A Perceptually Accurate Model of the Hand

by

Ye Lu

B.A.Sc. (Computing Science) Simon Fraser University, 1995

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE
in the School
of
Engineering Science

SIMON FRASER UNIVERSITY
July 1997

APPROVAL

Name:	Ye Lu
Degree:	Master of Applied Science
Title of Thesis:	A Perceptually Accurate Model of the Hand

Examining Committee: Dr. Kamal Gupta, Associate Professor
Chair

Dr. John C. Dill, Professor
Senior Supervisor

Dr. Tom W. Calvert, Professor
Supervisor

Dr. Ze-Nian Li, Associate Professor
School of Computing Science
External Examiner

Date Approved: July 25, 1997

Abstract

The human hand is used in virtually all aspects of everyday activities involving such tasks as selection, manipulation and communication. It is therefore important to study and understand how it works. However, a significant difficulty facing traditional hand researchers is acquiring accurate 3D data that describes actual human hand movements. To address this problem, a multidisciplinary team of researchers from kinesiology, computer science and engineering was formed to design and implement a Virtual Hand Laboratory to serve as a testbed for future studies of goal-directed human hand movements.

The primary objective of the work described in this thesis is to provide a framework for the collection, estimation and display of hand postures from live 3D data. A fast lookup table-based inverse kinematic algorithm was developed and used to estimate hand postures from real-time 3D data supplied by the Optotrak system, a powerful motion capture device.

The algorithm is executed in two stages: a calibration stage and a run-time stage. The former measures the characteristics of the subject's hand in order to customize the lookup table while the latter uses the table to estimate the subject's hand posture. A polygonal model of the human hand was developed for display of the estimated hand posture. A complete system integrating data collection with estimation and display of hand postures in real-time was designed and implemented to serve as the platform for real-time experiments in the study of goal-directed hand movements.

Dedication

To My Parents

Acknowledgments

I would like to thank my senior supervisor Dr. John C. Dill for his invaluable guidance, enthusiasm and financial support throughout the course of this work. I am also very grateful to my supervisor Dr. Tom Calvert for his helpful comments and insightful suggestions during the research and writing of this thesis. I would also like to thank Dr. Ze-Nian Li for taking the time to be my external examiner.

Many other people have helped and contributed their time to the research of this thesis. My thanks to Dr. Kellogg S. Booth and Jason Harrison of the University of British Columbia for their invaluable comments. Thanks to Sidi Yu for his efforts in the development of the polygonal model of the hand. Thanks to Dr. Kenji Amaya for his inspiration and development of the initial algorithm. Thanks to Dr. Hongsheng Chin for his mathematical expertise as well as his friendship. Thanks to Wan Gong for proofreading the draft of this thesis.

I would also like to take this opportunity to express my gratitutes toward everyone in the Human Motor Systems Laboratory of the School of Kinesiology at Simon Fraser University, especially Dr. Christine MacKenzie, Dr. Evan Graham, Yanqing Wang, Beth Brown and Chris Ivens for their valuable suggestions and help in setting up the equipments for the experiment.

Last but certainly not least, I will always be indebted to my parents, Xiqi Lu and Taiqiong Ma, and my brother Joe Lu, for their support and confidence in me. This thesis would not have been possible without their kindness and encouragement.

This research is supported in part by the NSERC Strategic Grant.

Contents

List of Figures

Chapter 1

Introduction

People use their hands for many everyday activities involving such tasks as selection, manipulation and communication. It is therefore important to study and understand how it works. Accordingly, the study and analysis of goal-directed human hand movement has become an increasingly important topic in the field of human-computer interaction (HCI). With the advent of 3D virtual reality systems, gestural input and full hand pointing are being explored as the input techniques of choice for future computer systems.

A traditional problem that faces researchers is the acquisition and analysis of accurate 3D data describing hand movements. The main purpose of the Virtual Hand Project, conducted by a multidisciplinary team that includes researchers from kinesiology, computer science and engineering, is to design and implement a Virtual Hand Laboratory to serve as a testbed for future studies of goal-directed human hand movements.

The primary objective of this thesis is to provide a framework and a set of tools for the collection, display, estimation and analysis of hand posture from real-time 3D data. In overview, a motion capture device (Optotrak) periodically samples the position of markers on a subject's hand. An inverse kinematic approach is then used to convert this 3D positional data into joint angle data which represents the hand postures. A 3D polygonal model of the hand is used for the display and animation of hand postures.

Since we manipulate the physical world most often and most naturally with our hands, there is a great desire to apply the skills, dexterity, and naturalness of the hand directly to the human-computer interface [40]. A number of research projects in the past few years dealt with precisely this subject. Much of the work has been done in the context of developing virtual environments.

1.1 Motivation

We believe that the next generation computer systems that employ virtual reality (VR) techniques will be the next quantum leap of HCI. This new frontier is currently very young and most work has focused on the development of hardware technology and the custom implementation of specific applications. There is very little research into the higher levels of abstraction that facilitates the composition of new systems.

The mouse is one of the most popular devices used in almost all Graphics User Interfaces (GUIs). It makes direct manipulation of objects on the screen possible by allowing selection, dragging and manipulation of these objects. Nevertheless, the mouse is no substitute for a human hand which people use in just about all their daily tasks. Manipulation of 3D objects on the screen is usually very awkward and unnatural for people using only a mouse. The hand is a much more attractive alternative for a VR system which provides the user with a fully rendered 3D view of the objects to be manipulated. Sturman [39] has discussed the use of whole hands as an input device. He suggested two paradigms: the manipulation paradigm and the sign language paradigm. The manipulation paradigm refers to point, reach and grab interactions. Applying this paradigm, most VR systems use the tip of the index finger as a 3D pointer in order to select items in menus floating in 3D space. This approach is a natural way to use the hand but it ignores the freedom that general hand gesture provides.

The sign language paradigm recognizes hand gestures as a stream of tokens that are similar to signs in sign languages. This visual language can be used to record gesture analytically. The major critique of this paradigm is that the devised gestures may not be easy enough for the user to recall. However, this can be remedied by allowing the user to define the gestures incrementally [41].

To accommodate either of these two major paradigms, the hand movements and postures would have to be accurately measured and analyzed. The complexities of the function and anatomy of the human hand have long been recognized and many studies have been conducted on these topics [2]. Traditional studies of the hand often involve experiments on hands from fresh cadaver specimens. Surgical incisions were made and markers made of different grades of surgical wire were then inserted into the hand to measure the movements of the tendons and muscles. The data gathered from those measurements were then analyzed to derive theoretical biomechanical models of the hand [2, 8]. Although useful for biomechanical analysis and dynamic simulations, these models usually require measurements of the internal forces and torques. These measurements are very hard to obtain using non-intrusive procedures. Because our intent is not to develop a device for accurate dynamic simulation but rather an input device that can easily be used by everyone, methods of measuring the hand postures using only non-intrusive techniques are needed.

A number of devices are available that allow a user to interact with objects in a virtual environment using the hand. The DataGlove developed by Thomas Zimmerman monitors 10 finger joints and the six degrees of freedom of the hand's position and orientation [48]. Physically, the DataGlove consists of a lightweight glove fitted with specially treated optical fibers along the backs of the fingers. Finger flexion bends the fibers, attenuating the light they transmit. The signal strength for each of the fibers is sent to a processor that determines joint angles based on precalibrations for each user. There are numerous other glove based hand input devices such as the Dexterous HandMaster, the Power Glove, the CyberGlove, the Space Glove and so on [40]. Each of these devices has its own strength and weakness with regard to accuracy, speed, and cost. However, a common weakness shared by all of them is that they are too restrictive for the hand to move around the space.

As an alternative, motion tracking devices are non-intrusive and able to measure the hand movements frame by frame with only a small number of markers attached to the hand. The Optotrak system used in the Virtual Hand Laboratory is such a motion capture system. The markers used by the Optotrak system are called infrared emitting diodes (IREDs). The coordinates of these markers are sent by the Optotrak system to a SGI workstation where they are used to compute and display the hand posture.

1.2 The Inverse Kinematics Approach for Hand Posture Estimation

Hand postures are usually represented using finger joint angles and orientation of the palm. A direct method of measuring joint angles would be to place IREDs on all joints of the fingers and simply measure their positions in 3D. The joint angles could then be trivially calculated using elementary methods from analytic geometry. However, it is well known that occlusion problems make monitoring several IREDs simultaneously using the Optotrak system very difficult.

As an alternative to the direct method, we can minimize the number of IREDs used, therefore minimizing the occlusion problem, by only putting one IRED at the tip of each finger and use techniques of inverse kinematics to compute the associated joint angles. Since inverse kinematic calculations are usually difficult and time consuming, the main focus of this research is to provide an efficient real-time inverse kinematic algorithm to calculate the joint angles from which hand posture can be calculated.

1.3 Organization of Thesis

Chapter 2 reviews kinematic methods in general, and discusses their relevance in hand posture estimation and provides an overview of hand anatomy and kinematics. Chapter 3 formally states the inverse kinematics problem and presents some of the commonly used approaches for solving it. In Chapter 4, an efficient real time inverse kinematic algorithm is presented. Chapter 5 describes an experiment that measures the performance of the algorithm and analyzes the results. The conclusion and future work are presented in Chapter 6.

Chapter 2

Hand Kinematics

2.1 Kinematic Methods

Kinematics is that part of the science of motion which treats motion without regard to the forces that cause it. Within the study of kinematics, there are two classes of problems: *forward kinematics* and *inverse kinematics.* In order to study them, we first have to consider the structure of the *kinematic chain.*

2.1.1 Link Description

A kinematic chain may be thought of as a set of rigid bodies connected by joints. These bodies are called *links.* The joints are usually rotational, but may also be prismatic. Each rotational joint allows rotation in 1, 2, or 3 orthogonal directions. This is called the degree of freedom (DOF) of the joint. Any joint with n degrees of freedom may be modeled as n joints of one degree of freedom connected with $n-1$ links of zero length. Therefore, without loss of generality, we only have to consider kinematic chains consisting entirely of joints each having just one degree of freedom. The two ends of the kinematic chain are called the *base* and the *end-effector* respectively. The base of the chain is fixed at one position while the end-effector can move freely around the space.

In order to describe the kinematic chain accurately and effectively, a convention is required. The Denavit-Hartenberg convention[11] establishes a framework for systematic specification of kinematic chains by using four link parameters for each link in the chain. The four link parameters uniquely determines a coordinate frame for each link in the chain. The link parameters are defined as shown in Figure 2.1:

- a_i = the distance from Z_i to Z_{i+1} measured along X_i;
- α_i = the angle between Z_i and Z_{i+1} measured about X_i;
- d_i = the distance from X_{i-1} to X_i measured along Z_i; and
- θ_i = the angle between X_{i-1} and X_i measured about Z_i.

The convention for affixing frames of reference on the links is as follows: the Z-axis of frame i is coincident with the axis of frame i; the origin of frame i is located where the a_i perpendicular intersects the axis of rotation of $i+1$; the X-axis points along a_i in the direction from joint i to joint $i+1$; the Y-axis is selected to complete a right handed coordinate system.

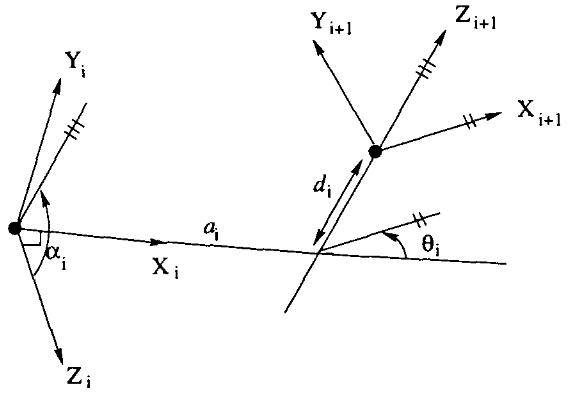

Figure 2.1: The link parameters

To determine the transformation which defines frame i relative to the frame $i-1$, we can define three intermediate frames P, Q and R for each link. Frame R differs from frame i by a rotation of α_i. Frame Q differs from R by a translation a_i. Frame P differs from Q by a rotation θ_i and frame i differs from P by a translation d_i. These transformations can be multiplied together to obtain a transformation matrix from frame $i-1$ to frame i.

2.1.2 Forward Kinematics

Forward kinematics involves finding the position and orientation of the end-effector relative to some coordinate system given a set of joint angles for each joint. Using the link parameters defined in the previous section, we can define a transformation matrix ${}^{i-1}_{i}T$ that transforms a vector in frame $i-1$ to frame i.

$$
{}^{i-1}_{i}T = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -\sin\alpha_{i-1}d_i \\ -\sin\theta_i\sin\alpha_{i-1} & \cos\theta_i\sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}
$$

Once the link parameters are found for each link and the corresponding link frames have been defined, finding the forward kinematic equation is straightforward. The individual link transformations can be multiplied together to find the single transformation ${}^{0}_{n}T$ that relates frame n to frame 0.

$$
{}^{0}_{n}T = {}^{0}_{1}T{}^{1}_{2}T \ldots {}^{n-1}_{n}T \tag{2.2}
$$

The matrix ${}^{0}_{n}T$ represents the last link's position and orientation in the Cartesian space.

For a planar 3 link kinematic chain, the value of the link parameters α_i and d_i are 0, so the transformation matrix ${}^{0}_{3}T$ reduces to

$$
{}^{0}_{3}T = \begin{bmatrix} c_{123} & -s_{123} & 0 & l_1c_1 + l_2c_{12} + l_3c_{123} \\ s_{123} & c_{123} & 0 & l_1s_1 + l_2s_{12} + l_3s_{123} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}
$$

where $s_{1\ldots n}$ and $c_{1\ldots n}$ are shorthand notations for $\sin(\theta_1+\cdots+\theta_n)$ and $\cos(\theta_1+\cdots+\theta_n)$ respectively, and l_1, l_2, and l_3 are the lengths of the links. From (2.3), the planar coordinates x, y and orientation ϕ are clearly

$$x = l_1c_1 + l_2c_{12} + l_3c_{123} \tag{2.4}$$

$$y = l_1s_1 + l_2s_{12} + l_3s_{123} \tag{2.5}$$

$$\phi = \theta_1 + \theta_2 + \theta_3 \tag{2.6}$$

2.1.3 Inverse Kinematics

The inverse kinematics problem is essentially the reverse of the forward kinematics problem. Inverse kinematics involves finding the joint angles of each link of a kinematic chain given the end-effector location and orientation. This problem has been extensively studied in both computer graphics and robotics. Since inverse kinematics is the main focus of this thesis, a detailed description and analysis of the related work will be presented in the next chapter. The following sections will give a brief overview of the application areas of inverse kinematics in both computer graphics and robotics.

Inverse Kinematics in Computer Graphics

The area within computer graphics that makes extensive use of inverse kinematics is computer animation, in particular, the animation of articulated figures. An articulated figure is usually represented by a collection of kinematic chains connected together. Each joint in this articulated structure may have one, two, or three degrees of freedom. The degrees of freedom of an articulated structure increases with its complexity. As an example, a detailed approximation of the human skeleton may have in excess of two hundred DOF. Although well understood traditional animation techniques [25] help animators produce expressive motions in their animation, they require extensive manipulation of the figure to achieve the desired effects. It is obviously a very difficult task to create animation by manipulating joint angles to set up key frames that place end-effectors of certain kinematic chains in desired locations. Multiple iteration of trial and error is generally required to produce the correct result. This approach is certainly very time consuming and error prone.

It is apparent that inverse kinematics offers an attractive solution to the above problem. Instead of letting the animator specify the joint angles that place the end-effector at a desired location, the computer automatically calculates these joint angles from the link configuration and the end-effector location specified by the animator. This technique was used by Girard and Maciejewski [28] to build the PODA system which synthesizes the kinematic model of legged locomotion. Zhao and Badler [47] proposed an algorithm that can incorporate various constraints and solve for simultaneous goals. Welman [45] has presented two very distinct inverse kinematic algorithms suitable for real time manipulation and showed their effectiveness in a powerful interactive editor *LifeForms.* By formulating inverse kinematics into an optimization problem, Bawa [4] has presented an algorithm which uses an iterative nonlinear constrained optimization algorithm for solving the inverse kinematics problem.

Inverse Kinematics in Robotics

The inverse kinematics problem was first extensively studied in the field of robotics. Since computer based robots are usually driven in joint space but the objects to be manipulated are expressed in the world coordinate system, the inverse kinematic solution is essential in controlling the position and orientation of the end-effector of the robot arm to reach its objects.

There are two classes of solution methods for the inverse kinematics problem: *closed form* and *numerical.* In robotics, a closed form solution is usually desired for the kinematic chain of a robot arm rather than a numerical solution. Numerical solutions are generally much slower than the corresponding closed form solution. Also, numerical solutions are not generally guaranteed to converge to the correct solution if they converge at all. It is therefore hard to predict the quality of the solution and the amount of time required to obtain the solution. For these reasons, researchers in robotics usually restrict their attention to closed form solutions.

The closed form solution of a kinematic chain can be obtained by one or both of the two solution methods: *algebraic* and *geometric.* Various algebraic methods include the inverse transform method[34], screw algebra[20], dual matrices[10], and the dual quaternion method [46]. Lee and Ziegler[26] have presented a geometric method to solve the inverse kinematics problem for the PUMA robot. A more thorough discussion of closed form solution methods can be found in [14].

2.2 Hand Model

Two components of hand model are presented in this thesis: an internal skeletal model is developed for the inverse kinematic calculations and a polygonal model for representing the outer skin of the hand. In much of the literatures involving animation of the hand [36, 44, 7], a simple three link pin joint model is often used internally to represent the fingers and for the computation of joint angles. It is, however, found to be insufficient in our application. The reasons will be made clear in Chapter 4. We have developed an internal skeletal model which generalizes the simple pin joint model by considering the effects of the finger joints. After the joint angles are calculated, the hand must be displayed on the screen. This is accomplished using a surface polygonal model of the hand. The following sections discuss each of the two components in detail.

2.2.1 Human Hand Anatomy

The hand is one of the most complex mechanisms in the human body as it has more than 25 degrees of freedoms. There have been numerous studies of the anatomical structure of the hand [2, 42, 43], anatomical representation of which is given in Figure 2.2.

The hand consists of five fingers and a palm. Each of the index finger, middle finger, ring finger and little finger has three joints. The joint closest to the palm is called the *metacarpophalangeal* joint, or the MCP joint for short. This joint has two degrees of freedom; an adduction-abduction range of approximately 30 degrees and a flexion and extension range of about 120 degrees. The remaining two joint are the *proximal interphalangeal* (PIP) joint and *distal interphalangeal* (DIP) joint respectively. They each have one degree of rotational freedom. The PIP joint has a range of 100 degrees while the DIP joint has a range of 60 degrees.

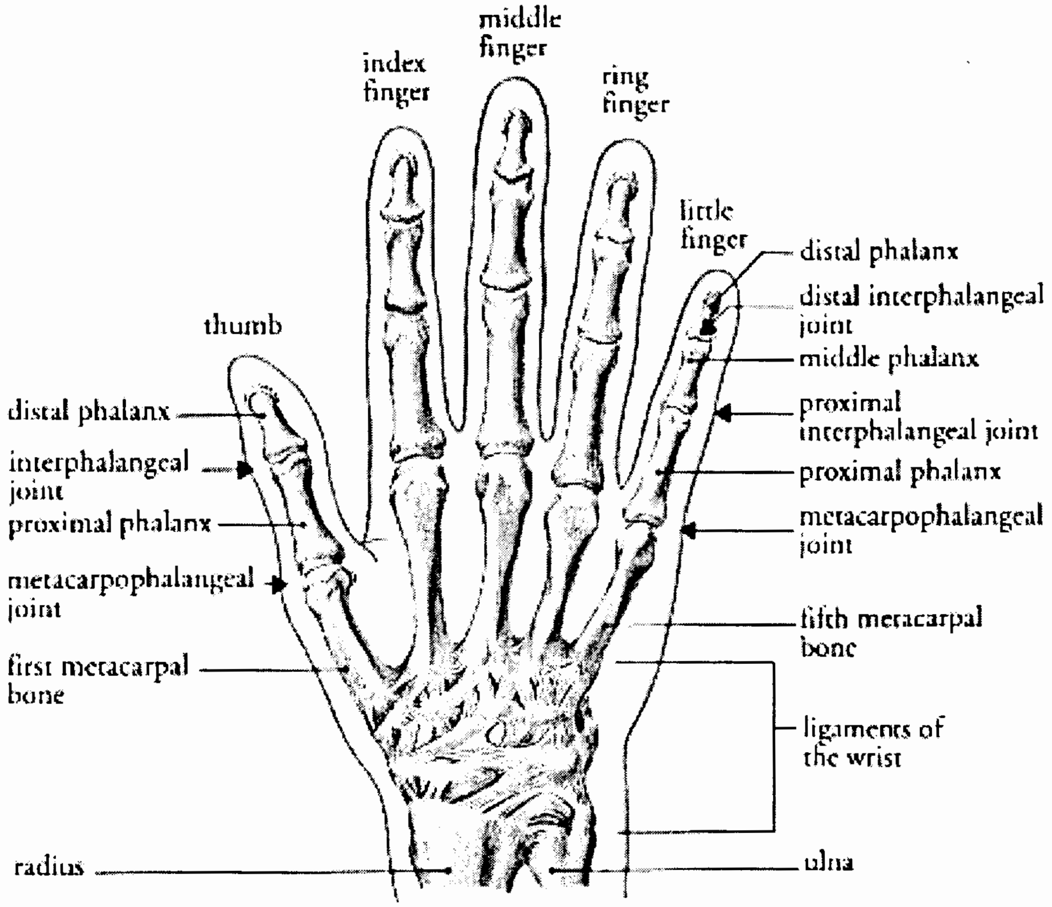

Figure 2.2: Anatomy of the human hand

The thumb is much more dexterous and therefore much more complex than the other four fingers. The thumb's proximal joint is known as the *carpometacarpal* (CMC) joint. It has two degrees of freedom; an adduction-abduction range of about 120 degrees and flexion and extension range of about 45 degrees. The next joint is the *metacarpophalangeal* (MCP) joint which also has two degrees of freedom; an adduction-abduction range of 30 degrees and a flexion-extension range of 50 degrees. The last joint is the *interphalangeal* (IP) joint which has only one degree of freedom and has a range of approximately 85 degrees.

