Institute for Research in Cognitive Science

Kinematic Control of Human Postures for Task Simulation

Xinmin Zhao

University of Pennsylvania
3401 Walnut Street, Suite 400A
Philadelphia, PA 19104-6228

December 1996

Site of the NSF Science and Technology Center for Research in Cognitive Science

IRCS Report 96--32

KINEMATIC CONTROL OF HUMAN POSTURES FOR TASK SIMULATION

XINMIN ZHAO

A DISSERTATION

in

COMPUTER AND INFORMATION SCIENCE

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy.

1996

Dr. Norman I. Badler

Supervisor of Dissertation

Dr. Mark Steedman

Graduate Group Chairperson

Acknowledgment

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Norman I. Badler, for his invaluable guidance and advice, for his confidence in me from the beginning to the end, for all the freedom to pursue the research topics that interest me the most, and for the excellent research environment that he fosters at the Center for Human Modeling and Simulation.

I wish to thank all my committee members, Dr. Ahmed Shabana, Dr. Richard Paul, Dr. Bonnie Lynn Webber, and Dr. Dimitris Metaxas, for their insightful comments which have greatly improved this thesis. I am especially thankful to Dr. Metaxas for many fruitful discussions, and to Dr. Webber for her guidance and support.

I am sincerely thankful to many people at the center for their help. In particular, I am grateful to Diane Chi, Harold Sun, Brett Achorn, Jong Park and Deepak Tolani for reading my thesis draft and for their comments which has immensely improved this thesis. I would like to thank Deepak Tolani for many discussions which inspired this work. I would like to thank many others for their help and friendship including: Tripp Becket, Pei-Hwa Ho, Hanns-Oskar Porr, Min-Zhi Shao, Bond-Jay Ting and Jianmin Zhao, as well as everyone in the *probability group* chaired by Barry Reich. I want to thank Karen Carter, Ken Noble and Mike Felker for all their help. I also want to express my gratitude to many people outside the University, including Soon Myoung Chung, Kevin Kirby, Prabhaker Mateti, Chen Ding, Kan Zhao, Leon Lidofsky, Dong Chen, and Qi-Mei Sun.

This research is supported by DMSO DAAH04-94-G-0402 through the US Air Force HRGA, WPAFB, and a fellowship from JustSystem of Japan. I am grateful to JustSystem for their generous support which allowed me to concentrate on my thesis research.

Finally, I want to thank my parents for raising me in a loving and caring family. I am grateful to my grandmother, my aunt and my two cousins. This thesis is dedicated to my wife, Min, for her love, support and encouragement, and to our daughter Jessica, for her beautiful smiles and countless kisses.

ABSTRACT

KINEMATIC CONTROL OF HUMAN POSTURES FOR TASK SIMULATION

XINMIN ZHAO

NORMAN I. BADLER

Kinematic control of human postures for task simulation is important in human factor analysis, simulation and training. It is a challenge to control the postures of a synthesized human figure in real-time on today's graphics workstations because the human body is highly articulated. In addition, we need to consider many spatial restrictions imposed on the human body while performing a task.

In this study, we simplify the human posture control problem by decoupling the degrees of freedom (dof) in the human body. Based on several decoupling schemes, we develop an analytical human posture control algorithm. This analytical algorithm has a number of advantages over existing methods. It eliminates the local minima problem, it is efficient enough to control whole human body postures in real-time, and it provides more effective and convenient control over redundant degrees of freedom. The limitation of this algorithm is that it cannot handle over-constrained problems or general constraint functions. To overcome this limitation, we transform the human posture control problem from a 40 variable joint space to a 4 to 9 redundancy parameter space. We then apply nonlinear optimization techniques on the transformed problem. Because the search space is reduced, this new numerical algorithm is more likely to find a solution than existing methods which apply optimization techniques directly in the joint space.

The contributions of this thesis include a decoupling approach for simplifying the human posture control problem, an analytical human posture control algorithm based on this decoupling approach, and a numerical human posture control algorithm in redundancy parameter space. These two new algorithms are more efficient and effective than existing methods, and they also give the user control to select the desired solution. Moreover, the analytical algorithm can control postures of a few 92 dof human figures at 30 Hz.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Problem statement 2

1.3 Related work 4

1.3.1 Pseudoinverse and task priority 4

1.3.2 Damped least squares and singular robust inverse 5

1.3.3 Configuration control 6

1.3.4 Inverse kinematics 7

1.3.5 Human arm inverse kinematics and human motion planning . . . 7

1.4 Thesis outline 8

2 Human Posture Control: Problem Analysis and Our Approach **10**

2.1 Articulated human figure model 10

2.2 Problem formulation and analysis 11

2.2.1 Number of degrees of freedom 12

2.2.2 Local minima problem 13

2.2.3 Redundancy resolution and control 13

2.3 Our decoupling approach 14

3 Analytical Human Posture Control **17**

3.1 Human posture control and the decoupling approach 17

3.1.1 Decoupling upper and lower body movements 18

3.1.2 Decoupling torso bending and arm movements 20

3.1.3 Decoupling position and orientation for arm posture computation . 20

3.2 Human body redundancy parameterization 20

3.3 Analytical human posture control algorithm 22

3.4 Default posture computation 23

3.4.1 Default arm posture computation 24

3.4.1.1 Determine the elbow angle 25

3.4.1.2 Control the elbow position 25

3.4.1.3 Determine the shoulder angles 26

3.4.1.4 Determine the wrist angles 28

3.4.2 Default torso posture computation 30

3.4.2.1 Reachable space approximation 31

3.4.2.2 New torso posture computation 34

3.4.3 Default base orientation computation 35

3.4.4 Default base position computation 35

3.4.4.1 Base height and torso bending 35

3.4.4.2 Base floor position 36

3.4.5 Default lower body posture computation 37

3.5 Human body redundancy control 39

3.5.1 Elbow position control 40

3.5.1.1 Forward elbow position control 40

3.5.1.2 Inverse elbow position control 42

3.5.1.3 Hand position control 43
3.5.2 Arm point position in terms of elbow height control 44
3.5.3 Arm posture in terms of torso bending 46
3.5.3.1 Point on the lower arm 46
3.5.3.2 Point on the upper arm 47
3.5.4 Shoulder position control using the waist group joint 49
3.5.4.1 Our approach 51
3.5.4.2 Approximating $u(\theta, \beta)$ and $v(\theta, \beta)$ without torso twist . 51
3.5.4.3 Approximating $u(\theta, \beta)$ and $v(\theta, \beta)$ with torso twist . . . 56
3.5.4.4 Results 59
3.5.4.5 Reachable space projection 59
3.6 Redundancy space posture representation 60
3.6.1 Converting postures between joint space and redundancy space . . 60
3.6.2 Advantages of redundancy space posture representation 61

4 Numerical Human Posture Control **63**
4.1 Nonlinear optimization and Newton's method 65
4.2 Human posture control using nonlinear optimization 66
4.2.1 Human posture control in joint space 66
4.2.2 Human posture control in redundancy parameter space 67
4.2.2.1 Arm point position 68
4.2.2.2 Shoulder position and orientation 69
4.2.2.3 Human posture control in redundancy parameter space . 69
4.3 Solving the human posture control problem using an NLP solver 70
4.3.1 The *LBFGSB* solver 71
4.3.2 Objective function and its gradient 72

4.3.3 End-effector constraint 74
4.3.4 Joint limit constraints 74
4.3.4.1 Elbow joint angle 75
4.3.4.2 Shoulder joint angles 75
4.3.4.3 Wrist joint angles 76

5 Experiments **77**
5.1 Experiments with the analytical human posture control algorithm 77
5.1.1 Default posture computation 77
5.1.2 Adjust posture in redundancy space 79
5.1.3 Performance 81
5.2 Experiments with the numerical human posture control algorithm 83

6 Conclusions **85**
6.1 Contributions 85
6.2 Future work 87
6.2.1 Shoulder joint complex 87
6.2.2 Foot placement heuristics 88
6.2.3 Automatic decoupling 88
6.2.4 Posture interpolation in redundancy space 88
6.2.5 Natural-looking human movements 89
6.3 Conclusions 90

References **92**

List of Tables

3.1 Average and maximum errors of the analytical torso inverse kinematics scheme 59

5.1 Performance data of the analytical human posture control algorithm . . . 81

List of Figures

2.1 Articulated model of the simulated human figure *Jack* 11

2.2 Human arm structure and the elbow joint angle computation 15

3.1 Reachable space of the hand and its projections (torso is straight) 31

3.2 Reachable space of the hand and its projections (torso is bending) 32

3.3 Half sphere approximations of the left and right hand reachable spaces . . 33

3.4 Compute the orientation of the base 35

3.5 Compute the base position on the floor (XZ plane) 36

3.6 Foot position computation 38

3.7 Elbow position control 41

3.8 Inverse elbow position control 43

3.9 Inverse arm point position control 45

3.10 Control the position of a point on the lower arm 47

3.11 Control the position of a point on the upper arm 48

3.12 The spine structure of the *Jack* human model 49

3.13 Function $u(\theta)$ 52

3.14 Function $u(\theta)$ with $v = 0$ and $v = 1$. 53

3.15 Function $\theta_0(\beta) = \theta(u = -1, \beta)$ 54

3.16 End-effector error (in cm) as a function of the torso twist parameter w . . 57

5.1 Step 1 of the analytical algorithm: default posture computation 78
5.2 Redundancy control 1: adjusting elbow height control of the left arm . . . 79
5.3 Redundancy control 2: adjusting torso flexion, side bending and twist . . 80
5.4 Redundancy control 3: adjusting base position and orientation 80
5.5 Minimizing the distance between the left upper arm and the small cube . . 84
6.1 Architecture of a human task simulation system 91

Chapter 1

Introduction

1.1 Motivation

For design evaluations, virtual prototyping offers a cost-effective alternative to building real prototypes of complex systems such as space stations, airplanes, ships [18] [15]. With the progress of CAGD, the building of virtual prototypes (in terms of geometry) is well understood today. On the other hand, much more work is needed in design evaluation using virtual prototypes.

One important aspect of a complex system design is the geometry of the human-machine interface. In evaluating a human-machine interface, we want to make sure that all prospective users are able to perform the required tasks in the designed system. An intuitive way of doing this is to simulate those tasks and see if they are feasible. This can be done by putting a simulated human in the virtual prototype and controlling it to perform the tasks. By simply observing and analyzing the motions of the simulated human, we can assess the feasibility of all tasks, and thus the feasibility of the design. We call this process virtual prototype evaluation. A key issue in this process is the *control* of the simulated human for performing a given task. In addition to virtual prototype evaluations, the human motion control problem is also important to many other areas such as simulation

and training, computer animation, and virtual environment.

1.2 Problem statement

In this study, we are interested in the human motion control problem for task performance in a virtual environment. More specifically, given the position and orientation of the end-effector (usually the hand) or its trajectory, an environment and a simulated human model, we need to compute a posture or a motion sequence of the human body that places the end-effector at the desired location.

In solving the human motion control problem, we need to consider the geometrical restrictions on the human body imposed by the environment, including:

1. *Collision avoidance constraints*: During the movements, there should be no intersections of rigid bodies.

2. *Task-related constraints*: These depend on the particular task. For example, if the task is to put a cup of coffee on the table, holding the coffee cup upright to avoid spilling would be a task-related constraint.

3. *Structural constraints*: These are the constraints inherent to the human articulated structure. For example, each joint has joint limits.

