation above:

$$\boxed{\begin{aligned} c_x &= C^n_{1,1} - x_P &= 0 \\ c_y &= S^n_{1,1} - y_P &= 0 \end{aligned}} \tag{B.16}$$

substituting K_x and K_y above, we get:

$$\boxed{\begin{aligned} c_x &= K_x &= 0 \\ c_y &= K_y &= 0 \end{aligned}} \tag{B.17}$$

B.2.8 Constraint Gradients

The Gradient $\nabla c(\boldsymbol{\theta})$ of $c(\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is an n-vector, is an n-vector of first partial derivatives of $c(\boldsymbol{\theta})$ and is given by the following equation:

$$\nabla c(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial}{\partial \theta_1}(c(\boldsymbol{\theta})) \\ \frac{\partial}{\partial \theta_2}(c(\boldsymbol{\theta})) \\ \vdots \\ \frac{\partial}{\partial \theta_i}(c(\boldsymbol{\theta})) \\ \vdots \\ \frac{\partial}{\partial \theta_n}(c(\boldsymbol{\theta})) \end{bmatrix} \tag{B.18}$$

Using Equation B.16 first partial derivative of $c_x(\boldsymbol{\theta})$ and $c_y(\boldsymbol{\theta})$ with respect to θ_i are given as follows:

$$\begin{aligned} \frac{\partial}{\partial \theta_i}(c_x(\boldsymbol{\theta})) &= \frac{\partial}{\partial \theta_i}(C^n_{1,1} - x_P) \\ \frac{\partial}{\partial \theta_i}(c_y(\boldsymbol{\theta})) &= \frac{\partial}{\partial \theta_i}(S^n_{1,1} - y_P) \end{aligned}$$

Using results given by Equations B.7 and B.6, above equations become:

$$\boxed{\begin{aligned} \frac{\partial}{\partial \theta_i}(c_x(\boldsymbol{\theta})) &= -S^n_{i,1} \\ \frac{\partial}{\partial \theta_i}(c_y(\boldsymbol{\theta})) &= C^n_{i,1} \end{aligned}} \tag{B.19}$$

using Equations B.19 and B.18 the Gradients of c_x and c_y can be written as:

$$\nabla c_x(\boldsymbol{\theta}) = - \begin{bmatrix} S^n_{1,1} \\ S^n_{2,1} \\ \vdots \\ S^n_{n,1} \end{bmatrix} \tag{B.20}$$

$$\nabla c_y(\boldsymbol{\theta}) = \begin{bmatrix} C^n_{1,1} \\ C^n_{2,1} \\ \vdots \\ C^n_{n,1} \end{bmatrix} \tag{B.21}$$

Bibliography

[ABB+92] E. Anderson, J. Bai, C. Bischof, J. W. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Oustrouchov, and D. Sorensen. *LAPACK User's Guide*. SIAM, Philadelphia, 1992.

[BB88] R. Barzel and A. Barr. A modelling system based on dynamic constraints. *Computer Graphics*, 22(4):179–188, August 1988.

[BC89] A. Bruderlin and T. Calvert. Goal directed dynamic animation of human walking. *Computer Graphics*, 23(3):233–242, July 1989.

[BN88] L. Brotman and N. Netravali. Motion interpolation by optimal control. *Computer Graphics*, 22(4):309–315, August 1988.

[BPW93] N. I. Badler, C. B. Phillips, and B. L. Webber. *Simulating Humans: Computer Graphics Animation and Control.* Oxford University Press, 1993.

[Bro88] C. G. Broyden. The convergence of a class of double-rank minimization algorithms. *J. Inst. Maths. Applics.*, 6:76–90, August 1988.

[Coh92] M. F. Cohen. Interactive spacetime control for animation. *Computer Graphics*, 26(2), July 1992.

[DH55] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *ASME Journal of Applied Mechanics*, 23:215–221, June 1955.

[Fle70] R. Fletcher. A new approach to variable metric algorithm. *Computer Journal*, 13:317–322, 1970.

[Fle87] R. Fletcher. *Practical Methods of Optimization.* John Wiley and Sons, 1987.

[FP63] R. Fletcher and M. J. D. Powell. A rapidly convergent descent method for minimization. *Computer Journal*, 6:163–168, 1963.

[GGMS74] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders. Methods for updating matrix factorizations. *Math. Comput.*, 28:505–535, 1974.

[Gir91] M. Girard. Constrained optimization of articulated animal movement in computer animation. In *Making Them Move: Mechanics, Control and Animation of Articulated Figures*, pages 209–229. Morgan Kaufmann Publishers Inc., San Mateo, Ca., 1991.

[GM79] P. E. Gill and W. Murray. The computation of lagrange multiplier estimates for constrained minimization. *Mathematical Programming*, 17:32–60, 1979.

[GM85] M. Girard and A. Maciejewski. Computational modelling for the computer animation of legged figures. *Computer Graphics*, 19(3):263–270, July 1985.

[GMSW84] P. E. Gill, W. Murray, M. Saunders, and M. Wright. Procedures for optimization problems with a mixture of bounds and general linear constraints. *ACM Transactions on Mathematical Software*, 10(3):282–298, September 1984.

[GMSW85] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Software and its relationship to methods. In P. T. Boggs, R. H. Byrd, and R. B. Schnabel, editors, *Numerical Optimization 1984*, pages 139–159. SIAM, Philadelphia, 1985.

[GMSW86a] P. E. Gill, W. Murray, M. Saunders, and M. Wright. User's guide for LSSOL (version 1.0). Technical Report SOL 86-1, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, Ca. 94305, 1986.