Biomechanical Models of the Hand

From a biomechanical standpoint, the human hand can be considered as a linkage system of intercalated bony segments. The joints between each phalanx are spanned by ligaments, tendons and muscles. With the contraction of muscles, these joints can be moved in a characteristic manner constrained by the interposing soft tissues and the bony articulation [2]. In the hand, most of the tendons span the joint and continue their course over one or more joints, thus forming a bi-articular or poly-articular system.

The functional anatomy of the spatial relationships between these tendons and muscles and their associated joints have been extensively studied by Landsmeer [21, 23]. Landsmeer also proposed a series of models to represent the various manners in which tendons bridge the associated joints. However, these studies show either a lack of quantitative description or that the information is restricted in only two dimensions. An et al. [2] established a workable model in a three dimensional manner based on the direct and careful measurements of 10 normal specimens. Such a model can easily be utilized in the study of hand motion or force analysis of hand under various functional activities. Various other studies on the tendon excursion and moment arm of the finger can be found in [22, 24, 3]. Since information about forces acting on the finger, the tendon excursion, and the moment arm of the finger cannot easily be obtained using non-intrusive techniques, we have to construct a kinematic model of the hand using only the information obtainable by our equipments.

2.2.2 Internal Skeletal Model

In this section we develop an internal skeletal model of the hand based on anatomical information. The hand can be considered as an articulated structure composed of rigid segments connected by joints. Even though the palm is not completely planar, for the sake of simplicity, it can be approximated using a plane whose normal points out from the back of the hand. Since the joints of the fingers are clearly not simple pin joints, there must be a small but significant distance separating two consecutive links at each joint. We will call this distance the length of the joint. It is further hypothesized that the lengths of the joints change as the joints rotate. This hypothesis will be verified in Chapter 4. Therefore, we use functions $F_i^j(\boldsymbol{\theta}^j, r_i^j)$ to represent the length of the ith joint of the jth finger. The fingers are numbered from left to right on a right hand. The $\boldsymbol{\theta}^j$ is the joint angle vector of the jth finger and r_i^j is the joint radius of the ith joint on the jth finger. Clearly, if the functions F_i^j are uniformly 0, this model would reduce to the simple pin joint model.

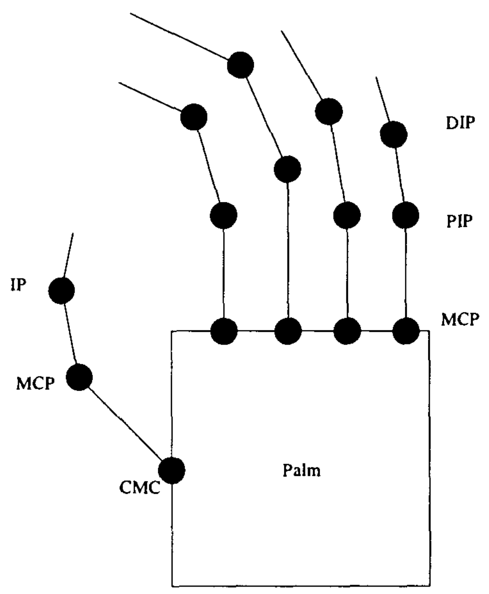

Figure 2.3: Internal Skeletal Model

Figure 2.3 shows a representation of a skeletal model of the hand. The black dots on the figure represent the joints. It is evident from the figure that each finger of the hand is represented a series of joint and rigid links. The functions F_i^j will be derived in Chapter 4.

2.2.3 Polygonal Hand Model

A polygonal model of the hand is used to represent the surface of the hand. This model consists of a list of vertices and a list of polygons, the latter forming the surface of the hand. Each polygon is defined by four vertices in the vertex list.

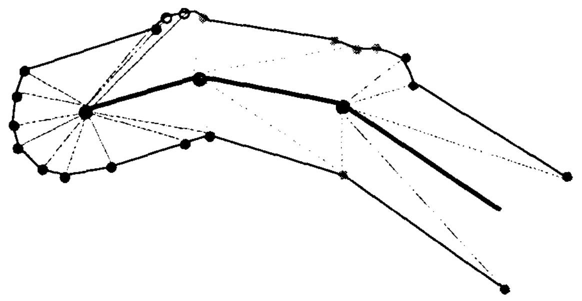

Figure 2.4: Attachment of vertices onto the bone segments

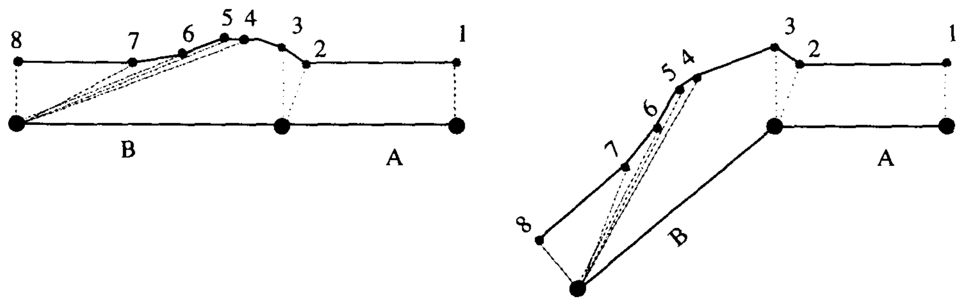

Figure 2.5: Finger bending

The polygonal data of the hand is obtained by digitizing an actual human hand. The surface polygons corresponding to each finger segment are initially unknown. This information is required in order to bend or rotate a specified finger. To obtain the needed information, a program was developed by Sidi Yu to allow the user to interactively select vertices and attach them to finger segments. All vertices of the polygonal hand model were manually attached to finger segments. Offsets from the distal end of each finger segment to the attached vertices are stored. Figure 2.4 shows the attachment of vertices onto the bones to create the polygonal hand model.

Finger bending is modeled by bending the finger segments and recalculating the positions of the vertices corresponding to each finger segment. The position of the distal end of the rotated finger segment is first calculated, then the stored vertex offsets were added back to the new distal position of the segment to obtain the new positions of the vertices. This process is shown in Figure 2.5. From the figure, we can see that the positions of the vertices four to eight have to be updated when segment B was rotates.

Chapter 3

Inverse Kinematics

Inverse Kinematics has been a practical problem in the field of robotics and computer graphics. In the past decades, researchers in both robotics and computer science have developed various algorithms to solve the inverse kinematics problem, but none seemed to work well in all situations. There are obvious tradeoffs between speed and accuracy among different classes of algorithms. In this chapter, the inverse kinematic problem is formally stated and various common approaches are presented along with their advantages and drawbacks.

3.1 Problem Definition

In Chapter 2, the basic notion of a kinematic chain of rigid segments has been introduced. For simplicity, we will refer to a kinematic chain as a *manipulator.* Let $\boldsymbol{x}$ be the position vector of the end-effector of the manipulator and $\boldsymbol{\theta}$ be the joint angle vector of the manipulator. Then the forward kinematic problem can be formulated as

$$\boldsymbol{x} = f(\boldsymbol{\theta}) \tag{3.1}$$

By inverting the function f, the inverse kinematic problem can be formulated as

$$\boldsymbol{\theta} = f^{-1}(\boldsymbol{x}) \tag{3.2}$$

The inverse kinematic problem is difficult because the function f is usually nonlinear. If the link parameters and the characteristics of the manipulator are well known in advance, we might be able to find a closed form solution. However, for arbitrary manipulators, we have to rely on numerical methods for solving systems of nonlinear equations.

3.1.1 Existence of Solution

The question of whether solutions exist or not is directly related to the manipulator's *workspace*. Intuitively, a workspace is a volume of space which the end-effector of the manipulator can reach. For at least one solution to exist, the specified goal point must lie within the workspace. There are two definitions of workspaces: *dextrous workspace* and *reachable workspace*. Dextrous workspace is the volume of space which the end-effector can reach with all orientations while the reachable workspace is the volume of space which the end-effector can reach in at least one orientation. The dextrous workspace is clearly a subset of the reachable workspace.

Consider the workspace of a two-link manipulator. If the length l_1 and l_2 of the two links are equal then the reachable workspace consists of a disc of radius $2l_1$. The dextrous workspace consists of only a single point, the origin. This example considers a workspace in which all joints of the two-link manipulator can rotate 360 degrees. This is rarely the case in actual configurations. When joint limits are a subset of the full 360 degrees, the workspace is obviously correspondingly reduced.

3.1.2 Redundancy

A manipulator is *kinematically redundant* if it possesses more degrees of freedom than are required to specify a goal for the end-effector. A planar arm with three revolute joints has a large reachable workspace and any position in the interior of its workspace can be reached with more than one orientation. Therefore, for any goal positions in the interior of the workspace, the kinematic equations will yield multiple solutions.

The fact that a manipulator has multiple solutions can cause problems because the system has to be able to choose one. The criteria with which to make a decision

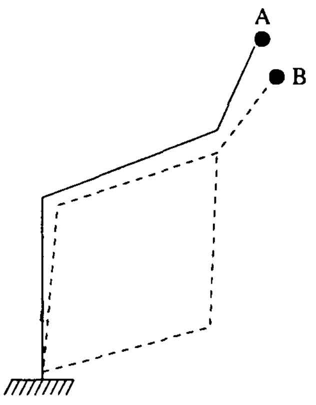

Figure 3.1: Possible solutions to reach point B

vary among different applications. As an example, if the end-effector is at point A as shown in Figure 3.1, and we wish to move to point B, a good choice for the solution would be one that minimizes the amount of change of joint angles. Therefore, if there is a choice and the previous position of the end-effector is known, we can choose the closest solution in joint space. To generalize this idea, we can assign weights to different links to generalize the notion of closeness.

3.2 Jacobian Based Methods

From (3.1), it is clear that in order to obtain f^{-1}, we need to solve a system of nonlinear equations. To avoid this time consuming process of solving a nonlinear system, we can reformulate the problem so that we have a linear relationship. Differentiating (3.1) with respect to time t, we have

$$\dot{\boldsymbol{x}} = J(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \tag{3.3}$$

where $J(\boldsymbol{\theta})$ is the Jacobian matrix. The Jacobian matrix and its inverse are discussed in Appendix B. The Jacobian is a multidimensional form of the derivative. At any particular instant, $\boldsymbol{\theta}$ has a certain value and $J(\boldsymbol{\theta})$ is clearly a linear transformation. At a new instant in time, $\boldsymbol{\theta}$ takes on a new value and so does the linear transformation $J(\boldsymbol{\theta})$. Therefore, the Jacobian is a time-varying linear transformation.

3.2.1 Inverse and Pseudoinverse of Jacobian

Intuitively, J maps the incremental changes in the joint variables to incremental changes in the end-effector position and orientation. If the matrix J is square and nonsingular, we can compute $\dot{\boldsymbol{\theta}}$ by

$$\dot{\boldsymbol{\theta}} = J^{-1}\dot{\boldsymbol{x}} \tag{3.4}$$

When this is not the case, J^{-1} does not exist and there is no solution for $\dot{\boldsymbol{\theta}}$. For an arbitrary manipulator, the Jacobian matrix J is not necessarily square and invertible. Therefore, a generalized form of the inverse J^{-1} is required that yields a "useful" answer for $\dot{\boldsymbol{\theta}}$ in the same form as (3.4):

$$\dot{\boldsymbol{\theta}} = J^{+}\dot{\boldsymbol{x}} \tag{3.5}$$

A commonly used generalized inverse J^{+} is the Moore-Penrose pseudoinverse[6]. It is shown in [5] and [6] that the Moore-Penrose pseudoinverse yields the minimum norm solution for the under determined case and the least squares solution for the over determined case.

3.2.2 Singularities

Most manipulators have values of $\boldsymbol{\theta}$ for which the Jacobian becomes singular. Such locations are called *singularities.* When a manipulator is in a singular configuration, it has lost one or more degrees of freedom. This means that there is some direction along which it is impossible to move the end-effector no matter what joint rate is selected. The two common classes of singularities are

- **Workspace boundary singularities** are those which occur when the manipulator is fully stretched out or folded back on itself such that the end-effector is near or at the boundary of the workspace.

- **Workspace interior singularities** are those which occur away from the workspace boundary and generally are caused by two or more joint axes lining up.

An in depth study of the classification of singularities can be found in [17]. When the manipulator is in a singular configuration, the inverse of the Jacobian matrix J is undefined. The pseudoinverse solution as defined in (3.5) yields unsatisfactory results since there is an undesirable discontinuity at the singularity. This often results in oscillations and unacceptably high joint velocities.

The Singular Value Decomposition

The Singular Value Decomposition (SVD) is a powerful method for the analysis of the kinematic properties of manipulators. The SVD theorem states that any matrix can be written as the product of three (non-unique) matrices. Decomposing J, we have

$$J = UDV^T \tag{3.6}$$

It is also common to write the SVD of the Jacobian matrix J as vector products. For an arbitrary Jacobian, it would be

$$J = \sum_{i=1}^{min(m,n)} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T \tag{3.7}$$

where m and n are number of rows and columns of J, and σ_i are diagonal elements of D, also known as *singular values.* The pseudoinverse solution is easily obtained from the singular value decomposition by taking the reciprocal of all nonzero singular values. In particular, the pseudoinverse of J is given by

$$J^+ = \sum_{i=1}^{r} \frac{1}{\sigma_i} \boldsymbol{v}_i \boldsymbol{u}_i^T \tag{3.8}$$

where r is the rank of J. The SVD has been used to detect and correct the numerical instabilities that arise when the manipulator is near its singularities[45, 30, 29, 13].

If one or more of the singular values σ_i are zero, then the original matrix is itself singular. The ratio of the largest singular value to the smallest one is called the *condition number* of the matrix. If the condition number is too large, then the matrix is ill-conditioned. This ill-conditioning is the cause of large joint velocities generated by the pseudoinverse near a singular configuration [29].

3.2.3 Homogeneous Solution

For a redundant manipulator, a goal can usually be satisfied in a number of ways. It is often desirable to select a solution that is optimal according to some additional criteria. Liegeois [27] has shown that the components of the gradient of such a criterion can be blended with the pseudoinverse solution.

$$\dot{\boldsymbol{\theta}} = J^{+}\dot{\boldsymbol{x}} + (I_n - J^{+}J)\nabla H(\boldsymbol{\theta}) \tag{3.9}$$

where $H(\theta)$ is a potential function to be minimized, I_n is the nth order identity matrix. The first term in (3.9) essentially selects the joint velocity vector which produces the desired change in the end-effector position, while the component $(I_n - J^{+}J)$ in the second term selects components of the gradient vector $\nabla H(\boldsymbol{\theta})$ that lie in the set of homogeneous solutions to equation (3.4). This has the effect of varying joint velocities in such a way that $H(\boldsymbol{\theta})$ is minimized without changing the end-effector position. Using this strategy, secondary goals can be created to avoid collisions with obstacles as well as maintain manipulator dexterity by avoiding kinematic singularities.

3.2.4 The Jacobian Transpose Method

Although Jacobian and pseudoinverse control methods offer generality and provide numerous ways to exploit redundancy, they suffer from several problems. First, these methods tend to be numerically unstable near singular configurations. Also, a matrix inversion is required at each iteration in order to obtain the joint velocity. This tends to slow down this class of algorithms and may cause problems if the Jacobian matrix is ill-conditioned. To address these problems, Welman [45] has proposed a Jacobian transpose method which is based on a simplified dynamic model.

This method applies the *principle of virtual work* [33] to obtain a linear relationship between the external force F applied to the tip of the manipulator and the *generalized* force τ.

$$\tau = J^T F \tag{3.10}$$

The difference between the desired position of the end-effector and the current position of the end-effector is then used as the force F pulling on the end-effector toward the desired trajectory. Since τ is equivalent to the vector of joint acceleration $\ddot{\boldsymbol{\theta}}$, a simplifying assumption is made to regard τ as the joint displacement $\dot{\boldsymbol{\theta}}$ instead of $\ddot{\boldsymbol{\theta}}$. Therefore, the simple relationship

$$\dot{\boldsymbol{\theta}} = J^T F \tag{3.11}$$

is obtained. Once $\dot{\boldsymbol{\theta}}$ is obtained, a single integration step yields a new vector $\boldsymbol{\theta}$ which moves the end-effector towards the desired trajectory. The procedure repeats until the end-effector reaches the desired position, or some other stopping criterion is met.

The Jacobian transpose method has many very attractive properties. First, there is no matrix inversion required. This not only speeds up the computation, it also avoids dealing with singular matrices. Also, since the computation is very simple at each step, the algorithm is able to provide real-time feedback to the user. Since the method is based on a simplified dynamic model, the solutions obtained in successive iterations are predictable. The end-effector of the manipulator is continuously pulled toward the desired trajectory by a force generated with the end-effector position error. Therefore, successive iterations would yield configurations that are close to each other and at the same time toward the goal. This characteristic makes the algorithm especially suitable for interactive applications in which intermediate solutions are often needed to refresh the screen.

A major drawback of this algorithm is that it does not address joint limits directly. Instead, joint limits are enforced by clamping to the upper and lower bounds. This might cause the algorithm to be trapped at an intermediate configuration and stopped only when the maximum number of iterations is reached. In addition, the algorithm is not completely immune to singular configurations since the Jacobian matrix itself is inherently ill-conditioned. Oscillations and high joint velocities can result near a singular configuration.

3.3 Optimization Based Methods

A different approach to solving the inverse kinematic problem is to reformulate it into an optimization problem and then apply the standard algorithms of nonlinear optimization to solve it. Since a positional goal of the end-effector can be viewed as a simple positional constraint, we can set up an objective function and minimize it subject to a set of constraints. This approach allows us to find a solution that satisfies multiple simultaneous constraints [35, 47]. The problem is generally formulated as

$$\begin{aligned} &\textit{minimize} \;\; P(\boldsymbol{\theta}) \\ &\textit{subject to} \;\; l_i \le \boldsymbol{\theta}_i \le u_i \qquad i = 1 \ldots n \end{aligned}$$

where $P(\boldsymbol{\theta})$ is the objective function to minimize, l_i and u_i are the lower and upper joint limits respectively.

The flexibility offered by the optimization formulation of the inverse kinematic problem enables us to specify various kinds of goals. Objective functions can be set up for each of them. A detailed discussion of different categories of goals and their corresponding objective functions can be found in [47].

Solving the constraint optimization problem is not a trivial task. It is still a relatively new research area that produces collections of numerical methods for each class of problem. Rosen's projection method is very effective in treating linear constraints[37]. Goldfarb combined the DFP method (a variable metric method) [12] with Rosen's projection method [16]. After that, the variable metric method was significantly improved by the BFGS formula [38].

Zhao and Badler have applied nonlinear optimization techniques to their animation system *Jack*. Their animation system specifies a configuration of an articulated figure as spatial constraints. The constrained parts of the articulated figure are the end-effectors and their counterpart in space are called goals. A goal can be as simple as a position, an orientation, a weighted combination of position and orientation, a line, a plane, a direction, and so on, or it could be as complicated as a region in the space. The optimization algorithm used in *Jack* is able to adjust the joint angles subject to joint limits so that the set of end-effectors concurrently attempts to achieve their respective goals. After the end-effectors and the goals are specified by the users, the system computes a final configuration. Since it is often impossible to satisfy all the goals owing to the actual constraints, the system outputs the best possible solution according to the users' assignment of importance to each goal.

Bawa [4] has proposed another iterative nonlinear constrained optimization algorithm to solve the inverse kinematic problem. This algorithm combines the augmented Lagrangian method [18] and the projected Lagrangian method [15] to achieve super-linear convergence. An animation system was built by Bawa that allows real-time manipulation of the end-effectors to demonstrate this technique. Similar to the *Jack* animation system, Bawa's system is also able to handle multiple simultaneous constraints. In addition, a trajectory can be associated with each end-effector over a period of time to create a complete animation sequence. Furthermore, real-time performance of this animation system is ensured by the use of efficient matrix inversion techniques.

Although the optimization formulation of the inverse kinematic problem has many advantages, it has some drawbacks as well. Because of the complexity of the problem and the scale of the search space, there is no guarantee that the optimizer will find a global minimum (or maximum). As a result, it may not return the best solution to the problem. Also, because of the iterative nature of the solution methods, it is hard to estimate how much time the optimizer will spend on a particular problem. Furthermore, since the optimizer is not constrained to search in any fixed direction, intermediate solutions may not be suitable for refreshing the screen even if the screen needs to be refreshed because the optimizer spends a long time to obtain a particular solution.

3.4 Table Based Method

In the efforts toward representing hand postures using a limited number of sensors, Amaya et al. [1] have proposed an algorithm that utilizes a lookup table to solve the inverse kinematic problem. Since this method is the predecessor of the algorithm proposed in this thesis, it will be discussed in detail. The notation used in Amaya's paper is discussed in Appendix A.

There are several motivations for the development of Amaya's algorithm. First, it is well known that optical motion tracking systems cannot effectively deal with the possibility of the occlusion of markers [32]. If we were to use the direct approach described in Section 1.2 which places an IRED on each joint and tip of the fingers, then the space in which the hand can freely move around without having any IREDs going out of the camera's view would be very small. As a result, only limited movements of the hand can be measured. This motivates the use of inverse kinematics to estimate the hand posture, since only one IRED is need at the tip of each finger to obtain the joint angles of that finger. This effectively enlarges the space in which the hand can freely move around without causing occlusion of IREDs.