In order to avoid collisions and handle task-related constraints, we must be able to control the position and orientation of not only the end-effector but other parts of the body as well. That is, we must be able to control the redundancy in the postures that place the end-effector at the given location.

Another consideration is the efficiency of the motion control system. The human body has many degrees of freedom, and many points on the body may be involved in collisions at the same time. Thus, the motion control algorithm must be very efficient in order to have real-time control over the human figure in a workstation environment.

For many tasks, the human body is a redundant system: there are an infinite number of ways to perform them. The user may prefer that the task be performed in a particular, “desired" way. This requires that the motion control algorithm provide a mechanism to express and compute the “desired" postures or motions.

A motion sequence consists of a series of coherent postures. In order to simplify the human motion control problem, in this thesis we do not consider the coherence in a posture sequence. As a result, we can solve the human motion control problem as a human posture control problem. Thus, we only discuss the human posture control problem in this thesis.

With these considerations, the goal of this thesis is to develop a human posture control algorithm for task simulations in a virtual environment that:

1. controls the hand to follow a given trajectory;

2. controls positions and orientations of other parts of the body to handle geometrical restrictions such as collision avoidance and task-related constraints;

3. provides a mechanism to let users express and synthesize “desired" human postures and motions; and

4. is efficient enough for real-time control of a human figure with 40 dof.

To satisfy the third requirement, we can provide either a direct mechanism for the user to manipulate the posture interactively in redundancy space, an indirect mechanism for the user to specify a scalar performance criterion minimization function, or both.

Human task performance is a complex process. There are many considerations that determine how a task is performed. Correspondingly, we can study human postures in task performance using many tools including kinematics, dynamics, biomechanics, human motor control theory, psychology, physiology, etc. In this study we are only interested in evaluating the geometry of the human machine interface. For this reason, we confine our study to kinematic human posture control for task performance.

1.3 Related work

There are a number of areas in computer animation and robotics which are related to our work. In this section we briefly review some of the posture and motion control techniques developed in those areas. In the next chapter we discuss the drawbacks of these techniques when they are applied to solving the human posture control problem. Here we only review kinematic motion and posture control techniques since they are directly related to this thesis work.

1.3.1 Pseudoinverse and task priority

In general we can decompose a task R into a number of subtasks r_i:

$$R = [r_0, r_1, ..., r_m]^T$$

Each subtask can be either end-effector tracking, collision avoidance, etc. These subtasks (also called tasks) can be represented by a set of nonlinear equations:

$$r_i - f_i(q) = 0 \; 1 \leq i \leq m$$

where m is the number of tasks, $q = [q_1, \; ..., \; q_n]^T$ is the vector of joint variables.

Let J_i be the Jacobian matrix, r_i' be the velocity vector of the i-task. Let q' be the joint velocity vector. We have the following equation relating joint space velocity and task space velocity:

$$R' = [r_1' \;\; r_2' \;\; ... \;\; r_m']^T = [f_1'q' \;\; f_2'q' \;\; ... \;\; f_m'q']^T = Jq' \tag{1.1}$$

where $J = [J_1 \; J_2 \; ... \; J_m]^T = [f_1' \; f_2' \; ... \; f_m']^T$. In general $m \neq n$, where n is the number of degrees of freedom (dof) in the system. So we cannot directly invert the Jacobian matrix J to compute q'. Instead, J^*, the pseudoinverse of J is used [26] [43]. That is:

$$q' = J^* R' \tag{1.2}$$

Moreover, the general solution of the linear system $R' = Jq'$ is given by:

$$q' = J^* R' + (I - J^* J)z$$

where I is the $n \times n$ identity matrix, and z is an arbitrary vector.

In equations 1.1 and 1.2, all tasks are treated equally. There are situations where some of the tasks are more important than others, and we would like to give higher priorities to those tasks. Methods based on priority tasks system are discussed in [26].

1.3.2 Damped least squares and singular robust inverse

In section 1.3.1, the pseudoinverse of the Jacobian matrix is used to transform 3D space velocity to joint velocity. This approach works well when the Jacobian matrix is well conditioned. However, if the Jacobian matrix is singular or nearly singular, the joint velocity computed from pseudoinverse of it will be excessively large. A method called damped least squares (also called singular robust inverse) is proposed by Nakamura and Hanafusa [27] and Wampler [41] to address this problem. Their work leads to the following formula:

$$q' = (J^T w_r J + w_q)^{-1} J^T w_r r' = \hat{J} r'$$

$$\hat{J} = (J^T J + kI)^{-1} J^T = J^T (J J^T + kI)^{-1}$$

Note that even if J is singular, $J^T J + kI$ is positive definite and its inverse exists. Thus, $\hat{J}$ is a singular robust inverse of the Jacobian matrix J. k is the damping factor, which balances the trade off between the joint velocity norm $||q'||$ and the task error norm $||r' - Jq'||$. Larger k sacrifices task accuracy to avoid excessive joint velocity, while smaller k improves task accuracy at the cost of possibly large joint velocity.

1.3.3 Configuration control

The damped least squares method was formulated to compute inverse kinematics robustly in the presence of a singular Jacobian. This approach has also been extended for motion control of redundant robots using the augmented Jacobian by Seraji, et al. [34] [8] [36] [13] [37] [35]. Their approach is summarized here.

Let $R = [r_1, r_2, ...]^T$ be the tasks to be performed.

$$R' = [r'_1, r'_2, ...]^T = [J_1, J_2, ...]^T q' = Jq'$$

where J is the augmented Jacobian of the system (i.e., we augment the original Jacobian matrix J_1 with $J_2, J_3, ...$). Because the singularities of the augmented Jacobian depend on the tasks being performed, in general they are complicated and unavoidable. Thus, the handling of singularities is necessary during the simulation. In this situation, the damped least squares method proves to be valuable, and it is used in the configuration control. Instead of using a strict task priority scheme, a weight matrix is used to control the relative priorities of various tasks.

Let r_d be the desired task trajectory and r the actual task trajectory. We can use the damped least squares solution to control the end-effector to follow the desired trajectory r_d:

$$q' = \hat{J}(r'_d + Ge)$$

where r'_d is the desired velocity, Ge is the feedback term, G is the gain matrix, and $e = r_d - r$ is the error term. This method is called configuration control since it works in configuration space and a feedback term is added to the formulation.

In configuration control, when inequality constraints are violated they are treated as equality constraints. Multi-objective optimization is achieved through the minimization of the sum of weighted error squares.

1.3.4 Inverse kinematics

If we ignore the constraints, the human posture control problem is essentially an inverse kinematics problem. There has been much work in this area [6] [10] [14] [24] [31] [33]. In this proposal we review an approach by Jianmin Zhao and Badler [44] which is very successful in controlling simulated human movements [1].

The inverse kinematics problem can be stated as an optimization problem:

$$min_q \ ||e(q) - e_{goal}||$$

where $e(q)$ is the end-effector coordinates in terms of joint angles q, e_{goal} is the goal coordinates of the end-effector. Instead of using first order methods such as gradient descent, a second order method was used by Zhao and Badler [44] to locate the minimizer. In general, it is much more efficient and stable than gradient-based methods. Moreover, the algorithm also handles joint limits in the form of linear constraints [1], which is essential in simulated human motion and posture control [1].

1.3.5 Human arm inverse kinematics and human motion planning

The inverse kinematics algorithms discussed in section 1.3.4 work for any n-dof structure. An inverse kinematics algorithm designed to be used for human arm structure is proposed by Kondo [20]. It decouples the problem into finding an upper and lower arm posture to match the hand position first and then computing the wrist joint angles to match the hand orientation. The computation of hand posture follows a model based on results from neurophysiology [39] [38]. The extra dof in the upper/lower arm (4 dofs) for positioning the hand (3 dofs) is used to select feasible (within joint limits) joint angles of the shoulder and elbow joints. Since the neurophysiology model is only an approximate model, final adjustment of the joint angles using a Jacobian-based approach is needed to zero out the

[1] Constraints are linear if they can be represented by linear functions. Otherwise, they are nonlinear.

end-effector position and orientation errors.

Koga, et al. use the above inverse kinematics algorithm and a randomized motion planning algorithm [2] [22] to plan human manipulation motions [19]. Ching and Badler [7] use a motion planning approach successfully to plan human task motions in configuration space. For surveys on motion planning, the reader is referred to the work by Latombe [22], Hwang and Ahuja [17]. One drawback of motion planning approaches is that they are usually too slow to be useful in an interactive system. For the human task simulation problem, we cannot use the motion planning approach.

Lee et. al. simulate human lifting tasks by considering the figure geometry, the external load, the goal position and the human strength model [23]. They also model different human behaviors (e.g., pull back) during the lifting task. The user can also control the level of comfort of the agent.

Tolani and Badler developed an analytical algorithm to the human arm inverse kinematics problem [40]. It controls the clavicle, the shoulder, the elbow and the wrist joints. The redundancy in the system is controlled by the elbow twist parameter. Given the hand position and orientation, and the elbow twist parameter, the algorithm can compute the joint angles analytically to put the hand in the desired location.

Zhao and Badler developed a collision avoidance algorithm for human movement simulation [45]. It surrounds obstacles with potential fields. Once the human body (or parts of it) enters a potential field, it will be pushed away by the repulsive force generated by the potential field. The action of repulsive forces is simulated using a nonlinear optimization solver.

1.4 Thesis outline

In the following chapter, we analyze the human posture control problem in detail. Then we discuss the deficiencies of existing methods when they are applied to the human posture control problem. Based on these discussions, we propose our approach to the problem.

The main idea of our approach is to reduce the complexity of the human posture control problem by decoupling the dofs in the human body.

In Chapter 3, we introduce a decoupling approach for human posture control. Using this approach we develop several decoupling schemes by considering the human body articulated structure. Based on these decoupling schemes, we identify a number of intuitive variables to parameterize the redundancy space of the human body for task performance. Then we develop an analytical human posture control algorithm.

In Chapter 4, we present a numerical algorithm for human posture control in redundancy parameter space. This algorithm solves the human posture control problem better than existing numerical algorithms because it works in the redundancy space of the human body, a significantly smaller space than the joint space of the human body in which most existing algorithms work.

In Chapter 5, we describe a few experiments with a prototype implementation of the analytical and numerical human posture control algorithms. Finally, in Chapter 6, we summarize the contributions of this thesis and discuss future work.

Chapter 2

Human Posture Control: Problem Analysis and Our Approach

In this chapter, we first describe the articulated human model. Then we analyze in detail the posture control problem for human task simulations. The deficiencies of existing methods in the context of the human posture control problem are discussed in section 2.2. Based on the discussions, we propose our approach to solve the human posture control problem.

2.1 Articulated human figure model

In this study, we use the *Jack* ®[1] human model [1]. Figure 2.1 shows the articulated model of the human figure in *Jack*. The human posture control algorithm directly controls the following joints: the wrists ($2 \times 3 = 6$ dofs), the elbows ($2 \times 1 = 2$ dofs), the shoulders ($2 \times 3 = 6$ dofs)[2], the neck (3 dofs), the waist (3 dofs) [3], the hips ($2 \times 3 = 6$ dofs), the knees ($2 \times 1 = 2$ dofs), and the ankles ($2 \times 3 = 6$ dofs). In addition, it also controls the

[1] *Jack* is a trademark of the Trustees of the University of Pennsylvania.

[2] The shoulder joint in *Jack* is an abstract (group) joint with 3 dof. It represents the mobility of 2 real joints with 5 dof: the real shoulder joint with 3 dof, and the real clavicle joint with 2 dof.