[GMSW86b] P. E. Gill, W. Murray, M. Saunders, and M. Wright. User's guide for NPSOL (version 4.0): A FORTRAN package for nonlinear programming. Technical Report SOL 86-2, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, Ca. 94305, 1986.

[GMSW92] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Some theoretical properties of an augmented Lagrangian merit function. In P. M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 101–128. North Holland, North Holland, 1992.

[GMW81] P. E. Gill, W. Murray, and M. Wright. *Practical Optimization.* Academic Press, 1981.

[Gol70] D. Goldfarb. A family of variable metric methods derived by variational means. *Mathematics of Computation*, 24:23–26, 1970.

[Hes75] M. R. Hestenes. *Optimization Theory, The Finite Dimensional Case.* John Wiley and Sons, New York, 1975.

[Hes80] M. R. Hestenes. Augmentability in optimization theory. *Journal of Optimization Theory and Applications*, 32:427–440, 1980.

[IC87] P. Isaacs and M. F. Cohen. Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics. *Computer Graphics*, 21(4):215–224, July 1987.

[KB84] D. Kochanek and R. Bartels. Interpolating splines with local tension, continuity and bias control. *Computer Graphics*, 18(3):33–41, July 1984.

[KGV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–679, 1983.

[Las87] J. Lasseter. Principles of traditional animation applied to 3-D computer animation. *Computer Graphics*, 21(4):35–44, July 1987.

[LHKK79] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Transactions on Mathematical Software*, 5:308–323, 1979.

[Loo72] F. A. Lootsma. A survey of methods for solving constrained optimization problems via unconstrained minimization. In *Numerical Methods for Nonlinear Optimization*, pages 313–347. Academic Press, London and New York, 1972.

[LWZB90] P. Lee, S. Wei, J. Zhao, and N. Badler. Strength guided motion. *Computer Graphics*, 24(4):253–262, August 1990.

[PB91] C. Phillips and N. I. Badler. Interactive behaviors for bipedal articulated figures. *Computer Graphics*, 25(4):359–362, 1991.

[Pix86] Pixar. Luxo Jr. (film), 1986.

[Pow70] M. J. D. Powell. A new algorithm for unconstrained optimization. In J.B. Rosen, O. L. Mangasarian, and K. Ritter, editors, *Nonlinear Programming*, pages 31–65. Academic Press, London and New York, 1970.

[Pow74] M. J. D. Powell. Variable metric methods for constrained optimization. In P. E. Gill and W. Murray, editors, *Numerical Methods for Constrained Optimization*, pages 1–28. Academic Press, London and New York, 1974.

[PZB90] C. Phillips, J. Zhao, and N. I. Badler. Interactive real-time articulated figure manipulation using multiple kinematic constraints. *Computer Graphics*, 24(2):245–250, 1990.

[RG91] H. Rijpkema and M. Girard. Computer animation of hands and grasping. *Computer Graphics*, 25(4):339–348, July 1991.

[RK72] J. B. Rosen and J. Kreuser. A gradient projection algorithm for non-linear constraints. In F. A. Lootsma, editor, *Numerical Methods for Nonlinear Optimization*, pages 297–300. Academic Press, London and New York, 1972.

[Ros60] J. B. Rosen. The gradient projection method for nonlinear programming: Part I: linear constraints. *SIAM Journal on Applied Mathematics*, 8:181–217, 1960.

[Ros61] J. B. Rosen. The gradient projection method for nonlinear programming: Part II: nonlinear constraints. *SIAM Journal on Applied Mathematics*, 9:514–532, 1961.

[Rya74] D. M. Ryan. Penalty and barrier functions. In *Numerical Methods for Constrained Optimization*, pages 175–190. Academic Press, London and New York, 1974.

[SGWM93] T. W. Sederberg, P. Gao, G. Wang, and H. Mu. 2-D shape blending: An intrinsic solution of the vertex path problem. *SIGGRAPH 93 Conference Proceedings*, pages 15–18, August 1993.

[Sha70] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24:647–657, 1970.

[Sho85] K. Shoemake. Animating rotation with quaternion curves. *Computer Graphics*, 19(3):245–254, July 1985.

[Ste83] G. Stern. BBop - a program for 3-dimensional animation. *Nicograph Proceedings*, pages 403–404, 1983.

[Stu84] D. Sturman. Interactive key frame animation of 3-D articulated models. *Graphics Interface*, pages 35–40, 1984.

[Sur92a] M. C. Surles. An algorithm with linear complexity for interactive, physically-based modelling of large proteins. *Computer Graphics*, 26(2):221–230, July 1992.

[Sur92b] M. C. Surles. *Techniques For Interactive Manipulation of Graphical Protein Models.* PhD thesis, University of North Carolina at Chapel Hill, 1992.

[SV89] M. Spong and M. Vidyasagar. *Robot Dynamics and Control.* John Wiley and Sons, 1989.

[TBM$^+$88] D. E. Thompson, W. L. Buford Jr., L. M. Myers, D. J. Giurintano, and J. A. Brewer III. A hand biomechanics workstation. *Computer Graphics*, 22(4):335–343, August 1988.

[Wel93] C. Welman. Inverse kinematics and geometric constraints for articulated figure manipulation. Master's thesis, Simon Fraser University, April 1993.

[WK88] A. Witkin and M. Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, August 1988.

[ZB90] J. Zhao and N. I. Badler. Real time inverse kinematics with joint limits and spatial constraints. Technical Report MS-CIS-90-09, Department of Computer and Information Science, University of Pennsylvania, 1990.