A second motivation for the development of Amaya's algorithm was the need for speed in real-time computation. The Optotrak system continuously sends positional data of IREDs to the data collection system. If a significant amount of time is required for the computation of finger joint angles, then the data collection system would not be able to keep up with Optotrak system. This means that the data collection system would have to discard some incoming data and try to keep up with the Optotrak system by computing and displaying the most recent hand posture. Because the inverse kinematic equations are very complex and difficult to solve given a small time frame, Amaya et al. have suggested using a lookup table to speed it up.

Given particular constraints of finger motion, there are few sets of joint angles that correspond to fingertip positions with respect to the palm. Therefore, if we have a table of fingertip position versus orientation, we can lookup the orientation of the fingertip given its position. Using this information from the table, only simple calculations are required to obtain the joint angles. Thus, the basic approach behind Amaya's algorithm is to build a lookup table of finger tip orientations using additional IRED information before the start of the actual data collection. Then, this lookup table is searched during data collection to obtain the current finger tip orientation which we can use to compute the joint angles with minimum effort. This approach suggests that the algorithm should proceed in two stages: a preprocessing stage before data collection to build the lookup table and a run-time stage which uses the lookup table to compute the joint angles.

The lookup table built by the preprocessing stage uses the location of the finger tip as an index for the corresponding orientation. Amaya et al. made the assumption that the position of the finger tip with respect to the root of the finger uniquely defines the orientation of the fingertip. The validity of this assumption is discussed in Section 3.4.3. The orientation is then used to determine the joint angles of the finger. The orientation of the finger tip is obtained in the preprocessing stage by placing additional IREDs on the finger tip as shown in Figure 3.2. However, during run-time, only one IRED is needed at the tip to obtain its coordinates with respect to the root of the finger. The orientation of the finger tip at run time is obtained by using this finger tip coordinate to index into the lookup table.

3.4.1 Preprocessing Stage

The preprocessing stage consists of several steps:

1. **Registration:** The positional measurement of the IREDs mounted on the hand with respect to the Optotrak workspace.

2. **Table Building:** The recording of position and orientation of the fingertip with respect to the hand as the finger joints are rotated.

3. **Data Filtering:** The filtering and regularization of the data recorded in the Table Building step.

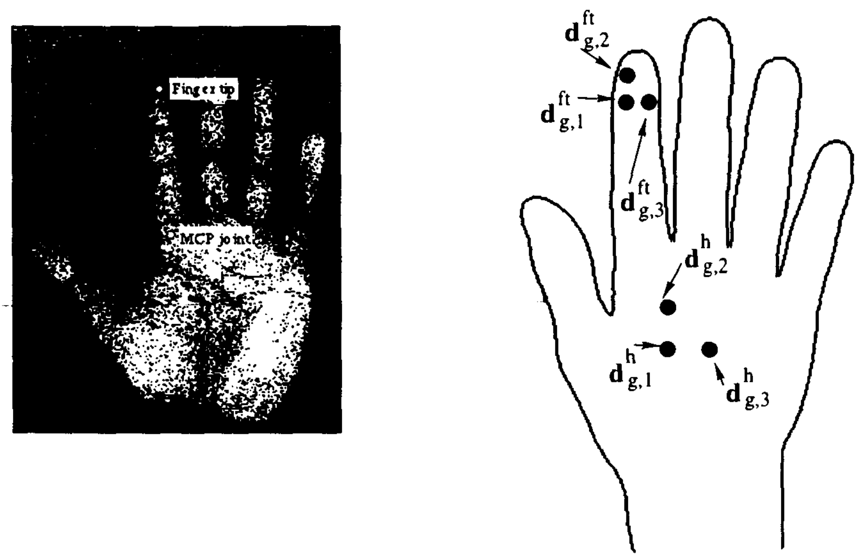

Figure 3.2: IRED placement in the preprocessing stage of Amaya's algorithm.

Registration

In the registration step, the lengths of the segments of the index finger and the thumb are measured using a caliper. The hand is then placed palm down on top of its scanned image on a fixed, rigid and flat surface as shown in Figure 3.2. Mounted on the back of the hand are three IREDs and on the tip of the finger are mounted another three IREDs. The positions of these IREDs in the Optotrak space (or global space) are $\boldsymbol{d}^{h}_{g,i}$ and $\boldsymbol{d}^{ft}_{g,i}$. Recall that $\boldsymbol{d}^{h}_{g,i}$ ($\boldsymbol{d}^{ft}_{g,i}$) represents the position of the ith IRED on the hand (fingertip) in the global coordinate system. The following measurements are taken in the global coordinate system:

- The position of the MCP joint (1 X-Y-Z point). It is done by scanning the subject's hand and placing an IRED on the MCP joint of the scanned image.
- The position of the fingertip (1 X-Y-Z point). It is also done by placing an IRED on the finger tip of the scanned hand.
- The position of the IREDs mounted on the back of the subject's hand (3 X-Y-Z points). The subject's hand is placed on top of the scanned hand.

- The position of the IREDs mounted on the tip of the subject's finger (3 X-Y-Z points). The subject's hand is placed on top of the scanned hand.

Using these initial measurements on the subject's right hand, the following transformations and vectors are defined:

- $\boldsymbol{O}_g^h$ - The position of the MCP joint in global space.
- $[B_g^h]$ - The orthonormal orientation of the MCP joint in global space.
- $\boldsymbol{O}_g^{ft}$ - The position of the fingertip in global space.
- $[B_g^{ft}]$ - The orthonormal orientation of the fingertip in global space.
- $\boldsymbol{x}_h$ - The x axis of the hand coordinate system lying along the vector from the MCP joint to the fingertip in the plane $z = 0$.
- $\boldsymbol{y}_h$ - The y axis of the hand coordinate system pointing to the right.
- $\boldsymbol{z}_h$ - The z axis of the hand coordinate system which is pointing out of the back of the hand.
- $\boldsymbol{x}_{ft}$ - The x axis of the fingertip coordinate system lying along the vector from the MCP joint to the fingertip in the plane $z = 0$.
- $\boldsymbol{y}_{ft}$ - The y axis of the fingertip coordinate system pointing to the right.
- $\boldsymbol{z}_{ft}$ - The z axis of the fingertip coordinate system which is pointing out of the back of the hand.

The hand and fingertip coordinate systems are shown in Figure 3.3. Since $\boldsymbol{x}_h$, $\boldsymbol{y}_h$ and $\boldsymbol{z}_h$, are unit vectors and they represent a right handed system, the transformation matrix $[B_g^h]$ is orthonormal. In particular, $[B_g^h]$ satisfies

$$[B_g^h]^T[B_g^h] = I \tag{3.12}$$

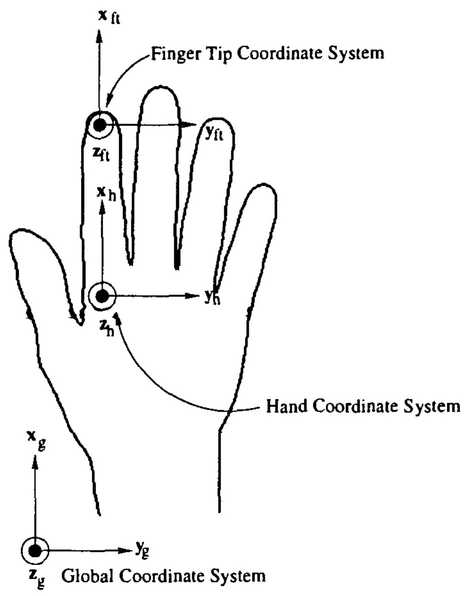

Figure 3.3: The hand and fingertip coordinate systems

where I is the identity matrix.

The positions of the IREDs, $\boldsymbol{d}^h_{h,i}$ and $\boldsymbol{d}^{ft}_{ft,i}$, with respect to the hand and fingertip coordinate frame are computed as

$$\boldsymbol{d}^h_{h,i} = [B^g_h](\boldsymbol{d}^h_{g,i} - \boldsymbol{O}^h_g), \tag{3.13}$$

$$\boldsymbol{d}^{ft}_{ft,i} = [B^g_{ft}](\boldsymbol{d}^{ft}_{g,i} - \boldsymbol{O}^{ft}_g) \tag{3.14}$$

The vectors $\boldsymbol{d}^h_{h,i}$ and $\boldsymbol{d}^{ft}_{ft,i}$ will be used to compute the rotation and translation of the hand and fingertip coordinate system within the global system. Vectors with identical superscript and subscript are assumed to be constant through all phases of the process following their measurement.

Table Building

It is assumed by Amaya et al. that the position of the fingertip in relation to the MCP joint in the hand coordinate system uniquely defines the orientation of the fingertip. Thus, a subject is asked to move his finger as extensively as possible to try to fill out the complete workspace of the finger. In the table building process, the hand and finger tip coordinate systems are constructed using the three IREDs on the fingertip and the three IREDs on the palm. This is done by writing the unit vectors pointing along each of the three axes into the columns of the transformation matrix. The origin of these fingertip and hand coordinate systems are translated to the finger tip position and the MCP joint respectively. Mathematically, this can be expressed as:

$$\begin{aligned}
\left[B_g^{ft}\right] &= \left[\frac{\boldsymbol{d}_{g,2}^{ft} - \boldsymbol{d}_{g,1}^{ft}}{\|\boldsymbol{d}_{g,2}^{ft} - \boldsymbol{d}_{g,1}^{ft}\|}, \frac{\boldsymbol{d}_{g,3}^{ft} - \boldsymbol{d}_{g,1}^{ft}}{\|\boldsymbol{d}_{g,3}^{ft} - \boldsymbol{d}_{g,1}^{ft}\|}, \frac{(\boldsymbol{d}_{g,2}^{ft} - \boldsymbol{d}_{g,1}^{ft}) \times (\boldsymbol{d}_{g,3}^{ft} - \boldsymbol{d}_{g,1}^{ft})}{\|(\boldsymbol{d}_{g,2}^{ft} - \boldsymbol{d}_{g,1}^{ft}) \times (\boldsymbol{d}_{g,3}^{ft} - \boldsymbol{d}_{g,1}^{ft})\|}\right] && (3.15)\\
\boldsymbol{O}_g^{ft} &= \boldsymbol{d}_{g,1}^{ft} - [B_g^{ft}]\boldsymbol{d}_{ft,1}^{ft} && (3.16)\\
\left[B_g^{h}\right] &= \left[\frac{\boldsymbol{d}_{g,2}^{h} - \boldsymbol{d}_{g,1}^{h}}{\|\boldsymbol{d}_{g,2}^{h} - \boldsymbol{d}_{g,1}^{h}\|}, \frac{\boldsymbol{d}_{g,3}^{h} - \boldsymbol{d}_{g,1}^{h}}{\|\boldsymbol{d}_{g,3}^{h} - \boldsymbol{d}_{g,1}^{h}\|}, \frac{(\boldsymbol{d}_{g,2}^{h} - \boldsymbol{d}_{g,1}^{h}) \times (\boldsymbol{d}_{g,3}^{h} - \boldsymbol{d}_{g,1}^{h})}{\|(\boldsymbol{d}_{g,2}^{h} - \boldsymbol{d}_{g,1}^{h}) \times (\boldsymbol{d}_{g,3}^{h} - \boldsymbol{d}_{g,1}^{h})\|}\right] && (3.17)\\
\boldsymbol{O}_g^{h} &= \boldsymbol{d}_{g,1}^{h} - [B_g^{h}]\boldsymbol{d}_{h,1}^{h} && (3.18)
\end{aligned}$$

The goal is to create a mapping from fingertip position to fingertip orientation in hand space. Thus, we measure the position and orientation of the fingertip with respect to the hand.

$$\left[B_h^{ft}\right] = [B_h^g]\left[B_g^{ft}\right] \tag{3.19}$$

$$\boldsymbol{O}_h^{ft} = [B_h^g]\,(\boldsymbol{O}_g^{ft} - \boldsymbol{O}_g^{h}) \tag{3.20}$$

Recording sufficiently many of these pairs, $< \boldsymbol{O}_h^{ft},[B_h^{ft}] >$, while the finger joints rotate creates the table which defines the mapping we need (i.e. given a position $\boldsymbol{O}_h^{ft}$, we can look up a transformation to orient the fingertip in the hand coordinate system):

$$Map: \quad \boldsymbol{O}_h^{ft} \rightarrow [B_h^{ft}] \tag{3.21}$$

However, the recorded pairs are scattered and usually sufficiently sparse that for any particular value of $\boldsymbol{O}_h^{ft}$ there is unlikely to be a particular tuple $< \boldsymbol{O}_h^{ft}, [B_h^{ft}] >$ in *Map*. Since we are interested in the specific $[B_h^{ft}]$ for a given $\boldsymbol{O}_h^{ft}$, we will use the notation $\left[B_h^{ft}(\boldsymbol{p})\right]$ to represent an idealized *Map* where $\boldsymbol{p}$ is a particular value of $\boldsymbol{O}_h^{ft}$ and $\left[B_h^{ft}(\boldsymbol{p})\right]$ is the corresponding value of $[B_h^{ft}]$ in the map.

Data Filtering

For values of $\boldsymbol{O}_h^{ft}$ where a corresponding value of $[B_h^{ft}]$ is not available from *Map*, Amaya et al. suggest using a 3D Gaussian filter over all of the entries in *Map*.

$$\left[\bar{B}_h^{ft}\left(\boldsymbol{d}_{h,1}^{ft}\right)\right] = \left(\sum_{y=\dot{\boldsymbol{d}}_h^{ft}} w\left(\boldsymbol{d}_{h,1}^{ft} - y\right)\left[\dot{B}_h^{ft}(y)\right]\right) / \left(\sum_{y=\dot{\boldsymbol{d}}_h^{ft}} w\left(\boldsymbol{d}_{h,1}^{ft} - y\right)\right) \quad (3.22)$$

$$\bar{\boldsymbol{O}}_h^{ft}\left(\boldsymbol{d}_{h,1}^{ft}\right) = \left(\sum_{y=\dot{\boldsymbol{d}}_h^{ft}} w\left(\boldsymbol{d}_{ft,i}^{h} - y\right)\dot{\boldsymbol{O}}_h^{ft}(y)\right) / \left(\sum_{y=\dot{\boldsymbol{d}}_h^{ft}} w\left(\boldsymbol{d}_{ft,i}^{h} - y\right)\right) \quad (3.23)$$

$$w(y) = e^{-\frac{\|y\|^2}{\sigma^2}} \quad (3.24)$$

where $[\bar{B}_h^{ft}()]$ is the fingertip orientation lookup table, $\boldsymbol{d}_{h,1}^{ft}$ is the position of the fingertip, $\dot{\boldsymbol{d}}_h^{ft}$ represent the neighbors of $\boldsymbol{d}_{h,1}^{ft}$ in $[\bar{B}_h^{ft}()]$, and $w(a)$ is a Gaussian with standard deviation σ (in millimeters). It is suggested by Amaya et al. that a value of $8mm$ is sufficient for σ. The calculation required to compute the 3D Gaussian convolution is very expensive under the real time experimental system. Instead, this computation is performed at regularly spaced grid points as a batch process. The grid used has 28 $\times$ 24 $\times$ 20 points with a point separation of 5mm. This grid of values, indexed by $\boldsymbol{O}_h^{ft}$, contains filtered values of $[B_h^{ft}]$ and is called *Map′*. At run time, simple linear interpolation is used to obtain the intermediate values. The lookup table is shown in Figure 3.4. The black dots on the figure represent the table entries.

3.4.2 Run Time Stage

After *Map′* is created, for any value of $\boldsymbol{O}_h^{ft}$ we can quickly perform a simple linear interpolation of the nearest eight values in *Map′* for the corresponding value of $[B_h^{ft}]$.

During run time, there are still three IREDs placed on the back of the hand in order to determine the orientation of the hand but only one IRED is placed on the tip

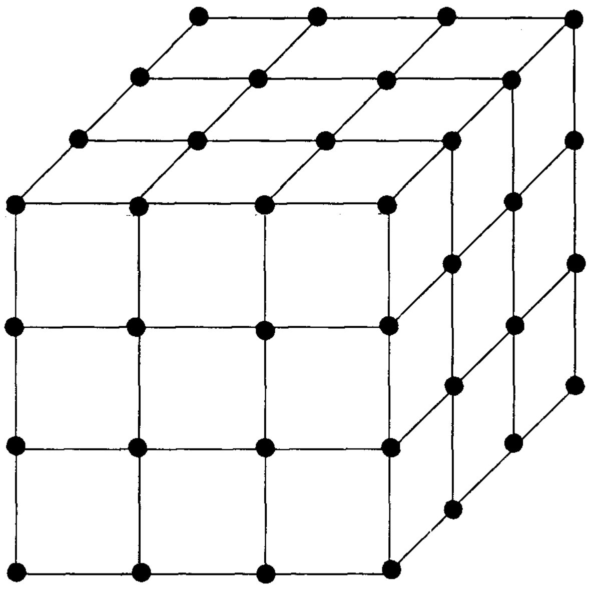

Figure 3.4: The lookup table in Amaya's Algorithm

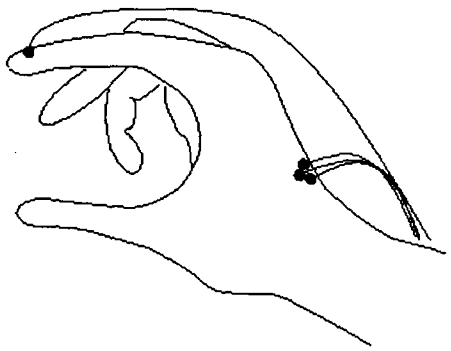

Figure 3.5: IRED placement at run time

of each finger as shown in Figure 3.5. These IREDs are not required to be placed in exactly the same position as in the preprocessing stage. The positions of the IREDs on the back of the hand with respect to the hand and fingertip coordinate system are determined using equations (3.13) and (3.14). Since there is only one IRED at the fingertip, it is necessary to transform the indices of the fingertip orientation map from the position of the fingertip to the position of the IRED. This is accomplished by

$$\boldsymbol{d}_{h,1}^{ft} = \left[B_h^{ft}\left(\boldsymbol{O}_h^{ft}\right)\right] \boldsymbol{d}_{ft,1}^{ft} + \boldsymbol{O}_h^{ft} \tag{3.25}$$

where $\boldsymbol{d}_{ft,1}^{ft}$ is the vector from the fingertip position $\boldsymbol{O}_h^{ft}$ to the IRED mounted on the finger in the fingertip space. This is illustrated in Figure 3.6. At experiment time, we know:

- $\boldsymbol{d}_{h,1}^{ft}$ - The measured position of the finger IRED in the hand coordinate system. This position changes as the finger joints are rotated.
- $\boldsymbol{d}_{ft,1}^{ft}$ - The measured position of the fingertip IRED in the fingertip coordinate system. This position is constant and is established at the registration step.
- $\left[B_h^{ft}(\boldsymbol{O}_h^{ft})\right]$ - The orientation of the fingertip in the hand coordinate system given the position of the fingertip. This orientation is constant and is established in the calibration and filtering step.

Thus the unknown is $\boldsymbol{O}_h^{ft}$. In order to use Map', we need to build a second index into it so that we can use the measured value of $\boldsymbol{d}_{h,1}^{ft}$. We can build this index using Equation (3.25). For each entry in Map', given the value of $\boldsymbol{d}_{ft,1}^{ft}$, we can compute the value of $\boldsymbol{d}_{h,1}^{ft}$ that will be measured at experiment time corresponding to the appropriate tuple $< \boldsymbol{O}_h^{ft}, [B_h^{ft}] >$. Linear interpolation is used to compute the value of $\boldsymbol{O}_h^{ft}$ and $[B_h^{ft}]$ for specific values of $\boldsymbol{d}_{h,1}^{ft}$.

After the orientation of the fingertip is obtained, we can use it to find the position of the DIP joint. This is done by subtracting the length of the segment from the DIP joint to the fingertip from the current position of the fingertip in the opposite direction of the current orientation of the fingertip. Using the location of the DIP joint, we

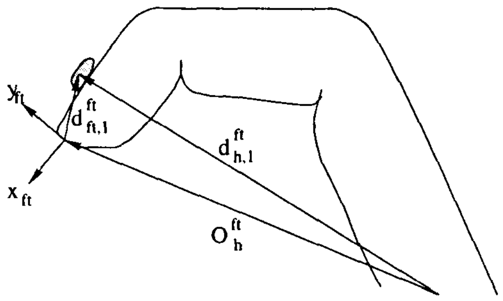

Figure 3.6: The known and unknown vectors when estimating the orientation and position of the fingertip

can solve for the position of the PIP joint by considering the two proximal segments of the fingers as a two link manipulator with the end-effector fixed at the DIP joint. Craig [9] discusses various analytical methods for solving two link manipulators.

3.4.3 Discussion

This table lookup based inverse kinematic method has several merits. Since the fingertip position to orientation map is built prior to the start of the real time experiment, the computation required to obtain the joint angles is kept to a minimum. Its superior computational speed makes it quite suitable for real time applications. Also, calibration is done for each individual subject and the fingers' workspace is also traced out by the subject. Doing so will not only avoid interior and boundary singularities, but also ensure that the computed hand posture closely resembles that of the actual hand posture of the subject. Furthermore, all the constraints and joint limits are directly built into the lookup table since it is constructed using only hand postures attainable by the subject. Therefore, no constraint violation needs to be checked before displaying the computed hand posture.