[3] The human spine has a complex articulated structure. In *Jack*, the torso has a total of 51 dofs [1]. To simplify the control structure, a group joint $waist$ with 3 dof is used to represent the mobility of the torso structure.

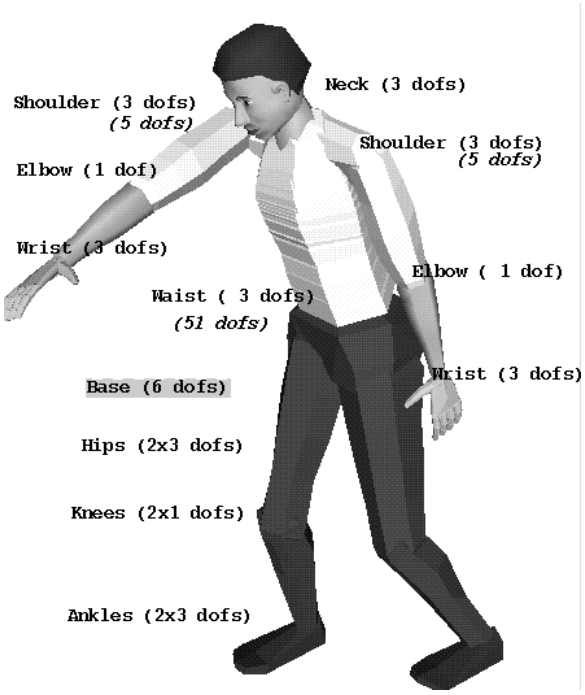

Figure 2.1: Articulated model of the simulated human figure *Jack*

position and orientation of the human figure which has 6 dofs. The total number of dofs in these joints is 40 (or 92 if we count all the real joint dofs).

Note that because of the spine model developed by Monheit and Badler [25], we only need 3 dofs to represent the human spinal structure of 51 dofs. This model provides a major reduction in the complexity of the problem.

2.2 Problem formulation and analysis

As described in Chapter 1, we need to control the position and orientation of the end-effector in simulating human tasks. In addition, we may also control how the task is to be performed by giving a set of task-related constraints CS, and a scalar performance criterion minimization function PC.

In this study, we are interested in geometrical constraints. These constraints are specified using the position and orientation information of the human body and the objects

in the environment. Similarly, the performance criterion function is also specified using position and orientation parameters.

Let $q = [q_1, q_2, .., q_n]^T$ be a set of joint parameters for the human body. All the possible values of q compose the joint space (also called the configuration space) of the human body. A human posture is defined by giving values to all of its joint parameters. Each posture is a point in the joint space. A posture q is *feasible* if and only if none of the constraints in CS is violated. A posture q is *a goal posture* if it places the end-effector at the goal position and orientation. Let $FS = \{q \mid \text{q is feasible}\}$ be the set of all feasible postures. The human posture control problem is to compute a feasible goal posture that minimizes the performance criterion function PC.

Mathematically, solving the human posture control problem corresponds to finding a point in the feasible joint space to minimize the performance criterion function specified by the user. As described in section 1.3, there are many approaches to solving this problem. Most of the approaches are search-based with different guidance information including function values, Jacobian, Gradient, and Hessian. These methods usually work well if the dimension of the search space is not large, and more importantly, if the objective and constraint functions are not very nonlinear. Unfortunately, neither is true in the case of human posture control. First, the search space of the human posture control problem we want to solve has a dimension of 40 (since there are 40 dofs to control), which is large. More importantly, the geometrical constraints and objective functions describe relations in 3D space, and they are highly nonlinear in joint space variables. Because of these two factors, there are a number of problems if we apply existing methods to the human posture control problem. We discuss these problems below.

2.2.1 Number of degrees of freedom

For posture control purposes, the human body has 40 degrees of freedom. Searching such a large space for a solution is computationally expensive, no matter which approach is

used. This problem can be overcome, to a certain extent, by using gradient (Jacobian) and Hessian information to speed up the search process.

2.2.2 Local minima problem

A more serious problem is the local minima problem: The search process may get stuck at a position (local minima) other than the goal. Since the performance criterion and constraint functions are highly nonlinear in joint angles, there may be many local minima. Further, usually it is not possible to choose the best local minima by finding all the minima and selecting one because of the large search space. Thus, in many cases solutions exist but we cannot find them using a search-based method.

2.2.3 Redundancy resolution and control

The human body is a redundant system for performing many tasks. While this is certainly an advantage from the task performance point of view, it also presents problems for human posture control. Among the infinite number of ways to perform a task, how do we choose a particular or "desirable" way of doing it. From a posture control point of view, what mechanisms should we provide so that:

1. the user can effectively and conveniently specify the "desired" postures, and

2. the posture control algorithm can compute the "desired" postures efficiently.

We call this problem the redundancy control problem. In existing approaches, the redundancy is not resolved effectively or conveniently. For example, in Jacobian-based or in optimization-based inverse kinematics algorithms, redundancy itself is not a problem. The algorithms resolve it automatically by providing a minimum change solution as is done in the pseudoinverse approach. The problem is that the solution obtained may not be what the user wants. Moreover, under these approaches, the user loses control over how the

redundancy is resolved. In order to regain control, the user may need to specify additional tasks (constraints) to perform in the Jacobian case, or additional criteria functions to minimize in the optimization case. Still there are two problems with this approach. First, it is not always easy to design an objective function to specify the intended characteristics of the desired postures or movements. Second, and more importantly, even if a criterion function can be specified that truly represents what the user wants, it is often impossible to compute the global minima solution of the nonlinear criterion and constraint functions, as explained earlier.

In summary, there are three problems associated with existing approaches when they are applied to solve the human posture control problem. The first one is high computational cost. The second one is the local minima problem. The third one is inefficient and ineffective redundancy control. In the next section, we propose a new approach to solve the human posture control problem which overcomes these problems.

2.3 Our decoupling approach

As described above, it is computationally expensive to search for a solution in a 40-dimension human joint space involving nonlinear constraint and performance criterion functions. In order to successfully solve the human posture control problem, it is very important to reduce the complexity of the problem.

Although human posture control is a very complex problem mathematically, humans do it every day with little effort. We are apparently not doing it by searching through a space of 40 dimensions. We must have strategies to reduce the problem complexity to a manageable size. It is not the aim of this thesis to investigate those strategies. Instead, we will study the mathematical structure of the problem and reduce the complexity through dof decoupling: partitioning the joint space into a set of independent subspaces. We illustrate the idea through the following example.

Let us consider human arm reach tasks. The human arm has 3 joints: the wrist with 3

dof, the elbow with 1 dof, and the shoulder with 3 dof. A reach task generally specifies the goal position and orientation of the hand (6 dofs). Using most existing approaches, this problem can be solved by searching the 7 dofs joint space of the arm for a point that places the hand at the goal position and orientation. Searching for a solution in a 7 dimensional space is computationally expensive. In general, there is no guarantee that a solution will be found even when one exists. And when there is no solution, the search process will not be able to tell. Search-based methods also have the local minima problem and the lack of effective redundancy control problem.

In fact, we can solve the arm reaching problem much better if we consider the special articulated structure of the arm [40]. Consider the figure of the human arm shown in Fig. 2.2. Given the shoulder position S and the hand (wrist) position W, the elbow angle θ_e is uniquely determined by equation 2.1:

$$d^2 = LU^2 + LL^2 - 2 * LU * LL * cos(\theta_e) \tag{2.1}$$

where d is the distance from the shoulder joint S to the wrist joint W, LU and LL are the upper and lower arm lengths, respectively.

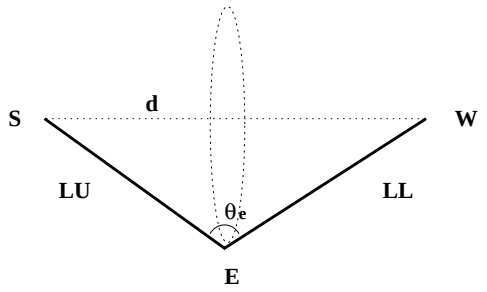

Figure 2.2: Human arm structure and the elbow joint angle computation

From the figure, it is easy to see that the 3 dofs at the wrist have no control over the wrist position. Their major role is to control the orientation of the hand without changing its position. So to a large degree, the control of hand position and orientation are decoupled: the shoulder and the elbow joint control the wrist position, and the 3 dofs at the wrist control hand orientation. Through this decoupled control scheme, the position

part and the orientation part of the human arm inverse kinematics problem can be solved independently. More importantly, their solutions can be computed using simple analytical formulas as shown by Tolani and Badler [40].

In order to have effective control over all available dofs in the human body, similar decoupling schemes are needed for the rest of the body. In Chapter 3, we analyze the articulated structure of the human body and propose a number of decoupling schemes that enable us to control the postures and movements of the whole human body analytically.

Note that Ching and Badler [7] employed decoupling to simplify the human motion planning problem. In their work, they used decoupling as a tool to speed up the search process. Our goal is to use decoupling to develop an analytical human posture control algorithm. The detail is discussed in the next chapter.

Chapter 3

Analytical Human Posture Control

In this chapter, we propose a decoupling approach for solving the human posture control problem. First, we analyze the human body articulated structure and identify several important decoupling schemes for human posture control. Based on these schemes, we parameterize the human body redundancy using a small number of intuitive control variables. Under this parameterization, we develop an analytical human posture control algorithm. There are two steps in the algorithm. Given the goal position and orientation, we first compute a default posture that places the hand at the goal. Using the default posture as the input, we control the redundancy in the posture to satisfy additional constraints (e.g., collision avoidance and task-related constraints).

3.1 Human posture control and the decoupling approach

As discussed in Chapter 2, there are a number of problems if we attempt to solve the human posture control problem in joint space using existing search-based approaches. These problems include high computational cost, the local minima problem, and the ineffective and inconvenient redundancy control problem. In order to overcome these problems, we must reduce the size of the search space. The reason that we have a joint space of 40 dimensions is that the dofs are coupled. Adjusting one dof changes the whole

body posture. For example, let us consider a human figure with the base [1] at the left toe. Adjusting one dof at the lower body causes the whole upper body to move. Assume the hand is at the desired position and orientation, and we need to move the left foot a little bit. After moving the foot, the hand will not be in the desired location any more. So we have to adjust the upper body again to restore the hand position. It is precisely this kind of coupling which makes the search space enormous.

To reduce the search space, we need to *decouple* the dofs so that their movements become independent. This decoupling approach can greatly simplify the human posture control problem, as will be shown throughout this chapter. In the following, we discuss several possible decoupling schemes useful for human posture control.

3.1.1 Decoupling upper and lower body movements

From experience we know that most human tasks are performed by the hands. In most cases, the lower body supports the upper body in performing a task, but the lower body itself does not perform the task directly. For this reason, we decouple the upper and the lower body degrees of freedom by treating them independently. The only constraint is that they connect to a common rigid body: the lower torso. The decoupling of the upper and the lower body can be done effectively by setting the figure base at the lower torso. By doing so, the upper body is controlled directly by the movements of the lower torso and not by the lower body joints. No mater how the joints in the lower body move, as long as the base (lower torso) does not move, the upper body does not have to move either. In many cases involving human task performance, the base (lower torso) position and orientation are determined by the task and the upper body dofs. This makes the lower body control simple: it just follows the movements of the lower torso in a natural manner. As a result, we may control the movements of the lower body independently of the upper body.