All methods have drawbacks in one way or another and the table based method is no exception. This particular construction of the lookup table relies heavily on the assumption that the position of the fingertip in relation to the MCP joint in the hand coordinate system uniquely defines the orientation of the fingertip. The result produced by this algorithm may not be valid if this assumption fails. In addition, the algorithm first calibrates for the length of each finger segment and makes the assumption that the finger segments are rigid so their lengths will be constant throughout.

Although the uniqueness assumption is valid if one assumes the finger joints are simple pin joints, the measured length of a finger segment may be perturbed by skin movement during an actual experiment. Further, since the joints of the finger are clearly not simple pin joints, the previous assumption is invalid even without the effect of skin movements. The result is that inconsistent measurement of finger segment lengths will occur during an experiment. The accuracy and even existence of the computed joint angles rely heavily on the accuracy of the finger segment length measurements. This is because the internal skeletal model is assumed to be a simple pin joint model. If the measured segment lengths are longer than the actual lengths, the set of computed joint angles is surely going to be inaccurate. Even worse, if the measured segment lengths are shorter than the actual lengths, the goal is simply beyond the reach of the finger and no solution can be computed.

This drawback of Amaya's algorithm often causes the data collection system to discard data sent by the Optotrak system simply because the finger segment length measurements do not match the initial calibration measurements. Therefore, we need to develop an algorithm which has the desirable properties of Amaya's algorithm and at the same time addresses its drawback. The algorithm we developed to do just this is described in Chapter 4.

3.5 Summary

In this chapter, we have reviewed some of the most commonly used techniques for solving the inverse kinematic problem. Each method consists of distinct advantages and disadvantages. Problems inherent in one method may not exist in another. Tradeoffs between generality and speed are apparent. For instance, optimization based methods tend to be more general in the sense that they can handle problems with an arbitrary number of links having a set of possible constraints for each link. However, algorithms for nonlinear optimization tend to be slow due to its complexity. By comparison, the Jacobian based methods are faster, but they are not very good at handling constraints.

The fastest method presented in this chapter is the table based method. Because of the number of assumption it made about the configuration of the manipulator and its workspace, this method can only be applied to a small well known set of simple manipulators. Since the human finger can be considered to be a simple three link manipulator with a small number of constraints, the table based method seems to be appropriate.

Chapter 4

A Table Based Real-time Algorithm

The table based method presented in Chapter 3 provides a quick way of obtaining inverse kinematic solutions for simple manipulators. Because of the demand for speed in real time applications and because the human finger can often be considered as a simple three link manipulator, the table based method is ideal for our application.

In this chapter, a table based method which addresses the problems discussed in Chapter 3 and which is specifically designed to model the human finger is presented. This method is relatively simple to implement and provides good performance for the intended application.

4.1 Motivation

The method presented by Amaya et al. [1] as discussed in Chapter 3 makes the assumption that the human finger is a simple three link manipulator with one degree of rotational freedom at each joint. Because the joint of the finger is much more complex than a simple pin joint and because the IREDs cannot be placed at the center of each joint, this assumption becomes invalid.

For any simple three link pin joint manipulator, it is necessary that the distance from the end-effector to the base is a maximum when the manipulator is fully stretched, or $\boldsymbol{\theta} = \mathbf{0}$. We can express the distance between the finger tip and the root of the finger as:

$$d_{mcp}^{tip} = \|\boldsymbol{p}_{tip} - \boldsymbol{p}_{mcp}\| \tag{4.1}$$

where $\boldsymbol{p}_{tip}$ is the location of the fingertip, $\boldsymbol{p}_{mcp}$ is the location of the MCP joint, and d_{mcp}^{tip} is the distance between them.

An experiment was conducted to verify whether the maximum of equation (4.1) occurs when a real finger is fully stretched, i.e. when $\boldsymbol{\theta} = \mathbf{0}$. The length of each segment of the index finger is measured: the length of the segment from the MCP joint to the PIP joint was 45.8mm, the length of the segment from the PIP joint to the DIP joint was 20.1mm, and the length of the segment from the DIP joint to the tip was 20.0mm. These measurements were first taken using the Optotrak system and then confirmed manually using a ruler. An IRED was placed on both the fingertip and the MCP joint of the index finger. The distance between the fingertip and the MCP joint was measured as the finger rotates. The result is shown in Figure 4.1. The dotted line represents the sum of three segment lengths measured when the index finger is fully extended. The solid line is the distance between the fingertip and the MCP joint measured by the Optotrak as the index finger rotates. By examining the recorded joint angles of the index finger, it is clear that the maximum of equation (4.1) did not occur when the index finger was fully extended. Furthermore, a significant number of measured distances between the fingertip and the MCP joint are greater than the sum of the segment lengths. However, the difference between the measured maximum distance and the length of the finger is surprisingly large (about 10mm). Because of this unexpected result, the experiment was repeated five times. The results from these experiments were similar. This shows that the finger joints are clearly not pin joints. Thus, there must be a small distance separating any two consecutive segments. Recall that we have called this separating distance the length of the corresponding joint in Chapter 2. Moreover, we also know that the length of the joints vary as a function of the corresponding joint angles, for otherwise the maximum of equation (4.1) would have occurred at $\boldsymbol{\theta} = \mathbf{0}$. Therefore, our hypothesis in Chapter 2 has been verified.

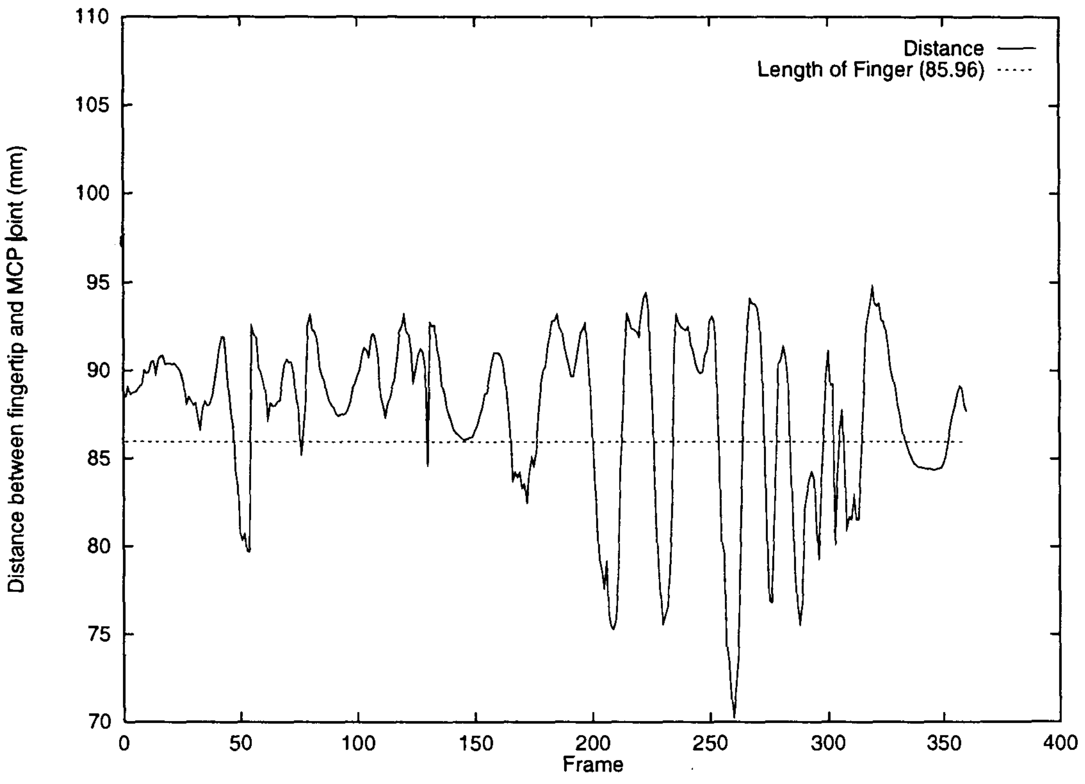

Figure 4.1: Distance between the finger tip and the MCP joint.

The fact that the maximum of equation (4.1) does not occur at $\boldsymbol{\theta} = \mathbf{0}$ has a very serious effect on Amaya's algorithm. Because Amaya's algorithm assumes that the joints are simple pin joints, it is able to subtract the length of the last segment of the length in the opposite orientation of the fingertip to obtain the position of the DIP joint. The last step in Amaya's algorithm is to regard the position of the calculated DIP joint as the end-effector position of the two link simple pin joint manipulator formed by the first two segments of the finger and solve for their joint angles. Since there is distance separating any two consecutive segments of the finger, the calculated position of the DIP joint is clearly outside the workspace of this two link manipulator. Therefore, the algorithm will not be able to find any solution. It is necessary to eliminate this undesirable property in Amaya's algorithm for it to function effectively in a real time setting.

4.2 IRED Placements

Before the hand posture can be estimated, IREDs will have to be strategically placed on the hand to allow the Optotrak system to track their positions. This section discusses the IRED placements required for calibration as well as for real-time experiments. For the remainder of this chapter, the ith IRED position in the Optotrak coordinate system is denoted as $\boldsymbol{d}_i$.

4.2.1 IRED positions

In order to track the position and orientation of the hand, IREDs have to be placed on the dorsal surface of the hand. There are two different IRED placements: one for calibration and the other for run time. The calibration step is necessary in order to obtain the auxiliary parameters of the finger; these parameters are the length of each segment of the finger, the radius r_i of the ith joint, and an additional parameter ρ which will be discussed in the next section. After these parameters are collected, unneeded IREDs are removed so that only a minimal number of IREDs are left.

This minimal configuration of remaining IREDs ensures that the occlusion problem of the Optotrak system is minimized and the movement of the fingers and the palm is not severely restricted by IRED visibility. Since the hand is not flat, any movement of the palm or bending of the fingers may block one or more IREDs from the view of the camera. Figure 4.2 shows some common situations in which one IRED can go out of the view of the Optotrak camera.

Calibration

During calibration, the thickness of the finger at each joint is measured. the joint radius r_i of the ith joint is simply one half of the thickness measured at the ith joint. The thickness is easily measured with a caliper.

The length of each of the finger segments is measured using the IRED placements shown in Figure 4.3. The hand should be placed palm down on a flat table and the fingers be fully extended.

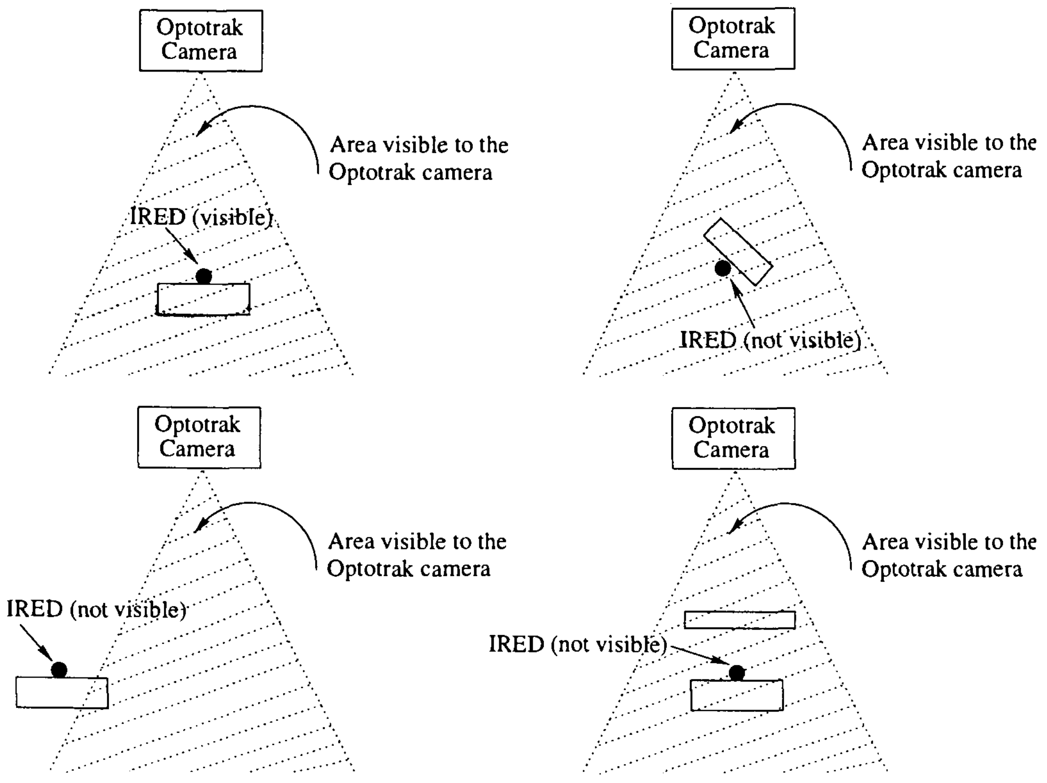
Optotrak Camera
Area visible to the Optotrak camera
IRED (visible)
Optotrak Camera
Area visible to the Optotrak camera
IRED (not visible)
Optotrak Camera
Area visible to the Optotrak camera
IRED (not visible)
Optotrak Camera
Area visible to the Optotrak camera
IRED (not visible)

Figure 4.2: Various cases that an IRED could become invisible to the Optotrak camera

The length of each individual segment can be calculated by computing the distance between IREDs placed at the ends of each segment. IREDs 3, 4, and 5 define the plane of the palm. The final calibration step measures the ratio between the angles of the DIP and the PIP joints of the index finger. We will call this parameter ρ. Rijpkema and Girard [36] suggested that there is a linear relationship between θ_{pip} and θ_{dip}:

$$\theta_{pip} = \rho\theta_{dip} \tag{4.2}$$

We can use this relationship to simplify the table building process as discussed in Section 4.3.2. The parameter ρ is computed as the slope of the least squared line that passes through the data points $(\theta_{pip_i}, \theta_{dip_i})$ in the $\theta_{pip}\theta_{dip}$ plane.

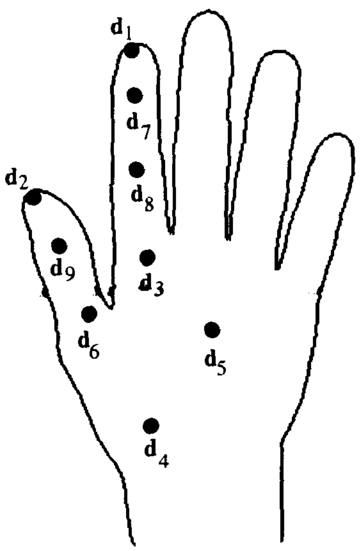

Figure 4.3: IRED placement for measuring segment lengths.

$$\rho = \frac{n\left(\sum_{i=1}^{n} \theta_{pip_i}\theta_{dip_i}\right) - \left(\sum_{i=1}^{n} \theta_{pip_i}\right)\left(\sum_{i=1}^{n} \theta_{dip_i}\right)}{n\left(\sum_{i=1}^{n}(\theta_{pip_i})^2\right) - \left(\sum_{i=1}^{n} \theta_{pip_i}\right)^2} \tag{4.3}$$

where n is the number of samples taken. As we will see in the description of the actual algorithm, this parameter is used to build the lookup table for the index finger.

Run Time

At run time, all but those needed would be removed from the hand. IREDs 7 to 9 will be taken off so that we are only measuring the position of the tips of the index finger and thumb, the position of the MCP joint of the thumb and the orientation of the hand. The remaining IREDs are shown in Figure 4.4.

4.3 Hand Posture Estimation

With the position of IREDs 1 to 6 given by the Optotrak system, we can now estimate the hand posture. In this section, a simple hand model will be given first, following

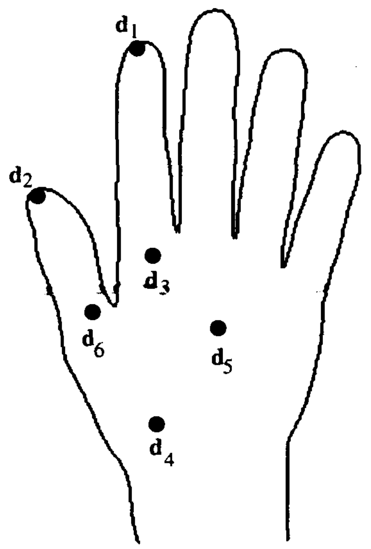

Figure 4.4: IRED placement for real-time experiment

which the algorithm that estimates the hand posture from a limited number of IREDs will be presented.

4.3.1 The Proposed Hand Model

The hand model presented in Chapter 2 is generic in the sense that the inner workings of the finger joints are represented by unknown "black box" functions $F_i^j(\boldsymbol{\theta}^j, \boldsymbol{p})$. Since accurate biomechanical simulation is not the intent of our application, only a simple model of the joint is used. This section describes this simple model and discusses some of its assumptions.

Applying the theory of opposition space [19, 31], we can represent the grasping behavior of the hand using only two fingers: the index finger and the thumb. Therefore, our simple hand model consists only of those two fingers and the palm. It is assumed that the index finger has four degrees of freedom: the MCP joint has two degrees of freedom while the DIP and PIP joints each have one degree of freedom. The thumb is assumed to have five degrees of freedom: the CMC and the MCP joints each have two degrees of freedom while IP has one degree of freedom. For a normal hand, it is easy to see that the fingertip, the PIP joint, the DIP joint, and the MCP joint of the index finger lie approximately on the same plane. Furthermore, any movement of the hand (without any external force applied to it) cannot change this property. Therefore, we can reasonably assume that all segments of the index finger lie on a single plane. Similarly, the tip of the thumb, the IP joint, and the MCP joint of the thumb lie approximately on the same plane, so we can again assume that the thumb segments between these joints lie on the same plane. As a result, a simple 2D model can be used to represent the fingers. The palm of the hand is simply represented by a plane and its orientation represents the orientation of the hand.

The hand coordinate system is such that the x axis is pointing in the direction of the index finger when it is fully extended. The y axis is pointing out of the back of the hand. The origin of the hand coordinate system is located at the MCP joint of the index finger. The z axis points to the right to complete a right handed coordinate system. Let $\boldsymbol{x}_g$, $\boldsymbol{y}_g$, and $\boldsymbol{z}_g$ be unit vectors in the global coordinate system pointing in the direction of the x, y and z axes of the hand coordinate system respectively. Let P be the plane containing the palm and its normal is just the vector $\boldsymbol{y}_g$ pointing out from the back of the hand.

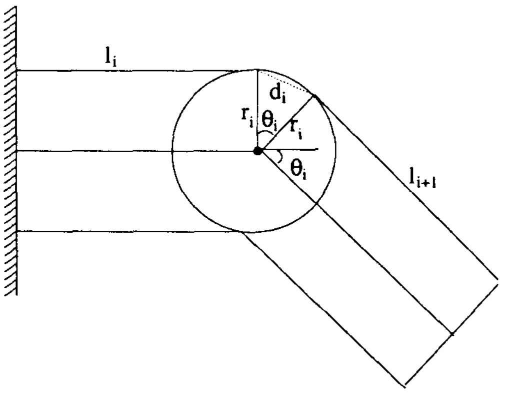

Figure 4.5: A circular joint

To develop a model for the finger, it is necessary to examine the finger joints more closely. The joints that control the rotation of the segments of the finger have a complicated structure of bones, muscles, and tendons. Since the IREDs are placed on the dorsal surface of the finger, our measurement of joint rotation is also affected by skin movements. Therefore, to simplify our model, we are going to assume that the joints are circular with radius r_i for the ith joint. From Figure 4.5, it is clear that as the finger joint i rotates, the distance between the end of segment i and the start of segment $i+1$ increases. This distance is represented as d_i. It is obvious that d_i is a function of the joint angle θ_i and the joint radius r_i. Using techniques from geometry, it can be shown that

$$d_i = 2r_i \sin \frac{\theta_i}{2} \tag{4.4}$$

where θ_i is the ith joint angle ordered from the proximal to distal end of the finger. Using d_i we can construct the functions $F_i^j(\boldsymbol{\theta}, r_i)$ for the circular joints by considering each joint as a *hidden link* of the finger. Since the position of the MCP joint is measured using an IRED, the first hidden link connecting the palm and the proximal segment of the finger is not needed. Thus, the finger can be represented using five links (including the two hidden links). This is shown in Figure 4.6.

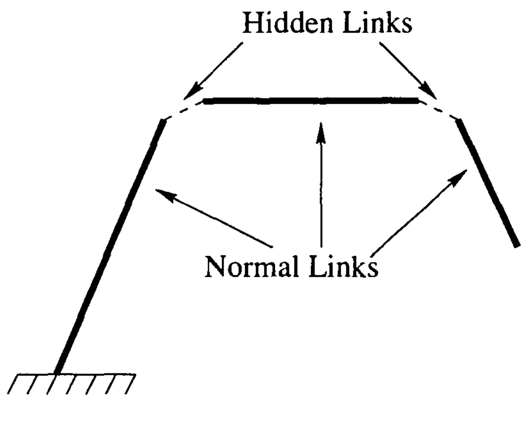

Figure 4.6: The five link representation of a finger

From standard forward kinematics, it is clear that

$$F_{i_x}^j(\boldsymbol{\theta}, r_i) = d_i \cos \phi_i \tag{4.5}$$

$$F_{i_y}^j(\boldsymbol{\theta}, r_i) = d_i \sin \phi_i \tag{4.6}$$

To derive the formula for ϕ_i, we can simply consider the finger as an ordinary five link manipulator with the special property that the sum of the two joint angles at each end of each hidden link sum to the value of the corresponding original joint angle of the joint in which the hidden link represents. From Figure 4.7, it is obvious that the angles labeled a and b are the same and $a + b = \theta_i$. Therefore, we can express ϕ_i as:

$$\begin{aligned} \phi_0 &= \theta_0 \\ \phi_i &= \phi_{i-1} + \theta_{\lfloor \frac{i}{2}+1 \rfloor}/2 \end{aligned} \tag{4.7}$$

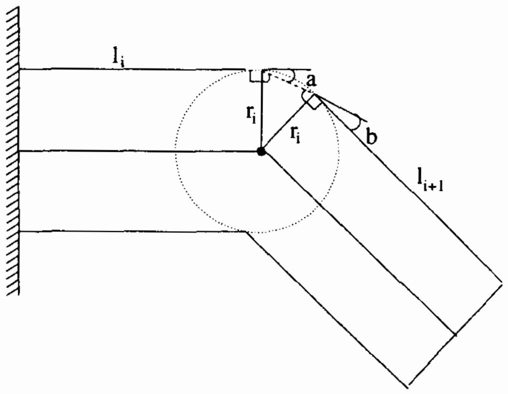

Figure 4.7: Calculation of ϕ_i

Extending the forward kinematic equations presented in Chapter 2, the location of the tip of the index finger can be expressed explicitly as a function of the modified joint angles ϕ_i.

$$\begin{aligned} x &= l_1 \cos\phi_1 + d_1 \cos\phi_2 + l_2 \cos\phi_3 + d_2 \cos\phi_4 + l_3 \cos\phi_5 \\ y &= l_1 \sin\phi_1 + d_1 \sin\phi_2 + l_2 \sin\phi_3 + d_2 \sin\phi_4 + l_3 \sin\phi_5 \end{aligned} \tag{4.8}$$

where (x, y) is the coordinate of the fingertip in the plane in which the index finger lies in. A similar equation can be set up for the thumb. Since both the index finger and the thumb have more DOF at the base, this simple model can only calculate a subset of their joint angles. The following is the convention used for naming the joint angles:

- Index Finger:
  - θ_0^{index} is the rotation about the y axis at the MCP joint.
  - θ_1^{index} is the rotation about the z axis at the MCP joint.
  - θ_2^{index} is the rotation about the z axis at the PIP joint.
  - θ_3^{index} is the rotation about the z axis at the DIP joint.
- Thumb:
  - θ_0^{thumb} is the rotation about the y axis at the CMC joint.
  - θ_1^{thumb} is the rotation about the z axis at the CMC joint.
  - θ_2^{thumb} is the rotation about the y axis at the MCP joint.
  - θ_3^{thumb} is the rotation about the z axis at the MCP joint.
  - θ_4^{thumb} is the rotation about the z axis at the IP joint.

The superscripts *index* and *thumb* are used to distinguish the joint angles of the index finger and the thumb. The values of θ_2^{index}, θ_3^{index}, θ_3^{thumb}, and θ_4^{thumb} can be determined using the lookup table at run time. The values of the remaining joint angles are calculated separately.

Since θ_0^{index} represents the rotation of the MCP joint of the index finger about the y axis, we can calculate it by taking the dot product of the vector $\boldsymbol{x}_g$ with the projection of the vector going from the position of the MCP joint to the index fingertip onto the plane P representing the palm. Let $\boldsymbol{t} = \frac{\boldsymbol{d}_1 - \boldsymbol{d}_3}{||\boldsymbol{d}_1 - \boldsymbol{d}_3||}$ be the unit vector from the MCP joint to the fingertip. This calculation is illustrated in Figure 4.8.

$$\begin{aligned} \boldsymbol{p} &= \boldsymbol{t} - \boldsymbol{y}_g(\boldsymbol{t} \cdot \boldsymbol{y}_g) \\ \theta_0^{index} &= \cos^{-1} \frac{\boldsymbol{x}_g \cdot \boldsymbol{p}}{||\boldsymbol{p}||} \end{aligned} \tag{4.9}$$

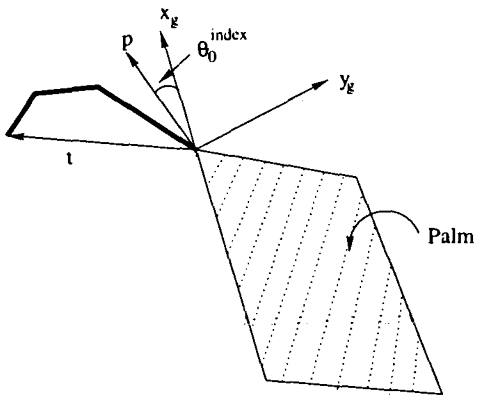

Figure 4.8: Calculation of θ_0^{index}

Notice that the angle α between $\boldsymbol{t}$ and the plane P is the sum of the joint angle θ_1^{index} and an angle β, where β is the angle between $\boldsymbol{t}$ and the vector $(\boldsymbol{d}_8 - \boldsymbol{d}_3)$. The angle α can be easily obtained by taking the dot product of $\boldsymbol{x}_g$ and $\boldsymbol{t}$. We can obtain β by rotating the vector $(\boldsymbol{d}_8 - \boldsymbol{d}_3)$ to align with the vector $[1,0,0]_g$ and moving it to the origin. Then, using the value of θ_2^{index} and θ_3^{index} from the lookup table, we can obtain the position vector $\boldsymbol{t}' = [t'_x, t'_y, 0]$ of the fingertip evaluating the forward equation once. The cosine of the angle β is then the dot product of the vector $[1,0,0]_g$ and $[t'_x, t'_y, 0]$. Therefore, θ_1^{index} can be easily obtain by

$$\begin{aligned}\theta_1^{index} &= \alpha - \beta \\ &= \left(\frac{\pi}{2} - \cos^{-1}\left(\frac{\boldsymbol{t}\cdot\boldsymbol{y}_g}{\|\boldsymbol{t}\|}\right)\right) - \cos^{-1}\left(\frac{t'_x}{\|\boldsymbol{t}'\|}\right) \qquad (4.10)\end{aligned}$$

The calculation of θ_1^{index} is illustrated in Figure 4.9.

Since we are measuring the position of the MCP joint of the thumb, θ_0^{thumb} and θ_1^{thumb} can be calculated very easily. Let $\boldsymbol{m} = \frac{\boldsymbol{d}_6 - \boldsymbol{d}_4}{\|\boldsymbol{d}_6 - \boldsymbol{d}_4\|}$ be the unit vector that goes from the CMC joint of the thumb to the MCP joint. The joint angle θ_0^{thumb} can be calculated by projecting the vector $\boldsymbol{m}$ onto the plane P. The dot product of this vector and $\boldsymbol{x}_g$ gives the cosine of this angle. This is illustrated in Figure 4.10. Then θ_0^{thumb} can be expressed as

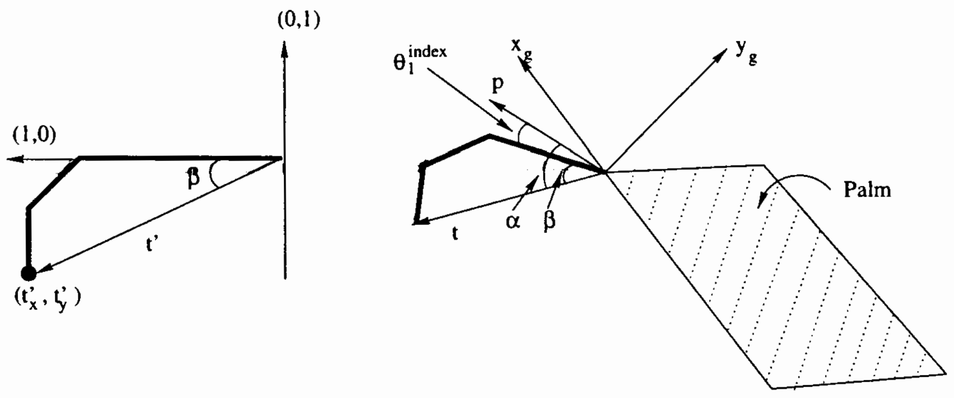

Figure 4.9: Calculation of θ_1^{index}

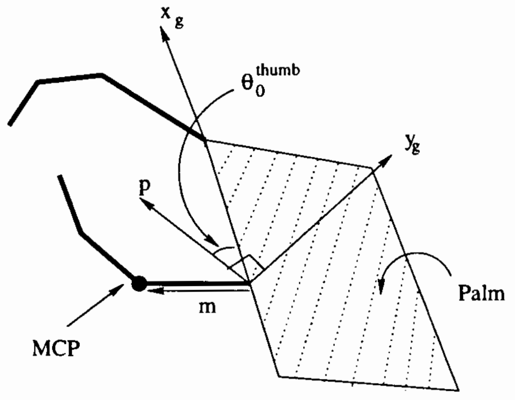

Figure 4.10: Calculation of θ_0^{thumb}

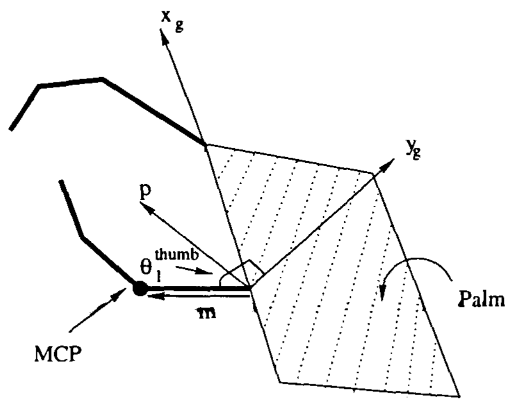

Figure 4.11: Calculation of θ_1^{thumb}

$$\begin{aligned} \boldsymbol{p} &= \boldsymbol{m} - \boldsymbol{y}_g(\boldsymbol{m} \cdot \boldsymbol{y}_g) \\ \theta_0^{thumb} &= \cos^{-1} \frac{\boldsymbol{x}_g \cdot \boldsymbol{p}}{||\boldsymbol{p}||} \end{aligned} \tag{4.11}$$

The cosine of the joint angle θ_1^{thumb} is the dot product of $\boldsymbol{m}$ and $\boldsymbol{y}_g$. Since $\boldsymbol{y}_g$ is pointing out of the back of the hand, we have to subtract $\pi/2$ from the resulting angle. Therefore, we have

$$\theta_1^{thumb} = \cos^{-1}(\boldsymbol{m} \cdot \boldsymbol{y}_g) - \frac{\pi}{2} \tag{4.12}$$

The calculation of θ_1^{thumb} is illustrated in Figure 4.11.

The calculation of θ_2^{thumb} can be done using a similar method as the calculation of θ_0^{thumb}. However, in this case, we are projecting the vector $\boldsymbol{\tau} = \frac{\boldsymbol{d}_2 - \boldsymbol{d}_4}{||\boldsymbol{d}_2 - \boldsymbol{d}_4||}$ that goes from the MCP joint of the thumb to the tip onto the plane P' defined by the positions of the MCP joint of the thumb, the CMC joint of the thumb and the MCP joint of the index finger. Let $\boldsymbol{\eta}$ be the unit normal of the plane P'. The angle θ_2^{thumb} can be calculated by

$$\begin{aligned} \boldsymbol{p} &= \boldsymbol{\tau} - \boldsymbol{\eta}(\boldsymbol{\tau} \cdot \boldsymbol{\eta}) \\ \theta_2^{thumb} &= \cos^{-1}(\boldsymbol{p} \cdot \boldsymbol{m}) \end{aligned} \tag{4.13}$$

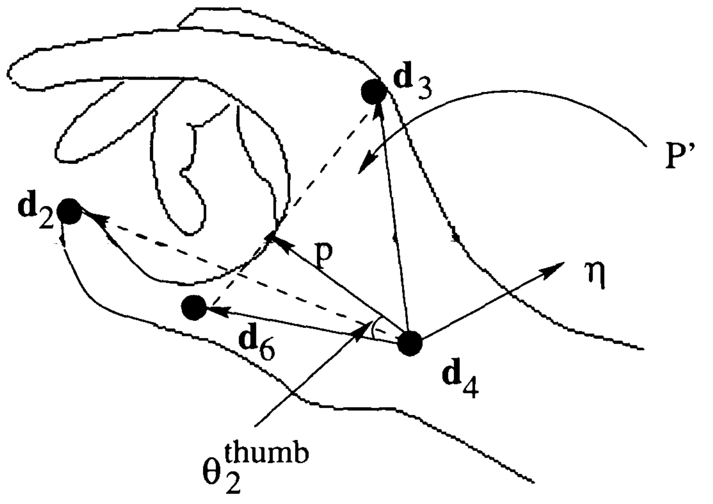

Figure 4.12: Calculation of θ_2^{thumb}

With the above expressions for the joint angles of the index finger and thumb, the simple hand model is complete. The following section will demonstrate how this model is used to estimate the hand posture using live data from the Optotrak system.

4.3.2 The Algorithm

Like Amaya's algorithm, this algorithm can also be divided into two stages: the preprocessing stage and the run time stage. The preprocessing step builds the lookup table while the run time step simply searches the lookup table for the solution using an appropriate index.

The algorithm for building the lookup table is very simple. The algorithm *BuildIndexTable* is given in Appendix C.1. As shown in Figure 4.4, we are measuring the location of the tip and the MCP joint of the index finger, so the entire kinematic chain of the index finger consists of five links (including the two hidden links). Similarly, there are three links between the tip and the MCP joint of the thumb. Before we can build a table, we have to select a way to index it. Instead of using the coordinate of the tip of the finger as the index as in Amaya's algorithm, we have chosen to use

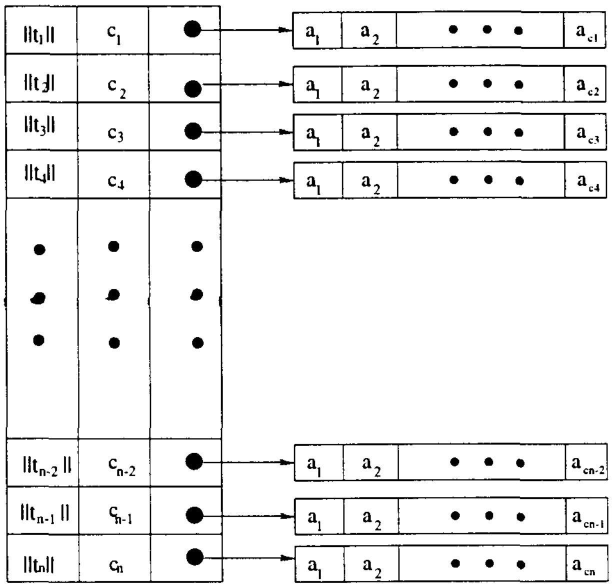

Figure 4.13: The lookup table build using the algorithm described in the thesis

the distance $\|t\|$ between the fingertip and the MCP joint as the index. Since this distance is a floating point number, we convert it to an integer before we use it as the index. The index has a range of 0mm to 200mm. By default, the interval between successive indices is 1mm. If denser indices are required, it can be specified using the parameter p as discussed in Appendix C.1. For any values of p less than 1, we have to scale $\|t\|$ up by corresponding factor $1/p$ and use the integer part as the index. The next step is to evaluate the forward kinematic equation for various values of the joint angles and insert these joint angle values into the lookup table using the calculated index for each entry. It is possible that more than one set of joint angles have the same index. In that case, an array is used to store all of them and the corresponding entry in the lookup table points to the array. The index finger and the thumb have different parameters, so a different lookup table is built for each. The lookup table build using this algorithm is shown in Figure 4.13. The numbers c_i in this figure represent the number of sets of joint angles a_i having the same index $\|t_i\|$.

Since the value of θ_1 does not affect the value of the index, 0 can simply be assigned as its value throughout. Since there are only three links between the tip and the MCP joint of the thumb, we can easily transform the algorithm *BuildIndexTable* in Appendix C.1 into an algorithm *BuildThumbTable* by removing the inner FOR loop and change the five link forward kinematic equations to the three link equations.

As an alternative to *BuildIndexTable*, we can use the parameter ρ to decrease the size of the lookup table. Since we know the ratio between the joint angles θ_2 and θ_3, we can remove the inner FOR loop from the BuildIndexTable and set θ_3 to $\rho\theta_2$. Because only one FOR loop is required, the size of the lookup table only increases linearly as the precision of the joint angles increases. This significantly increases the table building process as well.

At run time, the lookup table is searched to obtain a subset of the joint angles of the finger. The remaining joint angles are computed on the fly. The run-time algorithm is given in Appendix C.2. The norm of the vector $\boldsymbol{t}$ is computed to obtain the index. Because of the nature of our data, there might be more than one set of joint angles that have the same index. An array is used to store all of them. Since this array is relatively small in size, a linear search algorithm can be employed to find the set of joint angles according to some criteria. In our application, it is assumed that the relative angular change from one frame to the next is small. Therefore, the searching criterion is to minimize the change from the set of joint angles in the previous frame. It is possible that the table entry corresponding to the computed index is empty. In this case, the algorithm would then try to find the closest table entry which contains at least one set of joint angles.

The algorithm *GetIndexJointAngles* finds the joint angles of the index finger. Simple modifications to *GetIndexJointAngles* can be made to change it into *GetThumbJointAngles* which finds the joint angles of the thumb. The necessary modifications are:

- Change the first line from $||\boldsymbol{t}|| := ||\boldsymbol{d}_1 - \boldsymbol{d}_3||$ to $||\boldsymbol{t}|| := ||\boldsymbol{d}_2 - \boldsymbol{d}_6||$
- Replace θ_1^{index} and θ_2^{index} with θ_3^{thumb} and θ_4^{thumb}
- Delete all references of θ_3^{index}
- Change the three link forward equations that calculates the value of t'_x and t'_y to the two link equations.

- Replace the calculation of θ_0^{index} by the equations (4.11), (4.12) and (4.13).
- Change the last line to RETURN $\{\theta_0^{thumb}, \theta_1^{thumb}, \theta_2^{thumb}, \theta_3^{thumb}, \theta_4^{thumb}\}$.

4.4 Implementation

A complete data collection system which integrates data collection, estimation and display of hand postures has been implemented to allow the experimenters to collect accurate 3D hand posture data.

4.4.1 The Data Collection System

There are several modules in the data collection system: *getLength*, *getRatio*, *getData*, and *ikHand*. The first two modules are used for the calibration step while the last one is used during the run-time step. With the IREDs placed on the subject's hand as shown in Figure 4.3, the first module *getLength* sets the Optotrak camera to a frequency of 60Hz and measures this calibration hand posture for 0.5 seconds for a total of 30 frames. The lengths of each finger segments are computed at each frame and the average lengths are then written to a file *length.dat*. When the second module *getRatio* is invoked, the subject is asked to freely rotate and bend his fingers. The *getRatio* module collects this data for six seconds and computes the parameter ρ used to build the lookup table using equation (4.3). The value of ρ is written to *ratio.dat*. Before the start of the run-time stage, the thickness of the finger at each joint would have to be measured using a caliper. The values of the thickness data are stored in *thickness.dat*. Now, we have completed the calibration stage.

At the run-time stage, IREDs 7, 8, and 9 can be taken off the hand. After that, the module *ikHand* is invoked. It again sets the Optotrak camera to 60Hz and continuously collects the positional data of the IREDs. The program computes the hand posture at each frame from these positional data and displays the resulting hand posture onto the SGI monitor. The collected data is not written to a file since it is very slow to display and access the file system at the same time. If the data needs to be stored, the module *getData* can be used to collect and store the data into a file *hand.dat*. The stored data can later be viewed using a modified version of the module *ikHand* which gets the input from a file instead of getting it from the Optotrak system. A sample display is shown in Figure 4.14.

Implementation Details

The main data structure used in this system is the lookup table. It is implemented as a dynamic array. Each element of the table is another dynamic array. This is used to store different sets of joint angles having the same index ($\|\boldsymbol{t}\|$). At the run-time stage, the value of the index is computed from the live 3D data and this value is used directly to index into the lookup table to obtain the array of sets of joint angles having this index value. Since size of this array is very small (usually around three and four), a linear search algorithm is used to determine which set of joint angles has the minimum deviation from the joint angles of the previous frame.

All modules are implemented in C++ on a Silicon Graphics Indigo2. The Open Inventor Library is used to do the vector calculations and rendering of the hand.

4.5 Discussion

The table based inverse kinematic algorithm presented in the previous section uses the lookup table approach introduced by Amaya et al. to solve the inverse kinematic problem. It addresses some of the weaknesses of Amaya's algorithm by making additional assumptions about the finger and joint configurations. In particular, it utilizes the notion of hidden joints to simulate the behavior of the finger joint and account for the error introduced by skin movement. In addition, this algorithm is stable in the sense that it will always return a feasible solution. It is easy to see from the algorithm that it always returns the optimal solution in which the end-effector position error is minimized. Furthermore, the computational speed is not jeopardized by the accuracy and stability of the algorithm. The algorithm is very fast because no coordinate transformation needs to be explicitly performed. This also means that numerical errors associated with the transformation of coordinate system are eliminated as well.