[1] Each figure has a coordinate frame called the base or the root. The positions and orientations of points on the figure are defined in this frame. These are the local coordinates. To move the figure, we only need to move the base (root) frame. Note that local coordinates of the points stays the same, and their global coordinates can be computed once we know the position and orientation of the base frame.

There are a number of advantages of this lower and upper body decoupling:

1. Reduction of configuration space:

   Before decoupling, all the lower body dofs ($>= 14$) appear in the configuration (joint) space as independent variables. After decoupling, only the 6 dofs of the lower torso (a rigid body) appear in the configuration space as independent variables (and only 4 of the 6 dofs are used to control human movements, as will be discussed further later). Also, the decoupling scheme eliminates the problem of maintaining the closed-loop constraint imposed on the two legs when the two feet are on the ground.

2. Simpler center of mass control:

   The lower torso is close to the human body's center of mass. The center of mass motions in many tasks (such as rising from a chair) can be mapped directly to the motions of the base site. Without decoupling, the center of mass control is more difficult. For instance, if the base of the figure is at one of the toes, it is very difficult to adjust the joint angles of the lower body such that the body's center of mass moves to a certain location, and at the same time maintain the closed-loop constraint on the two legs.

3. Analytical control of lower body:

   This decoupling not only gets rid of the closed-loop problem, but also makes it possible to control the lower body movements analytically. As we will show later, given the lower torso position and orientation, we will be able to compute the hip, knee, and ankle joint angles analytically to place the two feet at the desired positions and orientations.

3.1.2 Decoupling torso bending and arm movements

In addition to upper and lower body decoupling, the movements of the torso can also be decoupled from the movements of the arm. Note that the torso bending controls the positions and orientations of the neck and the shoulder. Once the shoulder position and orientation are fixed, the arm posture can be controlled analytically by specifying the hand goal position and orientation, as will be shown in section 3.4.1.

3.1.3 Decoupling position and orientation for arm posture computation

As described in section 2.3, we can decouple position and orientation in solving the arm reach problem. Through the decoupled control scheme, the human arm inverse kinematics problem can be solved analytically. In addition to the arm, we can also use position and orientation decoupling to solve the lower body posture control problem, as will be shown in section 3.4.5.

3.2 Human body redundancy parameterization

Given the hand goal position and orientation, there are an infinite number of postures which can place the hand at the goal. These postures are controlled by a set of control parameters. In the previous section, we discussed the decoupling of human body dofs for posture control. Here we identify the control parameters under this decoupled scheme:

1. Base (lower torso):

   The position of the base has 3 dofs: the (x, y, z) global coordinates. In this study, we only consider 1 dof in the base orientation: the angle of the forward direction (of the human body projected onto the XZ plane) with the X axis. Thus, for human posture control purposes, the base has only 4 dofs.

2. Upper Body (including the arm):

As shown in Fig. 2.1, with the base fixed and considering the joint chain in one arm, the upper body has 10 dofs: 3 at the waist, 3 at the shoulder, 1 at the elbow, and 3 at the wrist.

If we fix the hand (wrist) position and orientation, we only have 6 constraints. Thus, there are 4 dofs left in the upper body joint chain. There are many ways to parameterize the 4 dofs redundancy. Two examples of control parameterization are given below:

1. One way to parameterize the 4 dofs redundancy is:

   (a) Elbow height: With the shoulder and the hand positions fixed, the elbow traverses a circle. The elbow height controls the position of the elbow on the circle.

   (b) Torso flexion/extension: This dof controls the forward and backward bending of the torso.

   (c) Torso side bending: This dof controls the left and right side bending of the torso.

   (d) Torso twist: This dof controls the torso twist around the vertical direction.

2. Another way to parameterize the 4 dofs redundancy is:

   (a) Elbow flexion/extension: This dof controls the distance between the shoulder and the wrist joints.

   (b) Elbow height: With the shoulder and hand positions fixed, the elbow traverses a circle. The elbow height controls the positions of the elbow on the circle.

   (c) Shoulder height: With the base and the hand position fixed and the elbow angle given, the shoulder traverses a circle. The shoulder height controls the position of the shoulder on the circle.

   (d) Torso twist: This dof controls the torso twist around the vertical direction.

Each parameterization is more suitable than the other for certain applications. For example, if the elbow flexion angle must be controlled explicitly in one application, the second parameterization is a better choice. In this study, we use the first parameterization since it is simple and intuitive, and since for many human tasks we want to control torso posture (torso bending) explicitly.

When we consider both arm joint chains together, we add 7 more dofs to the system (3 dofs at the shoulder, 1 at the elbow, and 3 at the wrist) and 6 more constraints (the position and orientation of the other hand). This leaves 1 redundancy dof, the elbow height control. The corresponding parameterization for the redundancy space of postures involving two hands is:

1. Left elbow height;
2. Right elbow height;
3. Torso flexion;
4. Torso side bending;
5. Torso twist.

Given the hand position and orientation and the 8 control parameters for one hand tasks (4 base control parameters and 4 upper body control parameters), or the 9 control parameters for two hands tasks, the human body posture is completely determined. Specifically, the elbow height parameter controls the arm posture (arm twist); the torso bending parameters control the upper body and the arm posture; and the base position and orientation parameters control the whole body posture.

3.3 Analytical human posture control algorithm

As discussed in Chapter 2, the human posture control problem is to compute a posture that:

1. places the hand at the goal;

2. satisfies the geometrical constraints;

3. minimizes the performance criterion function.

To solve this problem, we developed a two-step analytical human posture control algorithm:

1. First, it computes a default posture that places the hand at the goal.

2. Second, it utilizes the redundancy in the posture to satisfy the geometrical constraints and to minimize the performance criterion function.

These two steps are discussed in sections 3.4 and 3.5, respectively.

3.4 Default posture computation

The problem we address in this section is: given the hand goal position and orientation, we wish to compute a default posture of the human body that places the hand at the goal. Depending on the starting joint in the joint chain from the base to the end-effector, we need to consider several cases:

1. The shoulder position and orientation are fixed. In this case, the upper body does not move and we want to compute a default arm posture that places the hand at the goal.

2. The base (lower torso) position and orientation are fixed. In this case, we want to compute a default upper body and arm posture that places the hand at the goal.

3. The base is mobile. Here we need to compute a whole body posture to place the hand at the goal.

We discuss these cases in sections 3.4.1, 3.4.2, 3.4.3, and 3.4.4. Note that the basic requirement for default posture computation is to compute a posture that puts the hand at the goal. Also the posture must obey the joint limits of the human body. While we also try to compute a reasonable (natural) default posture, this is not done rigorously and this is not an absolute requirement (the only absolute requirements are the hand goal constraint and the joint limits). From the task performance point of view, this is not a problem since the user can adjust the default posture either directly by adjusting the redundancy control parameters, or indirectly by giving a performance criterion function to minimize.

Before we proceed, we introduce the following notations that will be used throughout this report.

$\mathbf{A}_s^g$: global transformation matrix of the shoulder proximal frame;

S^g: global position of the shoulder;

$\mathbf{A}_w^g$: global transformation matrix of the wrist distal frame;

$\mathbf{A}_w^s$: local transformation matrix of the wrist distal frame in shoulder proximal frame;

W^g: global position of the wrist;

W^s: local position of the wrist in shoulder proximal frame;

E^g: global position of the elbow;

E^s: local position of the elbow in shoulder proximal frame.

3.4.1 Default arm posture computation

The problem we want to solve is: given the shoulder and the hand positions and orientations, we wish to compute a default arm posture (the shoulder, the elbow, and the wrist joint angles) that places the hand at the goal [2].

[2]This problem has been solved by Tolani and Badler [40]. We include it here for completeness. Note that here we give a simple treatment of this problem under the position and orientation decoupling scheme. For a more extensive discussion of this problem, see Tolani and Badler [40]

As discussed in Chapter 2, to solve the human arm inverse kinematics problem analytically, we decouple the control of the hand (wrist) position from the hand orientation. We use the wrist position to determine the shoulder and elbow angles, and the hand orientation to compute the wrist angles.

3.4.1.1 Determine the elbow angle

Given the wrist position W^g and the shoulder position S^g, the elbow angle $\theta_e \in [0, \pi]$ is uniquely determined by the following equation (see Fig. 2.2):

$$d^2 = LU^2 + LL^2 - 2 * LU * LL * cos(\theta_e) \tag{3.1}$$

$$\theta_e = acos((LU^2 + LL^2 - d^2)/(2 * LU * LL)) \tag{3.2}$$

where d is the distance from the shoulder joint S^g to the wrist joint W^g,

$$d^2 = (S^g - W^g) \cdot (S^g - W^g)$$

LU and LL are the upper and lower arm lengths. From equation 3.1, we know that *only* the elbow joint angle controls the distance between the wrist and the shoulder.

3.4.1.2 Control the elbow position

The human arm has 2 joints (with 4 dof) that control the position of the wrist: the shoulder joint (3 dofs) and the elbow joint (1 dof). Since the position in 3D space only has 3 dofs (3 constraints), the system has 1 dof redundancy in terms of wrist position control. One way to utilize this extra dof is to use it to control the height of the elbow as discussed by Korein [21], Tolani and Badler [40]. As shown in Fig. 2.2, the elbow traces through a circle in a plane perpendicular to the line connecting the wrist and the shoulder. Given the elbow position on the circle, the arm posture is uniquely determined.

In computing the default arm posture, we will find a default value of the elbow position control parameter through experiments. The objective is to put the arm in a relaxed posture. One possibility is to find an elbow position parameter value so that the wrist is close to the neutral position. We may also need to put the shoulder in a comfortable posture as well.

3.4.1.3 Determine the shoulder angles

Given the elbow position E^s and the wrist position W^s, we can determine the shoulder angles in a straightforward manner. In *Jack*, the real shoulder joint rotation axes are: Y, then X and then Z. Since the shoulder twist angle θ_{sz} does not change the elbow position E^s, we may use E^s to compute the first two dofs θ_{sy} and θ_{sx} (Note that there are two sets of solutions for θ_{sx}, θ_{sy}, and $\theta_{sz} \in [-\pi, \pi]$. They are denoted as: [θ_{sx}, θ_{sy}, θ_{sz}] and [θ'_{sx}, θ'_{sy}, θ'_{sz}].) :

$$E^s.x = LU * sin(\theta_{sy}) * cos(\theta_{sx}) \tag{3.3}$$

$$E^s.y = -LU * sin(\theta_{sx}) \tag{3.4}$$

$$E^s.z = LU * cos(\theta_{sy}) * cos(\theta_{sx}) \tag{3.5}$$

From equations 3.3, 3.4, 3.5 we have:

$$sin(\theta_{sx}) = -\frac{E^s.y}{LU}$$

$$cos(\theta_{sx}) = \pm\frac{\sqrt{(E^s.x)^2 + (E^s.z)^2}}{LU}$$

Thus, we can compute θ_{sx} as follows:

$$\theta_{sx} = atan2(-E^s.y, \sqrt{(E^s.x)^2 + (E^s.z)^2}) \tag{3.6}$$

$$\theta'_{sx} = \begin{cases} \pi - \theta_{sx} & \text{if } sin(\theta_{sx}) = -\frac{E^s.y}{LU} > 0; \\ -\pi - \theta_{sx} & \text{otherwise.} \end{cases} \tag{3.7}$$

From equation 3.6 we know that $cos(\theta_{sx}) >= 0$. Thus, $\theta_{sx} \in [-\pi/2, \pi/2]$. Correspondingly, $\theta'_{sx} \in [-\pi, -\pi/2]$ or $\theta'_{sx} \in [\pi/2, \pi]$. From equations 3.3 and 3.5, we can compute θ_{sy} as follows:

$$\theta_{sy} = atan2(E^s.x, E^s.z), \quad \text{since } cos(\theta_{sx}) >= 0. \tag{3.8}$$

$$\theta'_{sy} = \begin{cases} \theta_{sy} + \pi & \text{if } \theta_{sy} < 0; \\ \theta_{sy} - \pi & \text{otherwise.} \end{cases} \tag{3.9}$$

Given θ_{sy} and θ_{sx}, the elbow angle θ_e, and the position of the wrist W^s, we can compute the third shoulder angle θ_{sz} (shoulder twist) as follows:

$$A_{sy}A_{sx}A_{sz}T_{se}A_eT_{ew}[0\ 0\ 0\ 1]^T = W^s \tag{3.10}$$

where A_s are the transformation matrices for the dofs of the shoulder and the elbow joints, T_{se} and T_{ew} are the translation matrices from the shoulder joint to the elbow joint and from the elbow joint to the wrist joint, respectively. Everything except A_{sz} in equation 3.10 is known. From equation 3.10 and following some simple manipulations, we obtain the following equations:

$$LL * sin(\theta_e) * cos(\theta_{sz}) = W^s.x * cos(\theta_{sy}) - W^s.z * sin(\theta_{sy}) \tag{3.11}$$

$$\begin{aligned} LL * sin(\theta_e) * sin(\theta_{sz}) = {} & W^s.y * cos(\theta_{sx}) + sin(\theta_{sx}) * W^s.x * sin(\theta_{sy}) \\ & + sin(\theta_{sx}) * W^s.z * cos(\theta_{sy}) \end{aligned} \tag{3.12}$$

Thus, we can compute θ_{sz} as follows:

$$\theta_{sz} = \begin{cases} atan2(y, x) & \text{if } sin(\theta_e) >= 0; \\ atan2(-y, -x) & \text{otherwise.} \end{cases} \tag{3.13}$$

$$\theta'_{sz} = \begin{cases} \theta_{sz} + \pi & \text{if } \theta_{sz} < 0; \\ \theta_{sz} - \pi & \text{otherwise.} \end{cases} \tag{3.14}$$

$$y = W^s.y * cos(\theta_{sx}) + sin(\theta_{sx}) * W^s.x * sin(\theta_{sy}) + sin(\theta_{sx}) * W^s.z * cos(\theta_{sy}) \quad (3.15)$$

$$x = W^s.x * cos(\theta_{sy}) - W^s.z * sin(\theta_{sy}) \quad (3.16)$$

In the preceding discussions, we assumed that the shoulder is a spherical joint. In the *Jack* human model, this is not the case. The *Jack* shoulder is an abstract (group) joint that includes two real joints: the clavicle joint with 2 dof and the shoulder joint with 3 dof. To get around this problem, we use a similar trick employed by Tolani and Badler [40]: using the elbow position, we compute a set of default clavicle joint angles as follows:

1. Given the elbow position (x, y, z), we compute its spherical coordinates (θ, β, r) in the clavicle joint base frame.

2. We use θ and β as the clavicle joint angles (projecting them inside the joint limits when necessary).

It is possible that the above scheme may sometimes put the shoulder in awkward postures. For example, when the elbow position is straight down, the above scheme puts the shoulder in the down posture instead of the neutral posture most people maintain (i.e., $\beta < \beta_{neutral} = 0$). We may need to experiment with it to find a better scheme. For example, if there is no load on the hand, we may simply set $\beta = \beta_{neutral} = 0$ whenever $\beta < \beta_{neutral} = 0$.

3.4.1.4 Determine the wrist angles

We have determined the shoulder and the elbow angles. By doing so, the position and the orientation of the wrist joint (or its proximal site) have been determined. Since we know the hand's goal orientation, we may determine the wrist angle by simply using Roll, Pitch and Yaw angles extraction (for details see Paul [30]). Similar to the shoulder joint case, there are two sets of solutions which are denoted as [θ_{wx}, θ_{wy}, θ_{wz}] and [θ'_{wx}, θ'_{wy}, θ'_{wz}]. In *Jack*, the rotation axes of the wrist joint are: Z, then X and then Y. Thus, we have:

$$A_w^s = A_{sy}A_{sx}A_{sz}T_{se}A_eT_{ew}A_{wz}A_{wx}A_{wy}$$

$$\begin{aligned} A_{wz}A_{wx} &= (A_{sy}A_{sx}A_{sz}T_{se}A_eT_{ew})^{-1}A_w^s(A_{wy})^{-1} \\ &= T(A_{wy})^{-1} \end{aligned} \tag{3.17}$$

where

$$T = (A_{sy}A_{sx}A_{sz}T_{se}A_eT_{ew})^{-1}A_w^s = \begin{pmatrix} T_{xx} & T_{yx} & T_{zx} & T_{tx} \\ T_{xy} & T_{yy} & T_{zy} & T_{ty} \\ T_{xz} & T_{yz} & T_{zz} & T_{tz} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

From matrix equation 3.17, we have:

$$cos(\theta_{wy}) * T_{zx} + sin(\theta_{wy}) * T_{zz} = 0$$

Thus, we have:

$$\begin{aligned} \theta_{wy} &= atan2(-T_{zx}, T_{zz}) \\ \theta'_{wy} &= \begin{cases} \theta_{wy} + \pi & \text{if } \theta_{wy} <= 0; \\ \theta_{wy} - \pi & \text{otherwise.} \end{cases} \end{aligned} \tag{3.18}$$

From matrix equation 3.17, we also have:

$$0 = cos(\theta_{wy}) * T_{zx} + sin(\theta_{wy}) * T_{zz} \tag{3.19}$$

$$cos(\theta_{wz}) = cos(\theta_{wy}) * T_{xx} + sin(\theta_{wy}) * T_{xz} \tag{3.20}$$

$$sin(\theta_{wz}) = cos(\theta_{wy}) * T_{yx} + sin(\theta_{wy}) * T_{yz} \tag{3.21}$$

$$cos(\theta_{wx}) = -sin(\theta_{wy}) * T_{zx} + cos(\theta_{wy}) * T_{zz} \tag{3.22}$$

$$sin(\theta_{wx}) = T_{zy} \tag{3.23}$$

Thus, we have:

$$\begin{aligned}
\theta_{wz} &= atan2(cos(\theta_{wy}) * T_{yx} + sin(\theta_{wy}) * T_{yz}, \\
&\qquad cos(\theta_{wy}) * T_{xx} + sin(\theta_{wy}) * T_{xz}) && (3.24)\\
\theta'_{wz} &= \begin{cases} \theta_{wz} + \pi & \text{if } \theta_{wz} <= 0; \\ \theta_{wz} - \pi & \text{otherwise.} \end{cases} && (3.25)\\
\theta_{wx} &= atan(T_{zy}, -sin(\theta_{wy}) * T_{zx} + cos(\theta_{wy}) * T_{zz}) && (3.26)\\
\theta'_{wx} &= \begin{cases} \pi - \theta_{wx} & \text{if } sin(\theta_{wx}) = T_{zy} > 0; \\ -\pi - \theta_{wx} & \text{otherwise.} \end{cases} && (3.27)
\end{aligned}$$

3.4.2 Default torso posture computation

In subsection 3.4.1, we discussed the default arm posture computation when the shoulder position and orientation are fixed. In this subsection, we consider the same problem, except now the shoulder is mobile. We assume that the base (lower torso) is fixed.

As discussed in section 3.2, the joint chain from the lower torso to the hand has 4 dofs redundancy in placing the hand at the goal. To compute a unique default posture analytically, we need to assign values to the redundancy parameters. However, these redundancy parameter values cannot be assigned arbitrarily since some assignments may not produce a solution to the problem. That is, each of these parameters has a range of values, and this range depends on the hand goal position and orientation. In computing the default torso posture, we want to make sure that the goal is in the reachable space of the hand in the default torso posture. This problem is solved in two steps:

1. If the goal is comfortably (defined later) inside the reachable space of the hand in the current torso posture, we keep the current torso posture.

2. Otherwise, we compute a new torso posture so that the goal is inside the hand's reachable space.

In the following, we discuss these two steps in detail. Before we proceed, we would like to emphasize that while we are attempting to compute a reasonable or a natural posture, the only absolute constraint is that the hand is at the goal. As we pointed out at the beginning of this section, it is not a problem if the computed posture is not desirable since it can be adjusted later.

A related point is that we have much more freedom to compute the default upper and lower body posture than we have with the default arm posture. This is because for many positions and orientations of the shoulder, we can compute an arm posture to place the hand at the goal. For this reason, our main considerations in the default upper and lower body posture computations are simplicity and efficiency.

3.4.2.1 Reachable space approximation

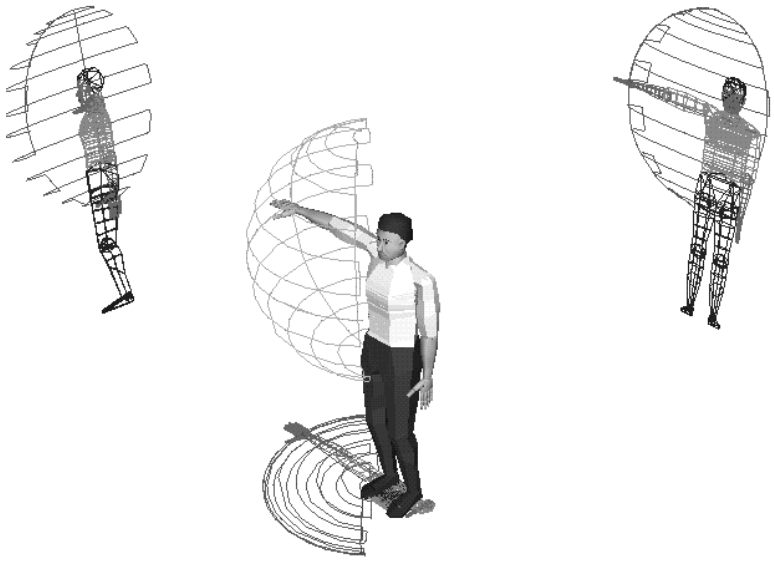

Figure 3.1: Reachable space of the hand and its projections (torso is straight)

As shown in Figs. 3.1 and 3.2 , with the shoulder position and orientation fixed, the reachable space of the hand can be crudely approximated using a half sphere. The origin

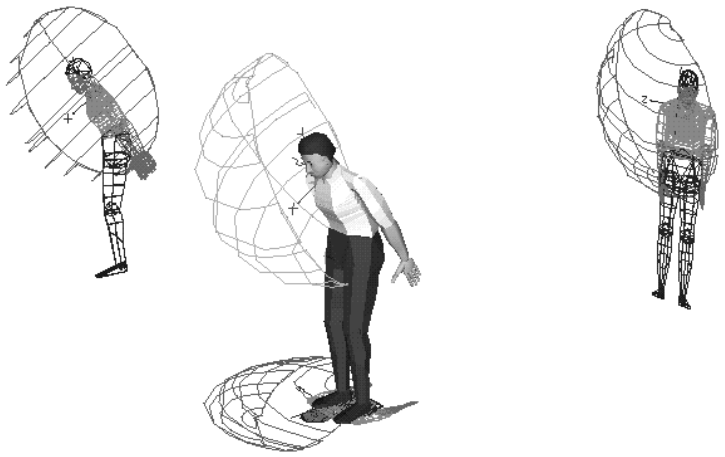

Figure 3.2: Reachable space of the hand and its projections (torso is bending)

of the sphere is the shoulder joint, the radius of the sphere is the arm length, and the X, Y, Z axis of the half sphere is approximately the clavicle joint base frame. Note that for efficiency and simplicity, here we only do a very crude approximation of the hand's reachable space. For our purposes, this is sufficient because we only need to compute a default posture so that the goal is *comfortably* inside the hand's reachable space. So even if there are large errors in our reachable space approximation, we can still compute an arm and body posture to put the hand at the goal.