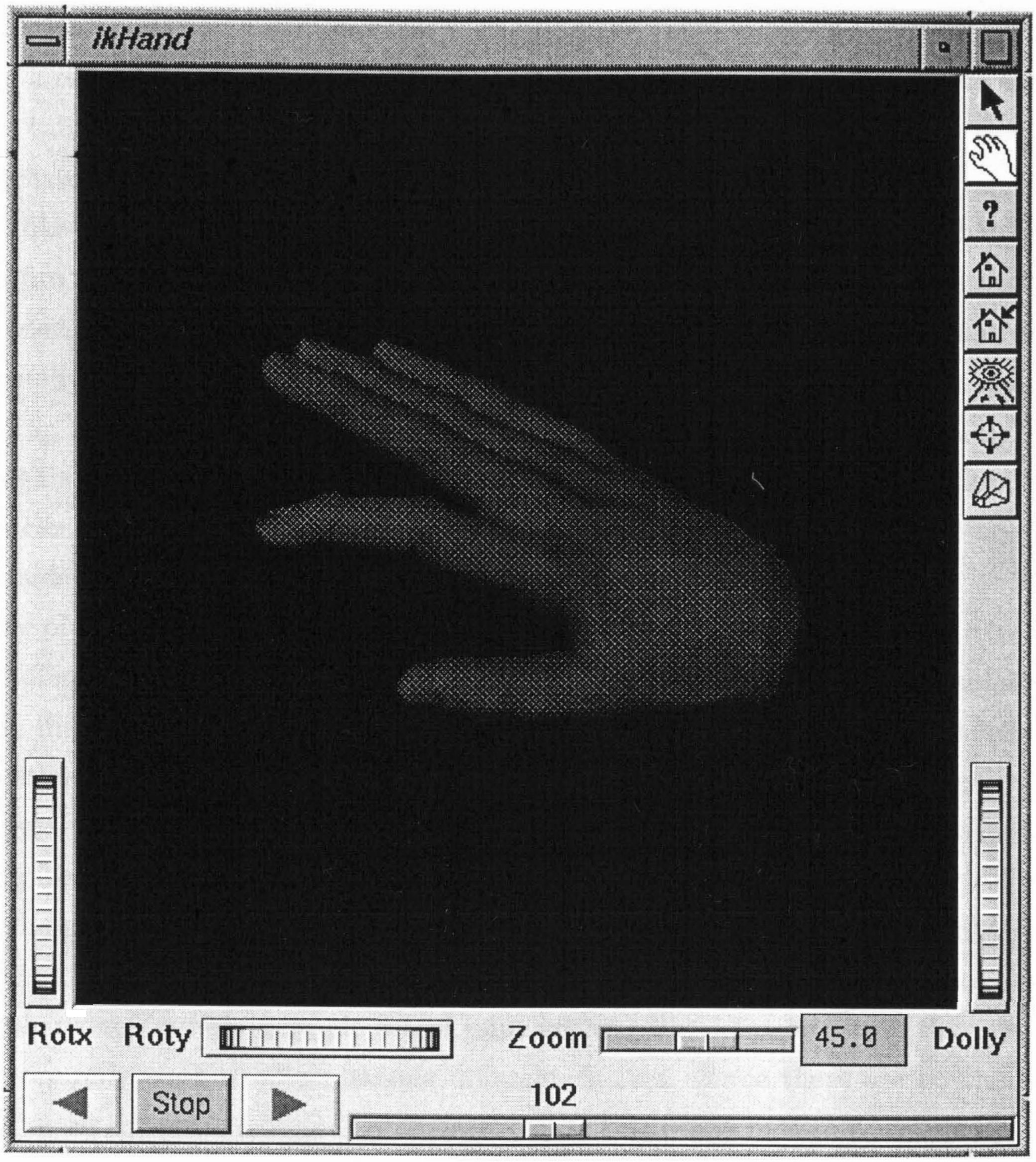

Figure 4.14: Display of collected data

However, unlike Amaya's algorithm in which the lookup table has to be regularized to obtain uniform table entries, there is no clear way to do regularization on the lookup table built by this algorithm. Since there can be more than one set of joint angles stored in a single entry, simple interpolation of the table entries cannot be done to obtain a regularized table. Furthermore, even if interpolation is applied to two entries in the table to obtain an entry between them, there can be no guarantee that the interpolated joint angles will produce the desired index.

Unlike the Jacobian based algorithms presented in Chapter 3, this table based algorithm does not explicitly address the singularity problem. Instead, this problem is avoided by finding the closest feasible solution in the workspace. Also, joint limits are usually hard to enforce using Jacobian based algorithms. It is usually done by clamping the joint angles to the maximum or the minimum limit. Handling joint limits this ways may produce unnatural looking joint angles. In our algorithm, joint limits can be explicitly specified in the table building process, so any solutions that lie outside the joint limits will not be returned.

One of the major drawbacks of this algorithm is that it cannot be readily extended to handle other manipulators. The algorithm is specifically designed to simulate the human finger. Furthermore, it assumes that the relative angular movement between the previous frame and the current frame is small. However, even in our application, it is possible for the Optotrak camera to miss some frames as the IREDs became hidden from its view. As a result, the next valid frame may be drastically different from the previous valid frame. In this case, the solution returned by the algorithm will be extremely unpredictable. Fortunately, there are many entries in the lookup table that contain only a single set of joint angles. The joint angles of the previous frame are only used to select among different choices. Since there are no choices if one of those entries is selected by the algorithm, this problem would vanish.

Chapter 5

Experiment and Results

This Chapter describes an experiment which applies the algorithm presented in Chapter 4 in a real-time setting and discusses the results obtained from the experiment. The main purpose of the experiment is to evaluate the accuracy of the algorithm and demonstrate that it can be effectively used in a real-time setting.

5.1 The Real-time Experiment

In order to evaluate the accuracy of this table based method, a practical experiment was performed. After setting up the equipment, the preprocessing stage begins by placing the subject's hand palm down on a flat table. The IREDs were placed as shown in Figure 4.3. The equipment setup is shown in Figure 5.1. This calibration posture of the subject's hand was sampled at 60Hz for 0.5 second. The lengths of the finger segments were computed at each frame. Then, the average lengths of the segments from the 30 sampled frames were used as the measured lengths of the finger segments to build the lookup table. The subject was then asked to freely move and rotate the index finger and the thumb. This data was sampled at 60Hz for six seconds. The PIP and DIP joint angles of the index finger were computed at each frame and used to obtained the parameter ρ using equation (4.3). The lookup table was built using an accuracy of $p = 0.1$ which means value of the joint angles stored in each entry of the table is accurate to 1/10th of a degree. The hand motion data was again

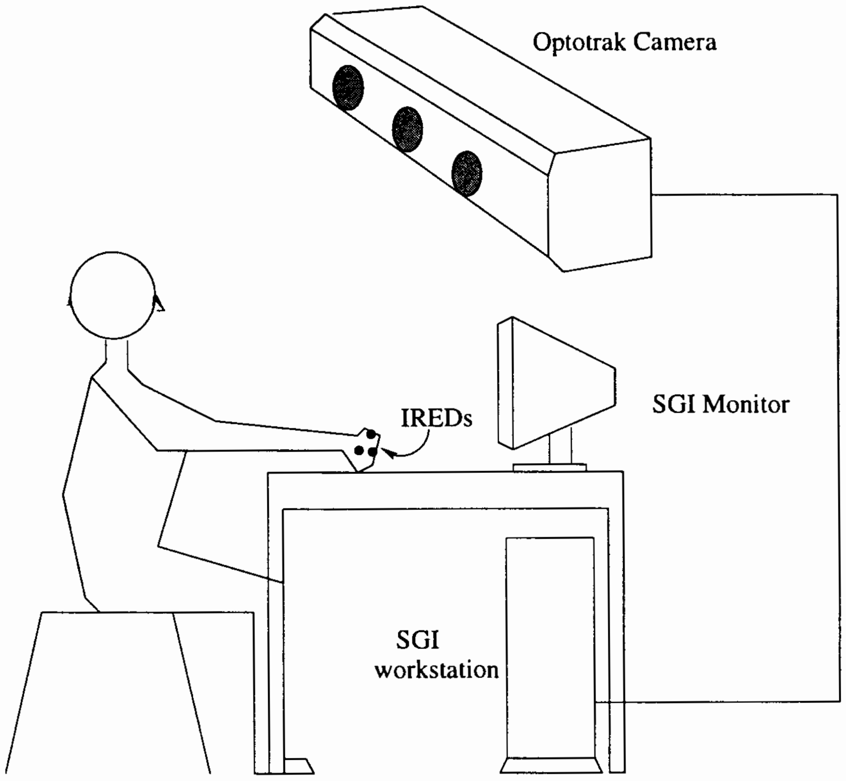

Figure 5.1: Equipment setup

collected at 60Hz for six seconds for a total of 360 frames.

During the run-time stage, only six IREDs were needed as shown in Figure 4.4. To determine the accuracy of the methods, the IRED placement was not changed from the preprocessing stage. Since there is an IRED on every joint of both the index finger and the thumb, the actual joint angles and the position of the fingertips can be easily measured. However, the only IRED information given to the inverse kinematic algorithm was the position of the six IREDs required to compute the joint angles in run-time. The computed joint angles were compared with the actual joint angles to determine the accuracy of our method. In addition, the actual position of the fingertip is compared with the computed fingertip position obtained by evaluating the forward kinematic calculations using the computed joint angles.

5.2 Difficulties Encountered

One of the major difficulties encountered during the experiment was the occlusion of IREDs from the Optotrak camera. It is obvious that the hand is not a flat surface, so as the hand rotates and the fingers bend, one or more of the IREDs placed on the hand are inevitably going to be blocked from the view of the camera. As a result, much of the data returned by the Optotrak system was rendered useless. This also severely restricts the workspace of the hand. Furthermore, the quality of solution from our algorithm can be severely affected since the table based algorithm requires the joint angles of the previous frame as initial approximation for the joint angles in the current frame. If IREDs in several consecutive frames between one valid frame of data and the next were blocked from the camera, the algorithm would have to use the last valid frame (not the previous frame) to approximate the joint angles of the current valid frame. This is usually not a good approximation and the algorithm will probably return poor results. One possible solution to this problem would be to surround the subject with Optotrak cameras, so all the IREDs will be in the view of at least one camera at all times. This solution, however, cannot be implemented at the time of this writing because of the high cost of the equipment.

Another problem encountered during the experiment was the speed of rendering the hand in real-time. As discussed in Chapter 2, the graphic hand is composed of many polygons. A significant amount of time is spent in drawing them. In addition, deformation calculations are performed in real-time to simulate the bending and stretching of the finger. With the additional overhead of polling data from the Optotrak system and computing joint angles using our algorithm, the graphic hand was not able to be rendered quickly enough to give the illusion of smooth hand movement. However, after hand tuning the matrix computations used in the deformation and speeding up the calculation performed for the inverse kinematic algorithm, the graphic hand can now be rendered at approximately 12 frames per second on a Silicon Graphics Indigo2 Extreme. As suggested by Welman [45], 10 frames per second is the minimum for interactive applications to maintain the illusion of continuous interactive control.