For the right arm, the X, Y, Z axes of the clavicle joint in a standing posture are:

1. X: forward direction;
2. Y: upward direction;
3. Z: from left to right.

For the left arm, the X, Y, Z axes of the clavicle joint in a standing posture are:

1. X: forward direction;

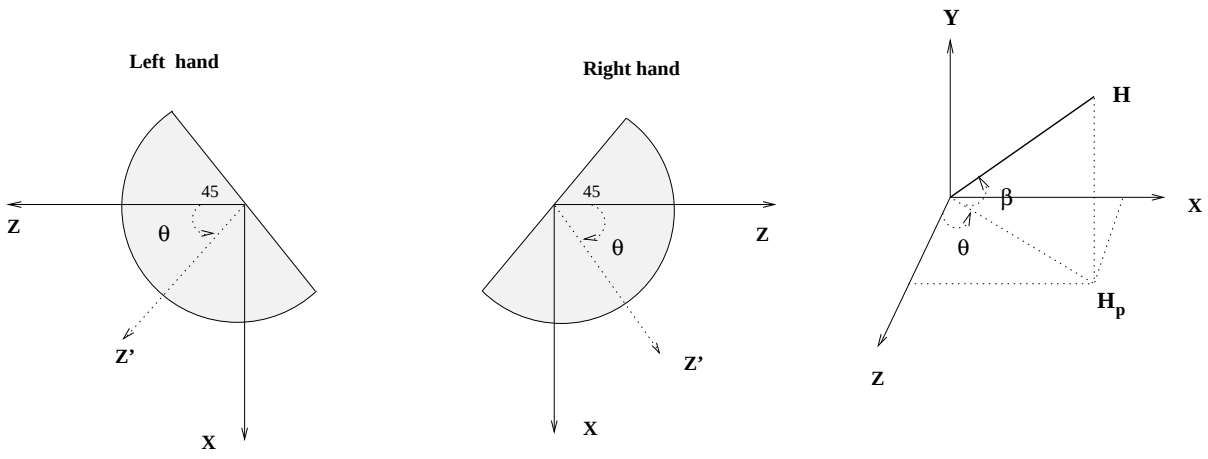

Figure 3.3: Half sphere approximations of the left and right hand reachable spaces

2. Y: downward direction;

3. Z: from right to left.

In spherical coordinates, the half sphere can be described as follows (see Fig. 3.3):

$$-45deg <= \theta <= 135deg \tag{3.28}$$

$$-90deg <= \beta <= 90deg \tag{3.29}$$

$$0 <= r <= LU + LL \tag{3.30}$$

To check if the hand goal position is inside the reachable space, we first compute its spherical coordinates $(\theta_{hg}, \beta_{hg}, r_{hg})$. If θ_{hg}, β_{hg}, and r_{hg} satisfy equations 3.28, 3.29 and 3.30, then we assume that it is inside the reachable space. Otherwise, it is outside the reachable space. For many applications, we may wish to place the goal *comfortably* inside the reachable space. There are many possible definitions of what is a comfortable subspace of the reachable space. One possible definition is as follows:

$$-10deg <= \theta <= 100deg \tag{3.31}$$

$$-65deg <= \beta <= 65deg \tag{3.32}$$

$$0.4 * (LU + LL) <= r <= 0.8 * (LU + LL) \tag{3.33}$$

If the hand goal position is *comfortably* inside the reachable space (satisfying equations 3.31, 3.32, and 3.33), then we keep the current torso posture. Otherwise, we compute a new torso posture as described below.

3.4.2.2 New torso posture computation

If the hand goal is not *comfortably* inside the reachable space, we try to compute a new torso posture to place the hand goal position inside the reachable space comfortably. In order to do this, we move the shoulder towards the goal or away from the goal as needed. Using the new shoulder position, we compute the corresponding torso posture (waist group joint angles) as follows:

1. First, compute the spherical coordinates $(\theta_{hg}, \beta_{hg}, r_{hg})$ of shoulder goal position under the figure's base frame (lower torso's base frame);

2. Project the spherical coordinates $(\theta_{hg}, \beta_{hg}, r_{hg})$ into the reachable space of the shoulder;

3. Compute the waist group joint angles using the projected spherical coordinates of the shoulder goal.

We will discuss coordinates projection and waist group joint angles computation in section 3.5.4. After computing a new torso posture, we can compute the arm posture as discussed in subsection 3.4.1.

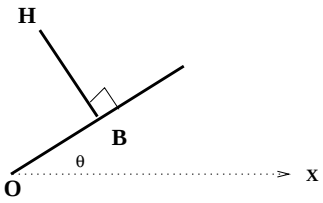

Figure 3.4: Compute the orientation of the base

3.4.3 Default base orientation computation

In this case, the base position is fixed, but the base orientation is free to rotate. We compute the default base orientation angle θ (i.e., the angle between the forward direction of the human body and the global X axis) such that the hand goal is at the front center of the body. Let B be the base position, and H be the hand goal position projected on the floor, this default base orientation can be computed as follows (see Fig. 3.4):

$$\theta = atan2(BH.y, BH.x) - 90deg \tag{3.34}$$

Given the base orientation, we can compute the upper body and the arm posture as discussed in subsections 3.4.1 and 3.4.2.

3.4.4 Default base position computation

In this case, the base position and orientation are mobile. We first compute the base orientation (using the current base position) as discussed in the previous subsection. In the following, we assume that the base orientation has been properly determined, and here we focus on the computation of the base position.

3.4.4.1 Base height and torso bending

In computing the base height, we adopt the minimal bending principle: do not bend the torso unless it is necessary. For simplicity, we assume that the waist group joint angles are 0 in the default posture. We also adopt the minimal squatting principle in controlling the

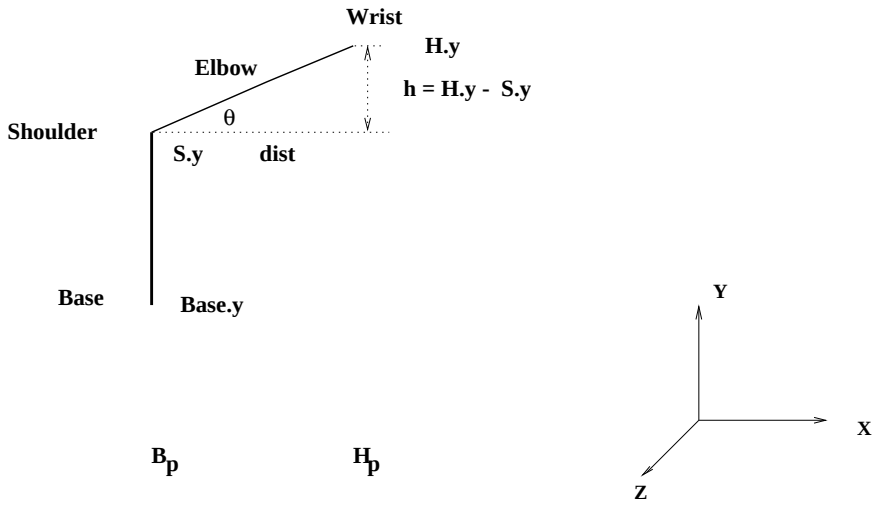

Figure 3.5: Compute the base position on the floor (XZ plane)

lower torso height: If the goal is reachable without squatting, we set the base height at the standing height. Otherwise, the base height is computed using the following equation:

$$Base.y + L_{torso} - LU - LL = H_{goal}.y \tag{3.35}$$

where L_{torso} is the length of the torso, LU and LL are the lengths of the upper and the lower arms. $H_{goal}.y$ is the height of the hand goal position.

3.4.4.2 Base floor position

Given the base height, now we compute the base position in the XZ plane so that the hand can reach the goal position.

Having determined the base orientation as discussed in the previous subsection, the desired position of the base on the floor (the XZ plane) lies in a straight line from the hand goal position (in the XZ plane) to the current base position (in the XZ plane). The distance from the base to the hand goal position (projected onto the XZ plane) determines the base position. The constraint on the distance is $[0, dist]$, where $dist$ is the maximum reachable distance of the hand in the XZ plane (see Fig. 3.5). The default distance of the

base from the goal is $0.5 * dist$. The maximum $dist$ can be computed as follows:

$$dist = (LU + LL) * cos(\theta)$$

where $sin(\theta) = (H_{goal}.y - Base.y)/(LL + LU)$. Then, the new base position in the XZ plane is ($Base.y$ is computed using equation 3.35):

$$Base_p = H_p - 0.5 * dist * B_p^c H_p / ||B_p^c H_p|| \tag{3.36}$$

where H_p and B_p are the projections of H_{goal} and $Base$ in the XZ plane, respectively, and B_p^c is the projection of the current base position in the XZ plane.

3.4.5 Default lower body posture computation

In this section, we address the following problem: given the lower torso position and orientation, and the requirement that the two feet be on the ground and the balance constraint is maintained [3], we wish to compute the hip, knee, and ankle joint angles.

Similar to the arm inverse kinematics problem, the lower body posture control problem can be solved analytically. Given the lower torso position and orientation and an upper body posture, we may approximate the center of mass position com. If com is inside the current foot support polygon, and the two feet are on the ground, we keep the current lower body posture. Otherwise, we compute new foot positions and a new lower body posture so that the center of mass is inside (near the center of) the support polygon.

Let the center of mass projected onto the XZ plane be at $com = (cx, 0, cz)$ in the body-fixed (at lower torso) coordinate, where positive Z aligns with the forward direction (see Fig. 3.6). Then, one way to compute the foot positions on the XZ plane is as follows (the Y component of the coordinates is 0):

[3] The balance constraint is that the center of mass of the human body, when it is projected on the floor (XZ plane), must stay inside the support polygon of the two feet.

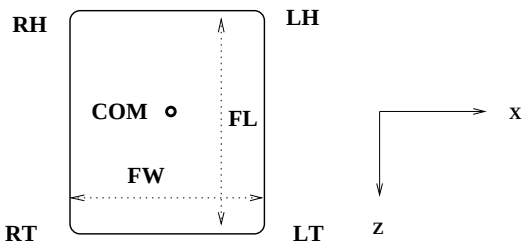

Figure 3.6: Foot position computation

$$LH = com + (0.5 * FW, 0, -0.5 * FL) \tag{3.37}$$

$$LT = com + (0.5 * FW, 0, 0.5 * FL) \tag{3.38}$$

$$RH = com + (-0.5 * FW, 0, -0.5 * FL) \tag{3.39}$$

$$RT = com + (-0.5 * FW, 0, 0.5 * FL) \tag{3.40}$$

where LH is the position of the left heel, LT is the position of the left toe, RH is the position of the right heel and RT is the position of the right toe. FL is the foot length and FW is the default distance between the two feet.

The heel and the toe positions fix 5 dofs of the rigid body of the foot. The other dof of the rigid body can be fixed by specifying the surface normal direction of the foot, which should be the global Y direction so that the foot is flat on the ground.

Given the positions and orientations of the two feet, the joint angles of the lower body can be determined as follows. Note that the articulated structures of the leg and the arm are the same. As a result, the solution to the leg inverse kinematics problem is similar to that of the arm inverse kinematics problem.

1. Knee joint angle:

This can be determined by the heel position and the hip joint position in the same way that the elbow angle is determined.

2. Knee height control:

   In most standing postures, the knee joint is straight. In this case, the position of the knee is almost uniquely determined by the hip and heel joint positions. With other postures, we can control knee twist similar to the way we control elbow twist.

3. Hip joint angles:

   They can be computed since the knee position is given, as is done in the case of the shoulder angles for the arm inverse kinematics problem.

4. Ankle angles:

   They can be computed using the foot orientation and the heel position in the same way that the wrist angles are computed.

3.5 Human body redundancy control

The human body has more degrees of freedom than necessary to perform tasks specified by the hand position and orientation. These extra dofs (redundancy) can be parameterized using the following control variables:

1. Base position (3 dofs) and base orientation (1 dof, rotation around the global Y axis).
2. Waist joint angles (3 dofs).
3. Elbow height: 2 dofs, 1 for each arm.

The total number of redundancy dofs is 9 if the base is mobile, and 5 if it is not. This is a large reduction in complexity from joint space control, which has 40 dofs. Moreover, the configuration space of these dofs can be further reduced by considering the following facts:

1. Each elbow height parameter can only control the posture of the arm it associates with. Also, it has control over the hand orientation, but not over the wrist position.

2. Waist joint angles only control upper body posture. They have no direct effect on the lower body posture. However, adjusting the waist joint angles may move the center of mass. As a result, they can affect the lower body posture indirectly.

3. Base position and orientation have control over the whole body posture.

Not only has the dimension of the configuration space has been reduced from 40 to 9, but the remaining control variables are more intuitive to use than joint space variables. This makes 3D space position control and geometrical constraints simpler to specify and solve.

Since we are developing a posture control algorithm for task performance, and most tasks are performed by the hands, in human body redundancy control we focus on the control of the arm posture. To solve this problem, we need to compute the arm position as a function of the control variables. Note that we need to compute both the forward and the inverse functions. That is, given the control parameters, we should be able to compute the arm and the body posture. Given the body posture or the positions of some points on the arm, we should be able to compute the corresponding control parameter values. These computations are discussed below.

3.5.1 Elbow position control

3.5.1.1 Forward elbow position control

The problem we are addressing here is: given the upper body (the torso) and the hand positions, we derive the formula for the elbow position as a function of the elbow height control parameter θ_{ec}.

As shown in Fig. 3.7, once the positions of the upper body and the hand are fixed, the elbow point E^g traverses a circle. The center of the circle, $O1$, can be computed as follows (also see Tolani and Badler [40]):

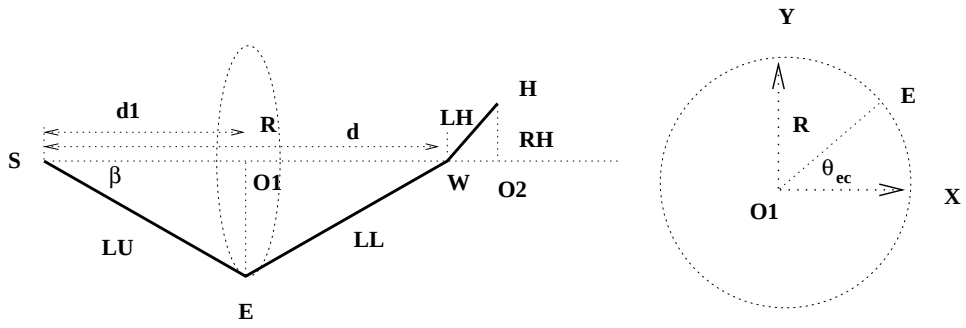

Figure 3.7: Elbow position control

$$\mathbf{n} = \frac{SW}{d} = \frac{W^g - S^g}{d} \tag{3.41}$$

$$O1 = S^g + LU * cos(\beta) * \mathbf{n} \tag{3.42}$$

$$cos(\beta) = \frac{d^2 + LU^2 - LL^2}{2 * LU * d} \tag{3.43}$$

Thus, we have

$$O1 = S^g + \frac{d^2 + LU^2 - LL^2}{2 * d} * \mathbf{n} \tag{3.44}$$

The radius of the circle traversed by the elbow point E is:

$$R = LU * |sin(\beta)| = LU * \sqrt{1 - (\frac{d^2 + LU^2 - LL^2}{2 * LU * d})^2} \tag{3.45}$$

The normal direction of the surface on which the circle lies is $\mathbf{n}$.

The arm posture can be controlled by giving the elbow position on the circle. To parameterize the elbow position on the circle we place two axes (X and Y) on the circle surface. Let $\mathbf{v}$ be the unit vertical-up direction of the shoulder frame. Then we define the X and Y vector (from $O1$) as follows:

$$O1X = \frac{\mathbf{v} \times \mathbf{n}}{||\mathbf{v} \times \mathbf{n}||} \tag{3.46}$$

$$O1Y = \mathbf{n} \times O1X \tag{3.47}$$

As shown in Fig. 3.7, the elbow position E^g on the circle can be parameterized using the following equation:

$$E^g = O1 + R * cos(\theta_{ec}) * O1X + R * sin(\theta_{ec}) * O1Y \tag{3.48}$$

where θ_{ec} is the angle between $O1E$ and $O1X$, rotated around the $\mathbf{n}$.

The hand position as a function of the elbow height control parameter θ_{ec} can be described similarly. First, the center of the circle is (see Fig. 3.7):

$$O2 = W + (WH \cdot \mathbf{n}) * \mathbf{n}$$

The radius of the circle is

$$RH = \sqrt{||WH||^2 - (WH \cdot \mathbf{n})^2}$$

Again, any point H on the hand circle can be characterized by the following equation:

$$H = O2 + RH * cos(\theta_{palm}) * O1X + RH * sin(\theta_{palm}) * O1Y$$

where θ_{palm} is the angle between $O2H$ and $O1X$, rotated around the vector $\mathbf{n}$. θ_{palm} and θ_{ec} are off by a constant θ_{const} if the wrist joint does not move. Given any arm and hand posture, this constant can be computed as follows: Use the $O2H$, $O1E$, $O1X$, and $O1Y$ to compute the corresponding θ_{palm} and θ_{ec}. Then

$$\theta_{const} = \theta_{ec} - \theta_{palm}$$

3.5.1.2 Inverse elbow position control

Here the problem we want to solve is: given the goal position E_g of the elbow, we want to compute an arm posture to place the elbow as close to the desired position as possible.

As shown in Fig. 3.8, we first project E_g onto the surface of the circle the elbow traverses, and then project it onto the circle as E_p:

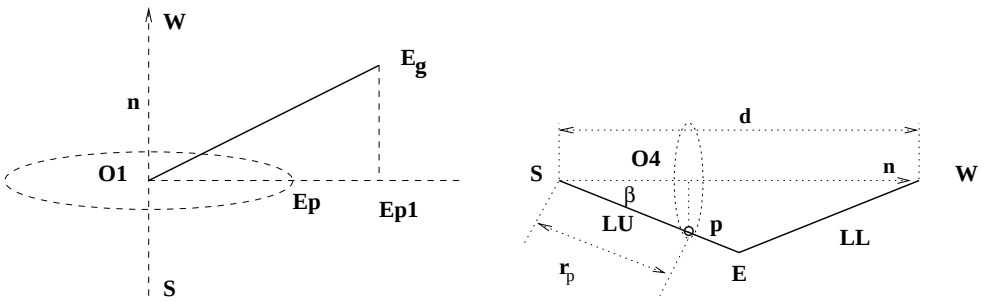

Figure 3.8: Inverse elbow position control

$$E_{p1} = O1 + O1E_g - (E_g \cdot \mathbf{n}) * \mathbf{n} \tag{3.49}$$

where $\mathbf{n}$ is the unit surface normal vector, and $O1$ is the center of the circle. To project the point onto the circle, we only need to make sure that vector $O1E_p$ has length R, the radius of the circle. That is,

$$E_p = O1 + O1E_{p1} * \frac{R}{||O1E_{p1}||} \tag{3.50}$$

E_p is the true goal position of the elbow. Using this elbow position, we can compute the shoulder and wrist joint angles as discussed in subsection 3.4.1.

3.5.1.3 Hand position control

Given the goal position of the hand, we project it onto the surface located at $O2$ (see Fig. 3.7), with the same surface normal $\mathbf{n}$, and then project it onto the circle as described above. Using the projected hand position H_p, we compute the elbow position as follows:

1. Compute the θ_{hc} using hand's current position H_c:

$$\theta_{hc} = atan2(O2H_c \cdot O1Y, O2H_c \cdot O1X)$$

2. Compute the θ_{hp} using the projected hand position H_p:

$$\theta_{hp} = atan2(O2H_p \cdot O1Y, O2H_p \cdot O1X)$$

3. Compute current elbow control parameter θ_{ecc} using the elbow's current position E_c:

$$\theta_{ec} = atan2(O1E_c \cdot O1Y, O1E_c \cdot O1X)$$

4. Compute the desired elbow height control parameter θ_{ec} as follows:

$$\theta_{ec} - \theta_{ecc} = \theta_{hp} - \theta_{hc}$$

$$\theta_{ec} = \theta_{ecc} + \theta_{hp} - \theta_{hc}$$

5. Compute the elbow's goal position E_g:

$$E_g = O1 + R * cos(\theta_{ec}) * O1X + R * sin(\theta_{ec}) * O1Y$$

where R is the radius of the elbow circle.

Using the elbow goal position, we can compute the shoulder joint angles as discussed in section 3.4.1.

3.5.2 Arm point position in terms of elbow height control

For a point p on the upper arm [4], we first compute the origin O_p of the circle it traverses. Then we compute the two axes on the circle surfaces as described in section 3.5.1:

$$\begin{aligned} O_p &= S^g + (r_p * cos(\beta)) * \mathbf{n} \\ &= S^g + \frac{r_p * (d^2 + LU^2 - LL^2)}{2 * d * LU} * \mathbf{n} \qquad (3.51) \\ O_pX &= \frac{\mathbf{v} \times \mathbf{n}}{||\mathbf{v} \times \mathbf{n}||} \qquad (3.52) \\ O_pY &= \mathbf{n} \times O_pX \qquad (3.53) \\ & \qquad (3.54) \end{aligned}$$

[4] We can derive similar equations for points on the lower arm

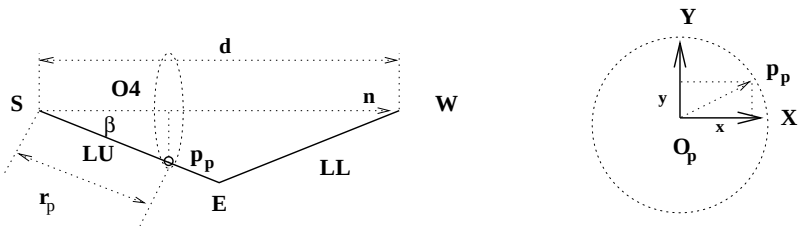

Figure 3.9: Inverse arm point position control

where v is the shoulder vertical axis and n is the unit vector from the shoulder to the wrist (which is also the circle surface normal). Then the position of the point on the circle can be parameterized as follows:

$$p = O_p + R_p * cos(\theta_{ec}) * O_pX + R_p * sin(\theta_{ec}) * O_pY \tag{3.55}$$

where R_p is the radius of the circle. Assume that the point on the upper arm with a distance to the shoulder r_p. Then R_p can be computed as follows:

$$R_p = r_p * |sin(\beta)| = r_p * \sqrt{1 - (\frac{d^2 + LU^2 - LL^2}{2 * d * LU})^2} \tag{3.56}$$

Given θ_{ec}, we can compute the location of point p on the circle using equation 3.55. Given the goal position p_g of point p, we project it onto the circle surface with origin O_p, and then project it onto the circle at p_p with radius R_p as done in equations 3.49 and 3.50. Using this projected position, we can compute the desired elbow position E^g and the elbow control parameter θ_{ec} as follows (see Fig. 3.8 and Fig. 3.9):

$$E^g = S^g + Sp_p * \frac{LU}{||Sp_p||} = S^g + Sp_p * \frac{LU}{r_p} \tag{3.57}$$

$$\theta_{ec} = atan2(O_pp_p \cdot O_pY, O_pp_p \cdot O_pX) \tag{3.58}$$

where Sp_p is the vector from the shoulder joint S^g to the projected position p_p of the point p. Using the computed elbow position, we compute the shoulder and the wrist joint angles as done in section 3.4.1. This works for all points on both the lower arm and the upper

arm.