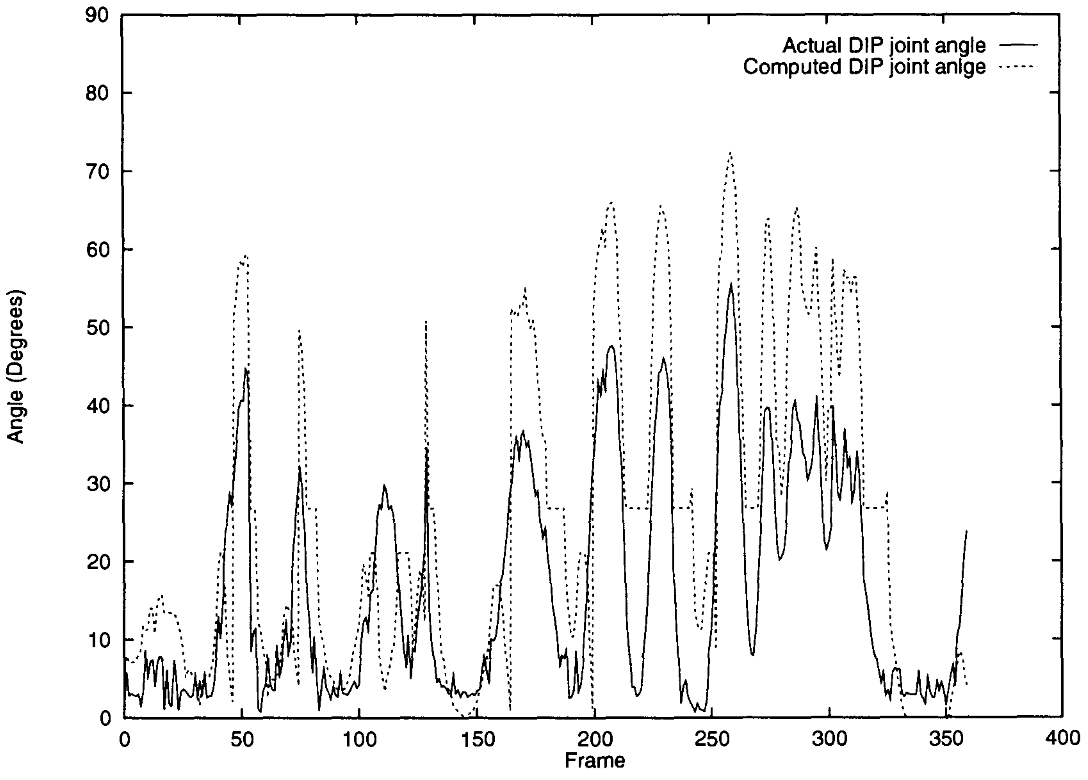

Figure 5.2: Comparison of θ_3^{index}

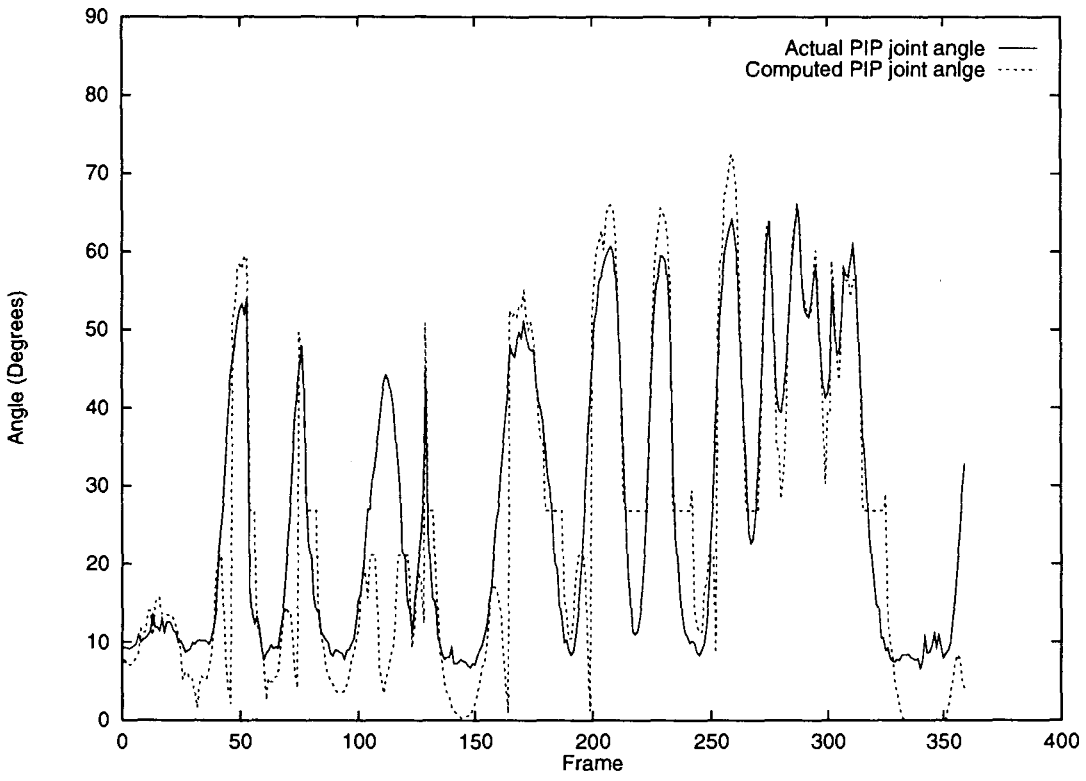

Figure 5.3: Comparison of θ_2^{index}

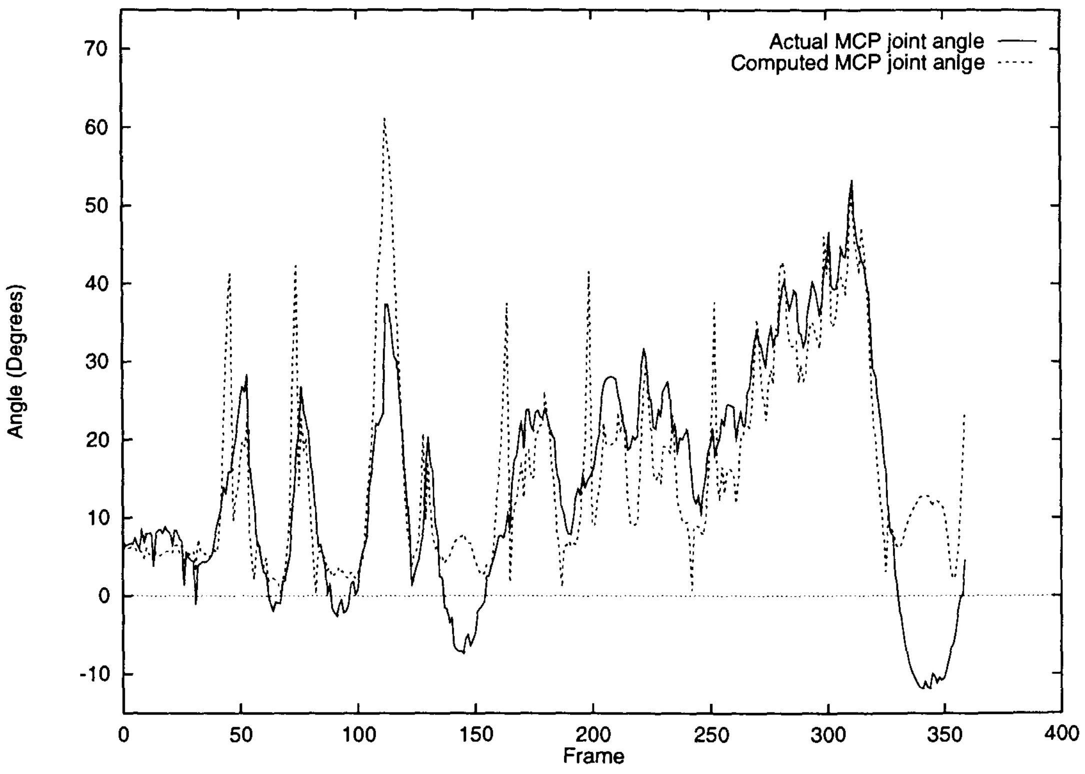

Figure 5.4: Comparison of θ_1^{index}

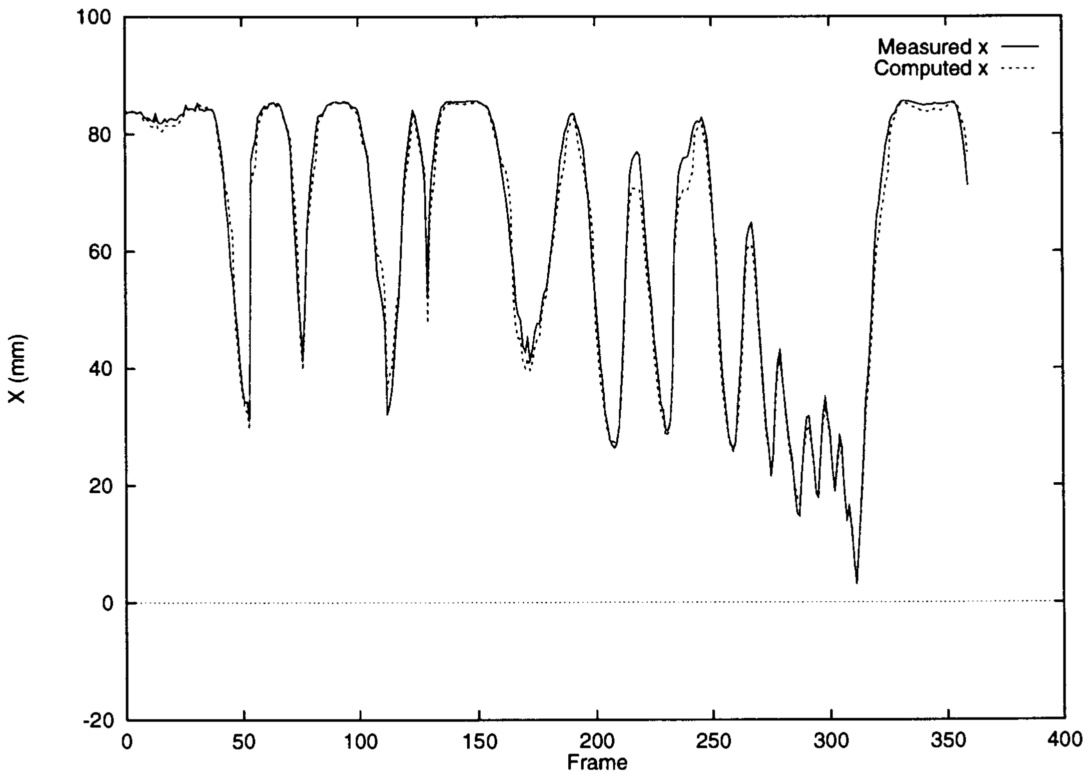

Figure 5.5: Comparison of the x coordinate of the tip of the index finger

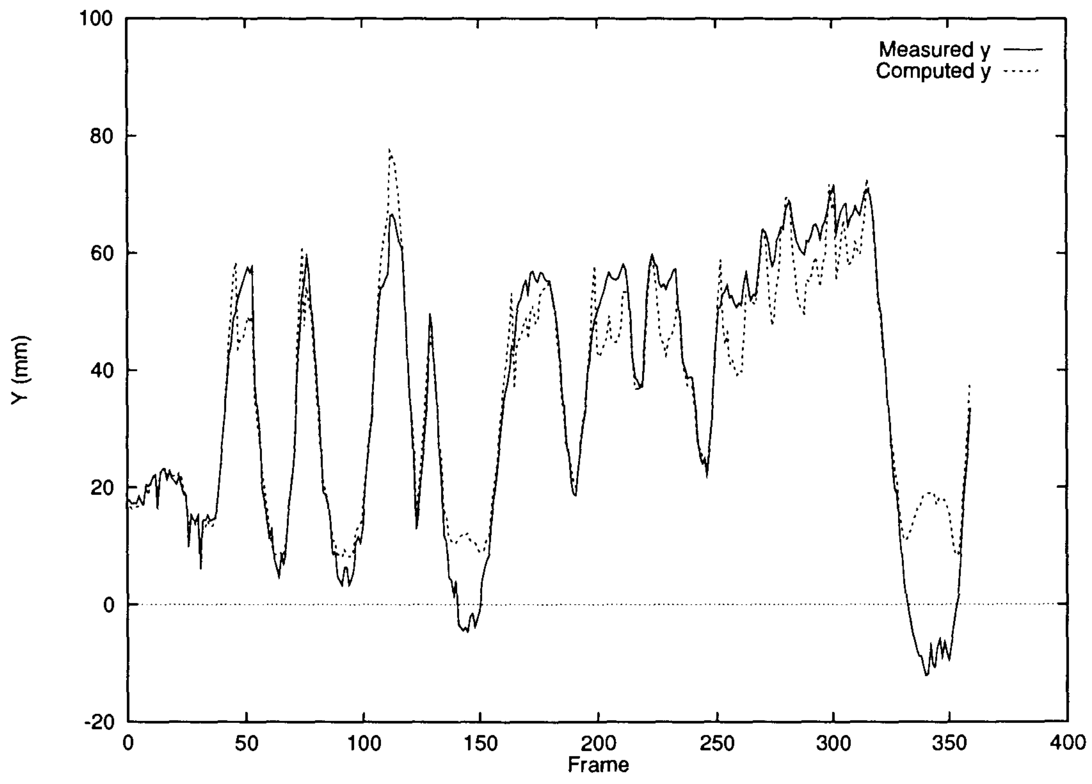

Figure 5.6: Comparison of the y coordinate of the tip of the index finger

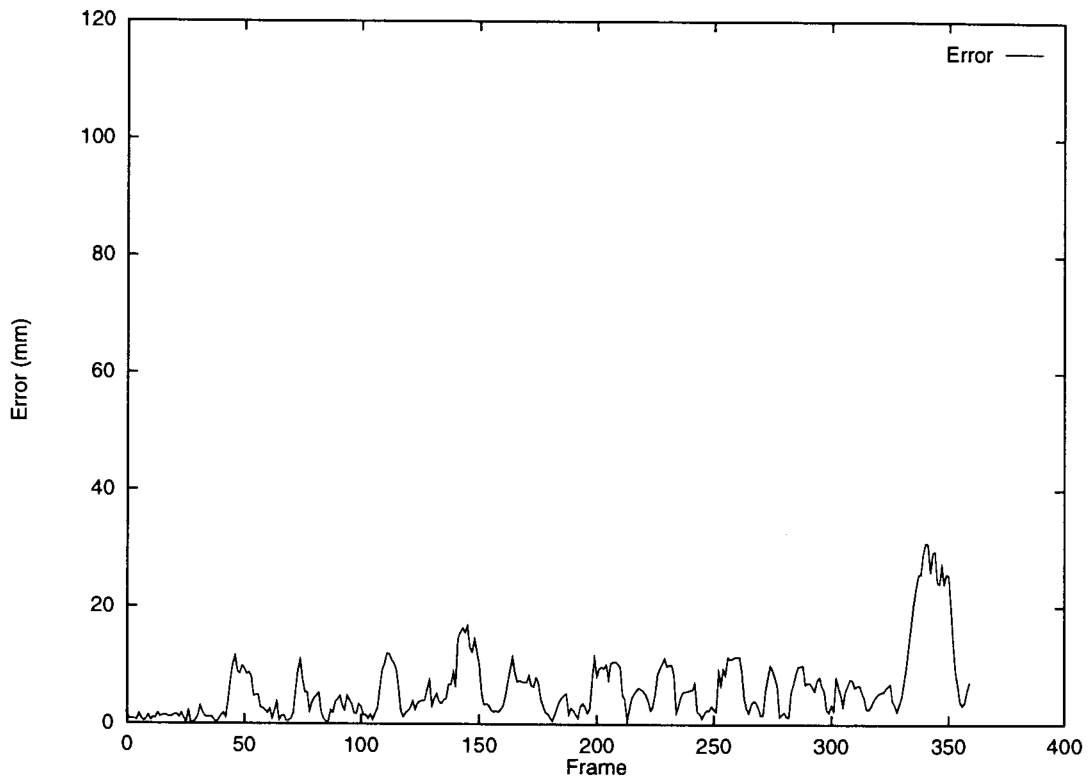

Figure 5.7: The error between measured tip location and computed tip location of the index finger

5.3 Result and Analysis

The results of the computations are plotted as graphs. The independent variable in these plots is time which is represented by the frame number. Comparisons between both joint angles and end-effector positions are made for the index finger and the thumb.

5.3.1 The Index Finger

Since the DIP, PIP and MCP joints as well as the tip of the index finger all lie on the same plane and that we know the location of the MCP joint and the tip, we can calculate the angle θ_0^{index} exactly in the run-time stage. Therefore, this joint angle is not compared. Figures 5.2, 5.3, and 5.4 show the plot of the comparisons between the measured joint angles θ_1^{index}, θ_2^{index}, and θ_3^{index} of the index finger and the computed joint angles.

From Figure 5.4, we can see that when the actual joint angle decreases to negative, the computed joint angle tends to go in the opposite direction. This phenomenon can be observed at frames 130 to 155 and 320 to 355. Intuitively, if the MCP joint angle is negative, it means that the finger is bending backwards. This causes the value of $||\boldsymbol{t}||$ to decrease, similar to the situation when the finger is bending forward. Since only the value of $||\boldsymbol{t}||$ is used as the index for the lookup table, the algorithm cannot distinguish between the two different situations. Therefore, the algorithm just assumes that the finger is bending forward which is the more natural movement. As a result, the value of the MCP joint angle increases. During typical grasping tasks, it is rarely the case if at all, that the index finger needs to bend backwards to complete the given task. Therefore, when the algorithm was designed, it is assumed that the index finger can only bend forward.

Since the ultimate goal of inverse kinematics is to obtain a set of joint angles to position the end-effector as close to a desired location as possible, we have plotted several graphs comparing the desired x and y coordinates and the x' and y' coordinates returned by our inverse kinematic algorithm.

The negative angular values of frames 130 to 155 and 320 to 355 in Figure 5.6

confirmed that the index finger is indeed bending slightly backwards. Figure 5.7 shows the error between the desired coordinate (x_i, y_i) and the computed coordinate (x'_i, y'_i). The error e_i of the ith frame is computed using the formula

$$e_i = \sqrt{(x_i - x'_i)^2 + (y_i - y'_i)^2} \tag{5.1}$$

The average error of Figure 5.7 is 6.01mm.

5.3.2 The Thumb

The data collected for the thumb is compared in a similar way. Since we measure the position of the CMC and the MCP joint of the thumb at run-time, we can calculate the joint angles θ_0^{thumb} and θ_1^{thumb} exactly. Also, we know that the MCP and the IP joints as well as the tip of the thumb all lie on the same plane, we can calculate the joint angle θ_2^{thumb} exactly using equation (4.13). Therefore, the only two joint angles obtained using the lookup table are θ_3^{thumb} and θ_4^{thumb}. The comparison results are shown in Figures 5.8 and 5.9.

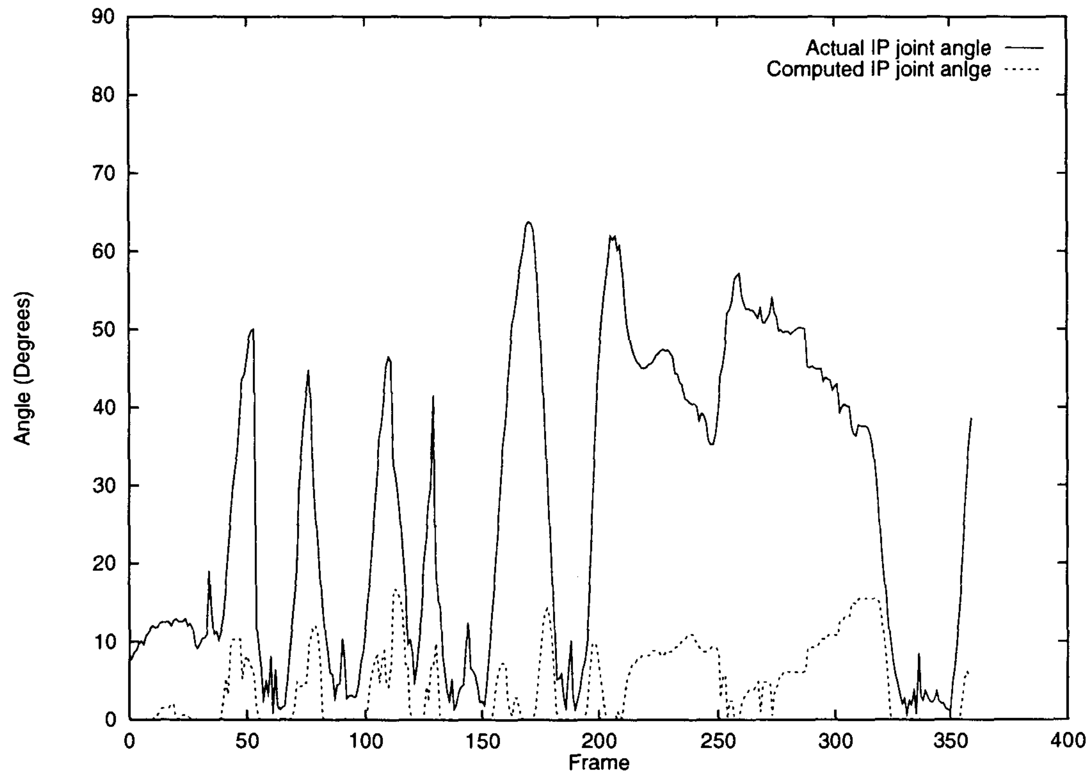

Figure 5.8: Comparison of θ_4^{thumb}

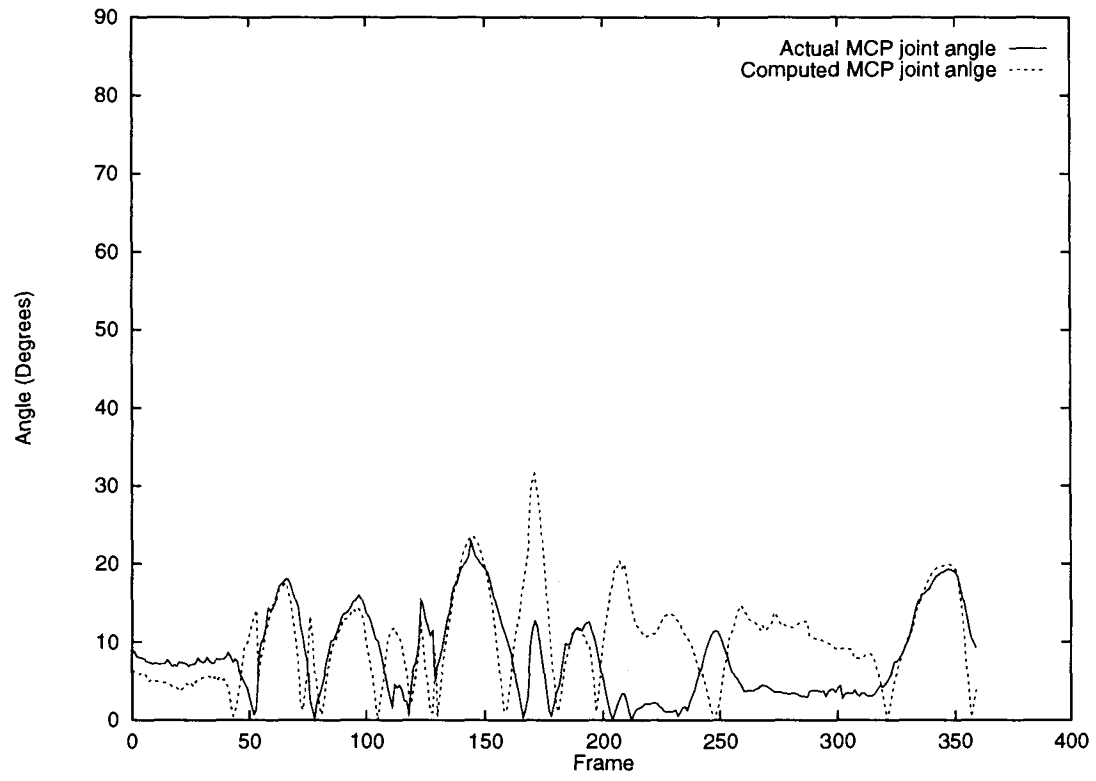

Figure 5.9: Comparison of θ_3^{thumb}

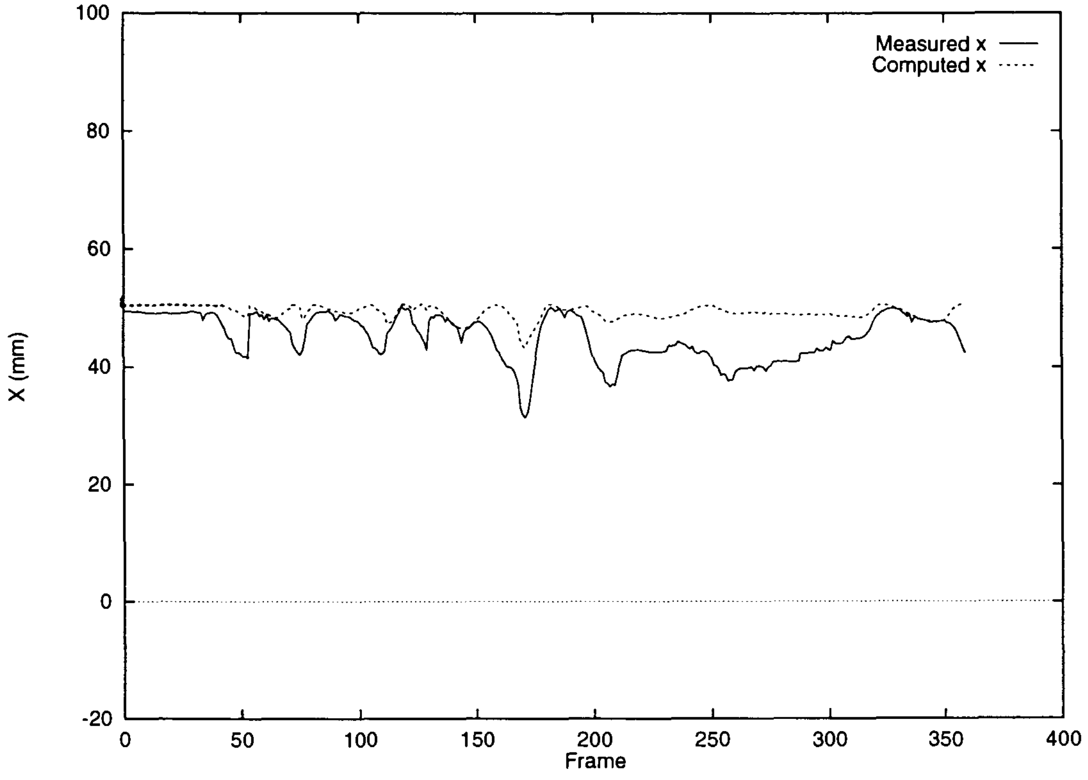

Figure 5.10: Comparison of the x coordinate of the tip of the thumb

Notice that the difference between the measured joint angles and computed joint angles from frame 200 to 300 in both Figures 5.8 and 5.9 are quite large. The values of the measured joint angles are large in that interval while the values of the computed joint angles are relatively small. This error could be caused by any one or a combination of two sources. Since the thumb is extremely dextrous, the skin movement on the thumb is very significant. An IRED was placed on top of the MCP joint and on the tip of the thumb to measure their distance. This distance is the value of the index into the lookup table. If the position of the IRED measuring the position of the MCP joint is severely affected by skin movement, the value of the index would not be computed very accurately. This could cause serious error in our computation. Also, since the radius of the IP joint (including the IRED thickness) is estimated using a ruler, it might not be very accurate. This introduces additional error into our computation. Placing the IREDs on rigid materials that are tightly fixed to the thumb

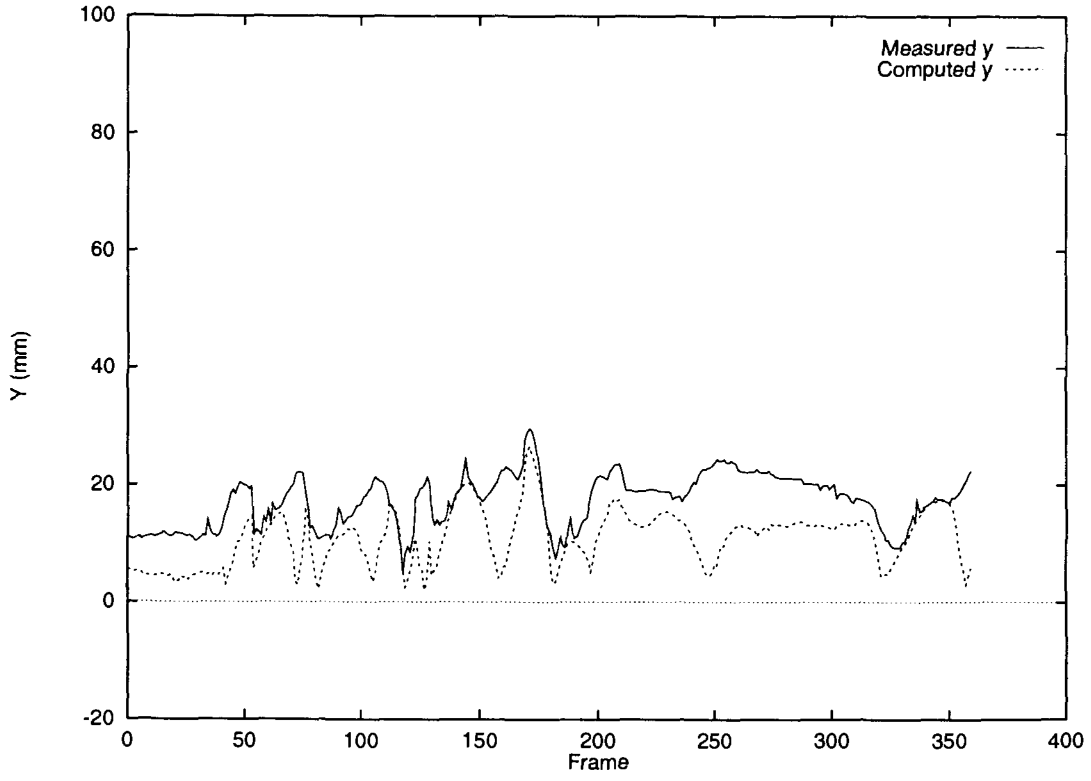

Figure 5.11: Comparison of the y coordinate of the tip of the thumb

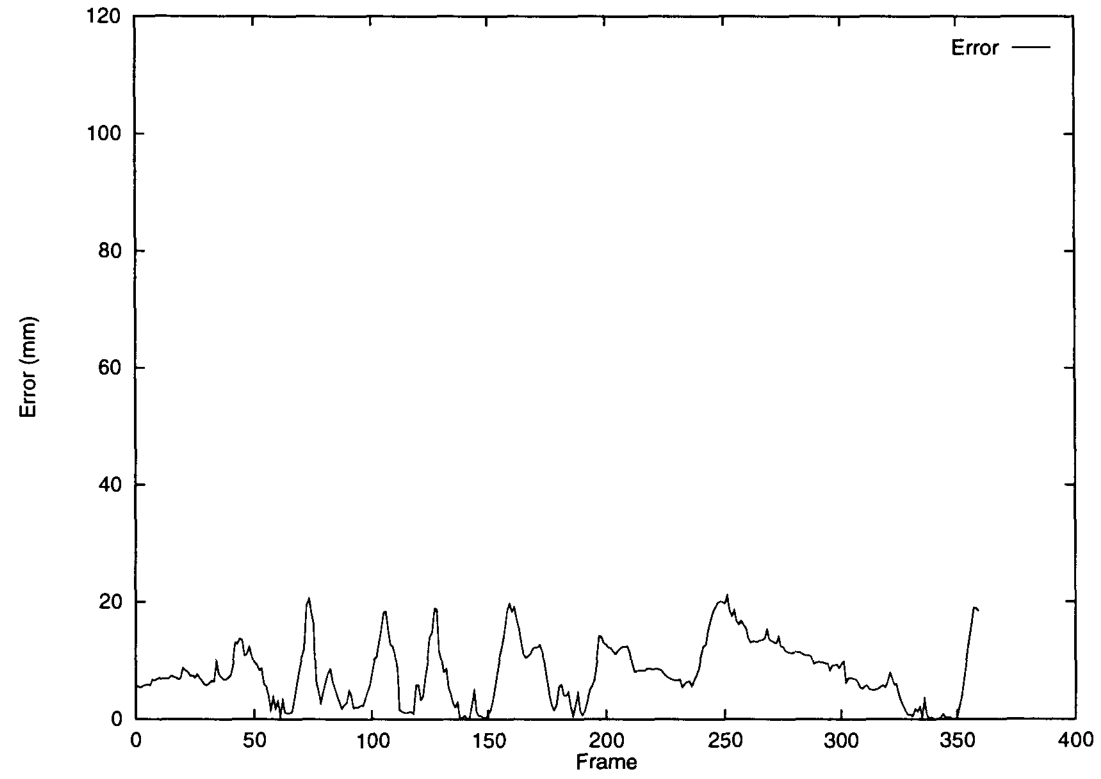

Figure 5.12: The error between measured tip location and computed tip location of the thumb

helps reducing errors caused by skin movement. However, this will severely restrict the movement of the thumb. Radius estimation error can be reduced by adjusting this radius parameter before the experiment until the computed movement of the thumb is similar to the actual movement.

Not very surprisingly, significant errors from frame 200 to 300 are observed from the plots of the thumb tip location. The average error of Figure 5.12 is 7.1mm.

5.4 Summary

An experiment was conducted to measure the performance of the proposed table based inverse kinematic algorithm. From the experimental results, we can see that the algorithm works extremely well for the index finger. Error between measured tip location and computed tip location only becomes large when the index finger bends backwards. Fortunately, this is rarely possible without external force exerted on the finger. The error for the thumb becomes significant when the value of the joint angle at the IP joint become large. This error can be reduced by eliminating skin movement effects on IRED position and careful estimates of the IP joint radius.

Chapter 6

Conclusions

We have examined the solutions to the inverse kinematic problem applied to the determination of hand posture in real-time. As an alternative to various algebraic and numerical methods published in robotics and computer graphics journals, a simple but very fast lookup table based inverse kinematic algorithm has been developed for the computation of the joint angles of the index finger and the thumb from live 3D positional data. This algorithm has been integrated into a complete data collection system which is able to calibrate, collect and display hand postures of test subjects in experiments.

A similar table based inverse kinematic algorithm was proposed by Amaya et al. [1]. However, the algorithm presented in this thesis addresses some of its weaknesses. The major contributions of this thesis include:

- developed a more accurate model of the finger using circular joints;
- developed a fast table-based algorithm which can be used effectively in real-time;
- integrated the polygonal representation of the outer surface of the hand into the display routines; and
- developed a data collection system which integrates data collection, hand posture estimation and display of the hand.

6.1 Summary

When doing this research, it is apparent that no one single inverse kinematic algorithm is uniformly superior to others and is suitable for all types of applications. For instance, computer animations systems requires their inverse kinematic algorithms to be able to produce "useful" intermediate solutions to refresh the screen in order to maintain the level of interactivity. On the other hand, trajectory control systems of robots require their inverse kinematic algorithms to produce accurate joint angle solutions to maintain the trajectory of the end-effector of the robot. So, the ability to produce intermediate solution is of little interest to these systems. By contrast, the inverse kinematic algorithms used in a real-time environment such as the one described in this thesis, is required to process incoming data (possibly containing noise) and produce a solution as quickly as possible. The quality of solution, of course, greatly depends on the correctness of the incoming data as well as any measured calibration parameters.

In Chapter 4, we have presented an algorithm which estimates a human hand posture from a limited number of IREDs. The key to this method is lookup tables built during a calibration step which measure the characteristics of different subjects' hands. With this information, the run-time system determines the joint angles of both the index finger and the thumb. The method presented provides the computational speed required for our real-time application and reasonable approximation of the hand postures of the subject. The solution returned by this algorithm is a compromise between accurate end-effector position and "natural" looking joint angles since it neither requires the end-effector to be accurately place at a particular point with no regard to the value of the joint angles nor it tries to set the joint angles in a way that the position of the end-effector is completely ignored. Instead, the algorithm tries to find a set of reasonably natural joint angles such that the error between the distance of the measured end-effector position and computed end-effector position is minimized. In Chapter 5, the method has been confirmed to produce adequate results.

6.2 Future Work

Since this thesis is only a part of the efforts in trying to construct a Virtual Hand Laboratory, there is still considerable work left to be done. First, the inverse kinematic algorithm presented in Chapter 4 is planned to be integrated into the head tracking system developed by Valerie Summers. This will provide a stereo view of the virtual hand for the subject while the experiment is in progress. Also, the calibration of the hand and the head tracking devices can be done together as one step. This will not only save time, but will also provide additional viewing information such as location of the viewer and so on for the display subroutine.

Since a neural network generalizes the notion of a lookup table, we would like to investigate ways to define a neural network to solve the inverse kinematics problem. Compared to a lookup table, the neural network has the added advantage that it can "learn" from past results. Since a neural network is essentially a lookup table, the processing speed would be comparable to the table based method described in Chapter 4. Further research is needed to define the structure of the neural network and its training method.

In order for the virtual hand to be truly useful, the occlusion problem would have to be solved. One way to solve this problem would be to place more than one Optotrak camera around the subject. If an IRED is occluded from the view of one camera, the other cameras will still be able to sense its position. Because of the high cost of the needed equipment, it is very unlikely that this method can be implemented in the near future.

In addition, a hierarchical spline representation of the hand can be constructed to allow for different resolutions of display of the hand. Using this property, experiments can be done to investigate the appropriate level of realism of the graphic hand needed to carry out a typical kinesiology experiment.

Furthermore, we plan to investigate ways to construct lookup tables having a uniform distribution of entries. Doing so, in theory, will speed up the time required to search the lookup table and resolve collisions present in each entry of the table.

Appendix: A

Notations Used in Amaya's Paper

Three coordinate frames are employed in Amaya's algorithm: the Optotrak "global" frame, the "hand" frame at the MCP joint created by three IREDs on the back of the hand, and the "fingertip" frame on the tip of the finger created by three IREDs on the fingertip.
The matrix $[B]$ is the basis matrix of a coordinate system where the ith basis vector is placed at the ith column of $[B]$. The vector $\boldsymbol{O}$ is the position vector of the origin of a coordinate system. The superscripts and subscripts on both $[B]$ and $\boldsymbol{O}$ read:

$$A^{source-frame}_{destination-frame}$$

The source frame is the frame of reference in which the input vector is currently expressed in which the destination frame is the frame of reference in which the output vector going to be expressed in. The positions of the IREDs are represented by the vector $\boldsymbol{d}$. The notation used is:

$$\boldsymbol{d}^{IRED\ group}_{coordinate-frame, IRED\ number}$$

The subscripts below other vectors represent the coordinate frame of that vector.

Appendix: B

Jacobian Matrix and Its Pseudoinverse

The Jacobian is a matrix of first derivatives. To derive it, we can differentiate the elements of equation (3.1) to get

$$\begin{aligned} dx_1 &= \frac{\partial f_1}{\partial \theta_1} d\theta_1 + \frac{\partial f_1}{\partial \theta_2} d\theta_2 + \ldots + \frac{\partial f_1}{\partial \theta_n} d\theta_n \\ dx_2 &= \frac{\partial f_2}{\partial \theta_1} d\theta_1 + \frac{\partial f_2}{\partial \theta_2} d\theta_2 + \ldots + \frac{\partial f_2}{\partial \theta_n} d\theta_n \\ &\vdots \\ dx_m &= \frac{\partial f_m}{\partial \theta_1} d\theta_1 + \frac{\partial f_m}{\partial \theta_2} d\theta_2 + \ldots + \frac{\partial f_m}{\partial \theta_n} d\theta_n \end{aligned} \tag{6.1}$$

Dividing by the differential time element dt and writing in matrix notation, we have

$$\dot{\boldsymbol{x}} = J(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \tag{6.2}$$

and clearly J is

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} & \cdots & \frac{\partial f_1}{\partial \theta_n} \\ \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_2} & \cdots & \frac{\partial f_2}{\partial \theta_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_m}{\partial \theta_1} & \frac{\partial f_m}{\partial \theta_2} & \cdots & \frac{\partial f_m}{\partial \theta_n} \end{bmatrix} \tag{6.3}$$

If J is not a square matrix or does not have a full rank, J^{-1} does not exist and there is no solution for $\dot{\boldsymbol{\theta}}$. In order to find a useful solution for $\dot{\boldsymbol{\theta}}$, the generalized inverse of J is usually used. A commonly used generalized inverse J^+ is the Moore-Penrose pseudoinverse[6]. The pseudoinverse J^+ is defined as

$$J^+ = J^T(JJ^T)^{-1} \tag{6.4}$$

for the under determined case and

$$J^+ = (J^TJ)^{-1}J^T \tag{6.5}$$

for the over determined case.

Appendix: C

C.1 Algorithm to build the lookup table

This section presents an algorithm that builds the lookup table for the index finger. The lookup table is built by evaluating the forward kinematics equations for $\frac{1}{p}$ degree increments for each of the PIP and DIP joint angles, where p is the precision of the joint angles. The index used for the lookup table is the distance between the tip of the finger and the MCP joint. Since the value of θ_1 (the MCP joint angle) does not affect the value of the index, it is set to 0.

ALGORITHM: *BuildIndexTable*

Input: *The segment length l_i ordered from the MCP joint to the tip. The joint radius r_i of each joint. The joint limits for the DIP and PIP joints. The precision, p, of the joint angle.*

Output: *A joint angle lookup table having $\|\boldsymbol{t}\|$ as its index. Each entry in the table consists of a counter and an array of joint angles.*

BEGIN

```
    $\theta_1 := 0$
    FOR $t_2$ := (pipLowerLimit/p) TO ($pipUpperLimit/p$) BY 1 DO
        FOR $t_3$ := (dipLowerLimit/p) TO ($dipUpperLimit/p$) BY 1 DO
            $\theta_2 := t_2 p$
            $\theta_3 := t_3 p$
            $d_0 := 2r_1 \sin\theta_2/2$
            $d_1 := 2r_2 \sin\theta_3/2$
            $\phi_0 := \theta_1$
            $\phi_1 := \phi_0 + \theta_2/2$
            $\phi_2 := \phi_1 + \theta_2/2$
            $\phi_3 := \phi_2 + \theta_3/2$
            $\phi_4 := \phi_3 + \theta_3/2$
            $x := l_1 \cos\phi_1 + d_0 \cos\phi_2 + l_2 \cos\phi_3 + d_1 \cos\phi_4 + l_3 \cos\phi_5$
            $y := l_1 \sin\phi_1 + d_0 \sin\phi_2 + l_2 \sin\phi_3 + d_1 \sin\phi_4 + l_3 \sin\phi_5$
            $index := \frac{\sqrt{x^2+y^2}}{p}$
            Table[$index$].angleList[$count$] := $\{\theta_2, \theta_3\}$
            Table[$index$].$count$ := Table[$index$].$count$ + 1
        END /* FOR */
    END /* FOR */
    RETURN Table
```

END

C.2 Algorithm to search the lookup table

This section presents an algorithm that searches the lookup table at run-time to obtain a subset of the needed joint angles. The algorithm first computes the index value (the distance between fingertip and the MCP joint). Then, it looks for the corresponding entry in the lookup table. If this entry is empty, the algorithm searches both forward and backwards to find the closest non-empty entry. After this entry is found, the algorithm then determines whether there are more than one set of joint angles stored in this entry. If not, the only set of joint angles stored in this entry is returned. Otherwise, the algorithm finds the set of joint angles which has the minimum change from the set of joint angles of the previous frame. This set of joint angles is then returned. This algorithm also checks to make sure that the angle β is less than or equal to the angle α (α and β are discussed in Chapter 4).

ALGORITHM: *GetIndexJointAngles*
Input: *The lookup table. The IREDs' positional vectors* $\boldsymbol{d}_i$ *in the global frame as in Figure 4.4. The index finger joint angles* $\{\theta_2^{index'}, \theta_3^{index'}\}$ *of the previous frame. The precision, p, of the joint angles.*
Output: *The joint angles of index finger for the current frame.*
BEGIN

```
    $||\boldsymbol{t}|| := ||\boldsymbol{d}_1 - \boldsymbol{d}_3||$
    $\boldsymbol{x}_g := \frac{\boldsymbol{d}_3 - \boldsymbol{d}_4}{||\boldsymbol{d}_3 - \boldsymbol{d}_4||}$
    $\boldsymbol{y}_g := \frac{(\boldsymbol{d}_5 - \boldsymbol{d}_4) \times \boldsymbol{x}_g}{||(\boldsymbol{d}_5 - \boldsymbol{d}_4) \times \boldsymbol{x}_g||}$
    $\boldsymbol{z}_g := \frac{\boldsymbol{x}_g \times \boldsymbol{y}_g}{||\boldsymbol{x}_g \times \boldsymbol{y}_g||}$
    $\alpha := \left(\frac{\pi}{2} - \cos^{-1}\left(\frac{\boldsymbol{t} \cdot \boldsymbol{y}_g}{||\boldsymbol{t}||}\right)\right)$
    $less$ := FALSE
    $minErr := maxFloat$
    $position := 0$
    $index := ||\boldsymbol{t}||/p$
    IF Table[$index$].$count$ = 0 /* if this entry is empty */
        $found$ := FALSE
        $forward := index$
        $backward := index$
        WHILE (NOT $found$) /* find first non-empty entry in either direction
        */
            $forward := foward + 1$
            $backward := backward - 1$
            $found$ := ((Table[$forward$].count>0) OR (Table[$backward$].count>0))
        END /* WHILE */
        IF (Table[$forward$].$count$ > 0) THEN /* get search direction */
            $index := forward$
        ELSE
            $index := backward$
        END /* IF */
```

```
END /* IF */
WHILE (NOT $less$) /* if angle $\beta$ is not less than $\alpha$ */
    FOR $i$ := 0 TO Table[$index$].$count$ − 1 BY 1 DO
        $\{\theta_2, \theta_3\}$ := Table[$index$].angleList[$i$]
        $t'_x = l_1 + l_2\cos(\theta_2) + l_3\cos(\theta_2 + \theta_3)$
        $t'_y = l_2\sin(\theta_2) + l_3\cos(\theta_2 + \theta_3)$
        $\beta := \cos^{-1}\left(\frac{t'_x}{\sqrt{(t'_x)^2+(t'_y)^2}}\right)$
        $err := \sqrt{((\alpha - \beta) - \theta'_1)^2 + (\theta_2 - \theta_2^{index'})^2 + (\theta_3^{index} - \theta_3^{index'})^2}$
        IF ($\beta <= \alpha$)
            IF (NOT $less$)
                $minErr = err$
                $position = i$
                $less$ = TRUE
            ELSE
                IF ($err < minErr$) THEN
                    $minErr = err$
                    $position = i$
            END /* IF */
            END /* IF */
        END /* IF */
    END /* FOR */
    IF (NOT $less$)
        $position = 0$
        $index := index + 1$
    END /* IF */
END /* WHILE */
$\{\theta_2^{index}, \theta_3^{index}\}$ := Table[$index$].angleList[$position$]
$\boldsymbol{p} := \boldsymbol{t} - \boldsymbol{y}_g(\boldsymbol{t} \cdot \boldsymbol{y}_g)$
$t'_x = l_1 + l_2\cos(\theta_2) + l_3\cos(\theta_2 + \theta_3)$
$t'_y = l_2\sin(\theta_2) + l_3\cos(\theta_2 + \theta_3)$
```

```
    $\beta := \cos^{-1}\left(\frac{t'_x}{\sqrt{(t'_x)^2+(t'_y)^2}}\right)$
    $\theta_0^{index} := \cos^{-1}\frac{\boldsymbol{x}_g\cdot\boldsymbol{p}}{||\boldsymbol{p}||}$
    $\theta_1^{index} := \alpha - \beta$
    RETURN $\{\theta_0^{index}, \theta_1^{index}, \theta_2^{index}, \theta_3^{index}\}$
END
```

Bibliography

[1] Kenji Amaya, Tom Calvert, and Christine L. MacKenzie. Posture estimation of a human hand from a limited number of markers (unpublished). August 1995.

[2] K. N. An, E. Y. Chao, W. P. Cooney III, and R. L. Linscheid. Normative model of human hand for biomechanical analysis. *Journal of Biomechanics*, 12:775–788, 1979.

[3] K. N. An, Y. Ueba, E. Y. Chao, W. P. Cooney, and R. L. Linscheid. Tendon excursion and moment arm of index finger muscles. *Journal of Biomechanics*, 16(6):419–425, 1983.

[4] Sumeet Bawa. Interactive creation of animations: An optimization approach to real-time inverse kinematics. M.Sc. thesis, Simon Fraser University, April 1995.

[5] A. Ben-Israel and T. N. E. Greville. *Generalized Inverses: Theory and Applications.* Wiley, New York, 1974.

[6] T. L. Boullion and P. L. Odell. *Generalized Inverse Matrices.* Wiley Interscience, New York, 1971.

[7] William L. Buford and David E. Thompson. A system for three-dimensional interactive simulation of hand biomechanics. *IEEE Transactions on Biomedical Engineering*, 34(6):444–453, 1987.

[8] E.Y. Chao, J. D. Opgrande, and F. E. Axmear. Three dimensional force analysis of finger joints in selected isometric hand functions. *Journal of Biomechanics*, 9:387–396, 1976.

[9] John J. Craig. *Introduction to robotics: mechanics and control.* Addison-Wesley, New York, 1989.

[10] J. Denavit. *Description and Displacement Analysis of Mechanisms Based on 2x2 Dual Matrices.* Ph.D. thesis, Northwestern University, 1956.

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

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

[13] G. Forsythe and C. B. Moler. *Computer Solution of Linear Algebraic Systems.* Prentice-Hall, Englewood Cliffs, N.J., 1967.

[14] K. S. Fu, R. C. Gonzales, and C. S. G. Lee. *Robotics: Control, Sensing, Vision, and Intelligence.* McGraw-Hill, Inc., New York, 1987.

[15] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization.* Academic Press, New York, 1981.

[16] Donald Goldfarb. Extension of Davidson's variable metric method to maximization under linear inequality and equality constraints. *SIAM J. Appl. Math.*, 17:739–764, 1969.

[17] B. Gorla and M. Renaud. *Robots Manipulateurs.* Cepadues-Editions, Toulouse, 1984.

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

[19] Thea Iberall, Carme Torras, and Christie MacKenzie. Parameterizing prehension: A mathematical model of opposition space. pages 2–12, Proceedings of COGNITIVA 90.

[20] D. Kohli and A. H. Soni. Kinematic analysis of spatial mechanisms via successive screw displacements. *Journal of Energy for Industry*, 2:739–747, 1975.

[21] J. M. F. Landsmeer. Anatomical and functional investigations on the articulation of the human fingers. *Acta anatomica*, 25:1–69, 1955.

[22] J. M. F. Landsmeer. A report on the co-ordination of the interphalangeal joints of the human finger and its disturbances. *Acta morphologica neerlando-scandinavica*, pages 59–84, 1958.

[23] J. M. F. Landsmeer. Studies in the anatomy of articulation. *Acta morphologica neerlando-scandinavica*, 3:287–303, 1961.

[24] J. M. F. Landsmeer. The coordination of finger joint motions. *The journal of bone and joint surgery*, 45:1654–1662, 1963.

[25] J. Lasseter. Principles of traditional animation applied to 3d computer animation. *Computer Graphics*, 21(4):35–44, Proceedings of SIGGRAPH87.

[26] C. S. G. Lee and M. Ziegler. A geometric approach in solving the inverse kinematics of puma robots. *IEEE Trans. Aerospace and Electronic Systems*, 20(6):695–706, 1984.

[27] A. Liegeois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Trans. System, Man, Cybernetics*, 7(12), 1977.

[28] Girard. M. and Maciejewski. A. A. Computational modeling for computer generation of legged figures. *Computer Graphics*, 19(3):263–270, 1985.

[29] Anthony A. Maciejewski. Dealing with the ill-conditioned equations of motion for articulated figures. *IEEE Computer Graphics and Applications*, pages 63–71, 1990.

[30] Anthony A. Maciejewski and Charles A. Klein. The singular value decomposition: Computation and applications to robotics. *The International Journal of Robotics Research*, 8(6):63–79, 1989.

[31] C. L. Mackenzie and T. Iberall. *The grasping hand.* Elsevier Science, Amsterdam, 1994.

[32] Axel Mulder. Human movement tracking technology. Technical report, School of Kinesiology, Simon Fraser University, B.C., July 1994.

[33] R. P. Paul. *Robot Manipulators: Mathematics, Programming, and Control.* MIT Press, Cambridge, MA, 1981.

[34] R. P. Paul, B. E. Shimano, and G. Mayer. Kinematic control equations for simple manipulators. *IEEE Trans. Systems, Man, Cybern.*, 11(6):449–455, 1981.

[35] Cary B. Phillips, Jianmin Zhan, and Norman I. Badler. Interactive real-time articulated figure manipulation using multiple kinematic constraints. *Computer Graphics*, pages 245–250, 1990.

[36] Hans Rijpkema and Michael Girard. Computer animation of knowledge-based human grasping. *Computer Graphics*, 25(4):339–348, 1991.

[37] J. B. Rosen. The gradient projection method for nonlinear programming. *SIAM J. Appl. Math.*, 8:181–217, 1960.

[38] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Math. Computation*, 24:647–664, 1970.

[39] D. J. Sturman. *Whole-hand Input.* Ph.D. thesis, MIT, 1989.

[40] David J. Sturman and David Zeltzer. A survey of glove-based input. *IEEE Computer Graphics and Applications*, 14(1):30–39, 1994.

[41] S. Augustine Su and Richard Furuta. A logical hand device in virtual environments. Technical report, Department of Computer Science, University of Maryland, College Park, Maryland 20742, April 1993.

[42] C. Taylor and R. Schwartz. The anatomy and mechanics of the human hand. *Artificial Limbs*, (2):22–35, 1955.

[43] Chor Quan Teo. A hybrid procedural/knowledge-based approach to the animation of human hand grasping. M.Sc. thesis, Simon Fraser University, February 1994.

[44] David E. Thompson. Biomechanics of the hand. *Perspectives in computing*, 11(3):12–19, 1981.

[45] Chris Welman. Inverse kinematics and geometric constraints for articulated figure manipulation. M.Sc. thesis, Simon Fraser University, April 1993.

[46] A. T. Yang and R. Freudenstein. Application of dual number quaternian algebra to the analysis of spatial mechanisms. *Trans. ASME, J. Appli. Mech.*, 31:152–157, 1964.

[47] 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, 1989.

[48] T. G. Zimmerman. A hand gesture interface device. In *Human Factors in Computing Systems and Graphics Interfaces*, pages 189–192, April 1987.