3.5.3 Arm posture in terms of torso bending

Bending the torso moves the shoulder. In turn, this changes the arm posture so that the hand stays at the goal position and orientation. In the next section, we discuss the relationship between the shoulder position and the waist joint angles. Here we derive the relationship between the shoulder position and the arm posture.

The arm posture depends on two things: (1) the SW vector from the shoulder to the wrist, and (2) the elbow height control parameter θ_{ec}. The SW vector determines the circle the arm points traverse. In subsection 3.5.1, we discussed how to compute the positions of arm points given the shoulder position and θ_{ec}. Now we discuss how to compute the shoulder position given the θ_{ec} and the goal position of a point on the arm. There are two cases to consider: (1) when the point is on the lower arm, and (2) when the point is on the upper arm.

3.5.3.1 Point on the lower arm

In this case, we can compute the wrist angles (except the wrist twist angle θ_{wz}) easily to place the point at the desired position (or at the position that is closest to the desired position). Now the shoulder position is controlled by the elbow angle θ_e and the wrist twist θ_{wz}. These two dofs can be used to select a shoulder position to minimize some performance criterion function (e.g., closest to the current shoulder position, minimal torso bending, etc.). We develop a formula for the shoulder position in terms of these two dofs here.

Let Z be the global rotation axis of the wrist twist dof, and let θ_e be the elbow joint angle. We first compute the center of rotation O of the shoulder (see Fig. 3.10):

$$O = E^g + Z * (LU * cos(180deg - \theta_e)) \tag{3.59}$$

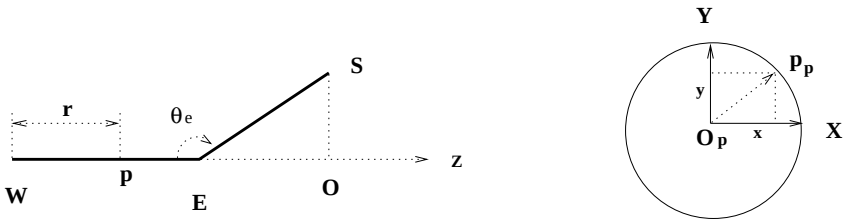

Figure 3.10: Control the position of a point on the lower arm

$$E^g = W^g + \frac{Wp * LL}{r} \tag{3.60}$$

where r is the distance from the wrist to point p. The radius of the circle that the shoulder traverses is:

$$rad = LU * sin(180deg - \theta_e) \tag{3.61}$$

Let us project the other two axes (X and Y) to the surface of the circle that the shoulder traverses. Since the normal direction of the surface is Z, both X and Y axes are parallel to the surface and X, Y remain the same. The points on the circle can be parameterized as:

$$S^g = O + rad * cos(\theta_{wz}) * X + rad * sin(\theta_{wz}) * Y \tag{3.62}$$

where θ_{wz} is the wrist twist angle.

3.5.3.2 Point on the upper arm

Since we know the point's global position, then the arm posture is controlled by the elbow twist θ_{ec}, the same as in the case where the shoulder position is fixed. In this case, the shoulder position traverses a circle (as shown in Fig. 3.11). We may use this dof to select a shoulder position to minimize some performance criterion function (e.g., closest to the current shoulder position, minimal torso bending, etc.). We derive the relationship for the shoulder position in terms of the elbow height control parameter here. First, the center of the circle O is computed as follows:

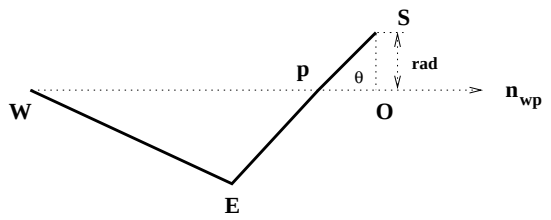

Figure 3.11: Control the position of a point on the upper arm

$$\mathbf{n}_{wp} = \frac{Wp}{||Wp||} \tag{3.63}$$

$$O = p + \mathbf{n}_{wp} * LU * f * cos(\theta) \tag{3.64}$$

where $cos(\theta)$ is computed as follows (assume the distance between p and the shoulder S is $f * LU$, 0 <= f <= 1, and the distance from elbow to p is $(1 - f) * LU$):

$$cos(\theta) = \frac{||Wp||^2 + ((1-f) * LU)^2 - LL^2}{2 * (1-f) * LU * ||Wp||} \tag{3.65}$$

$$rad = f * LU * |sin(\theta)| \tag{3.66}$$

We compute the two axes on the circle surface as follows:

$$OX = \frac{\mathbf{v} \times \mathbf{n}_{wp}}{||\mathbf{v} \times \mathbf{n}_{wp}||}$$

where $\mathbf{v}$ is a wrist axis vector which is not parallel to $\mathbf{n}_{wp}$.

$$OY = \mathbf{n} \times OX$$

Then the shoulder point can be parameterized as

$$S^g = O + OX * rad * cos(\theta_{ec}) + OY * rad * sin(\theta_{ec}) \tag{3.67}$$

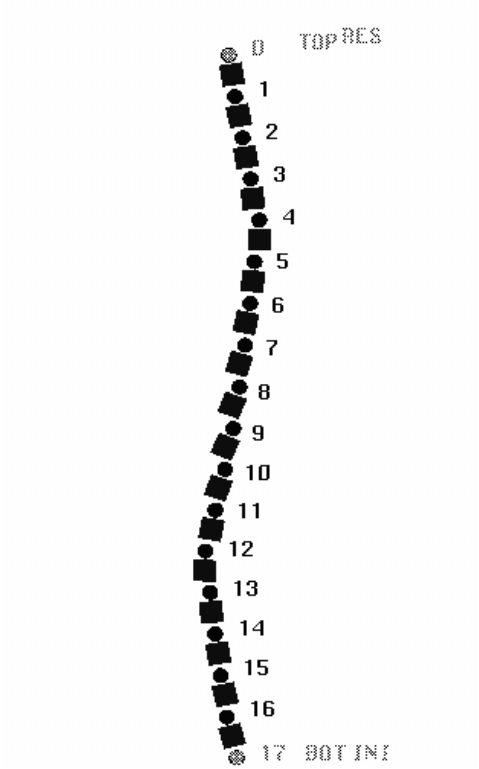

Figure 3.12: The spine structure of the *Jack* human model

3.5.4 Shoulder position control using the waist group joint

Given the waist group joint angles, computing the shoulder position may be easily solved using forward kinematics algorithms. Here we discuss the inverse problem: given the shoulder position, compute the waist group joint angles.

The waist is an abstract joint that represents the human spine structure. The kinematics property of the human spine is studied in Badler, et. al. [1] and [25]. Based on the model developed in [1] and [25] , the human spine structure is represented by 17 spherical joints with 51 dof (3×17) (see Fig. 3.12). The combined mobility of these 51 dofs is represented by the waist, an abstract (group) joint with 3 dof: flexion, side bending, and twist. The group joint angles are distributed into the real joint angles as described in [1].

Let $u \in [-1, 1], v \in [-1, 1]$, and $w \in [-1, 1]$ be the flexion, side bending, and twist group joint angles. The forward kinematics functions for the shoulder position (x, y, z)

can then be written as follows:

$$x(u, v, w) = f_x(u, v, w)$$

$$y(u, v, w) = f_y(u, v, w)$$

$$z(u, v, w) = f_z(u, v, w)$$

These functions can be computed analytically as any other open chain mechanism. In the following, we discuss the inverse kinematics computation of the torso structure.

Given x, y, and z, we wish to compute the group joint angles u, v, and w. That is, we want to compute the inverse of the above mappings:

$$u = f_u(x, y, z)$$

$$v = f_v(x, y, z)$$

$$w = f_w(x, y, z)$$

Apparently, the mapping between the group joint angles and the position of the neck (shoulder) is not simple. Computing its inverse mapping analytically is not an option. Instead, we approximate the inverse mapping using piece-wise quadratic functions.

For each given u, v, and w, we can compute the end-effector (e.g., neck or shoulder) position x, y, and z. From the data, we can build tables $x(u, v, w), y(u, v, w)$, and $z(u, v, w)$, where the u, v, and w vary in regular intervals. In our case, we need to compute the inverse mapping, i.e., the mapping from x, y, z to u, v, w. From the same data, we can build tables $u(x, y, z), v(x, y, z)$, and $w(x, y, z)$. Since x, y, and z do not vary in regular intervals, these data are called scattered data.

We tried to approximate the mapping $u(x, y, z), v(x, y, z)$, and $w(x, y, z)$ using techniques described in [16]. In particular, we implemented the multiple quadratic method (MQ) and Shepard method. Both methods failed to provide an acceptable solution for this 3 dimensional volume fitting problem: when we used the fitted u, v, and w to compute the forward kinematics for x, y, and z, the maximum error is more than 5 cm (about 10 percent relative error). In the following, we present an alternative approach to solving this problem.

3.5.4.1 Our approach

Even though the group joint is not a spherical joint, it is similar so we approximate it using a modified spherical joint. Let the joint angles of the spherical joint be θ, β, and γ. We would expect functions $u(\theta)$ and $v(\beta)$ to be quite simple. So instead of computing the mappings $u(x, y, z)$, $v(x, y, z)$, and $w(x, y, z)$ directly, we convert the Euclidean coordinates (x, y, z) into the spherical coordinates (θ, β, r). Then we use the spherical coordinates to compute the group joint angles u, v, and w. The details are discussed below.

3.5.4.2 Approximating $u(\theta, \beta)$ and $v(\theta, \beta)$ without torso twist

By holding v and w constant and plotting the function $u(\theta)$ (see Fig. 3.13), we found out that $u(\theta)$ can be reasonably approximated using two quadratic functions, one for $u >= 0$ and one for $u < 0$.

In order to approximate the $u(\theta)$ function using two quadratic functions, we need 5 data points:

$$(\theta_0 = \theta(u_0 = -1), u_0 = -1), \quad (\theta_1 = \theta(u_1 = -0.6), u_1 = -0.6),$$

$$(\theta_2 = \theta(u_2 = 0), u_2 = 0), \quad (\theta_3 = \theta(u_3 = 0.6), u_3 = 0.6), \quad (\theta_4 = \theta(u_4 = 1), u_4 = 1)$$

Using the 5 data points, we can fit the two quadratic functions as follows:

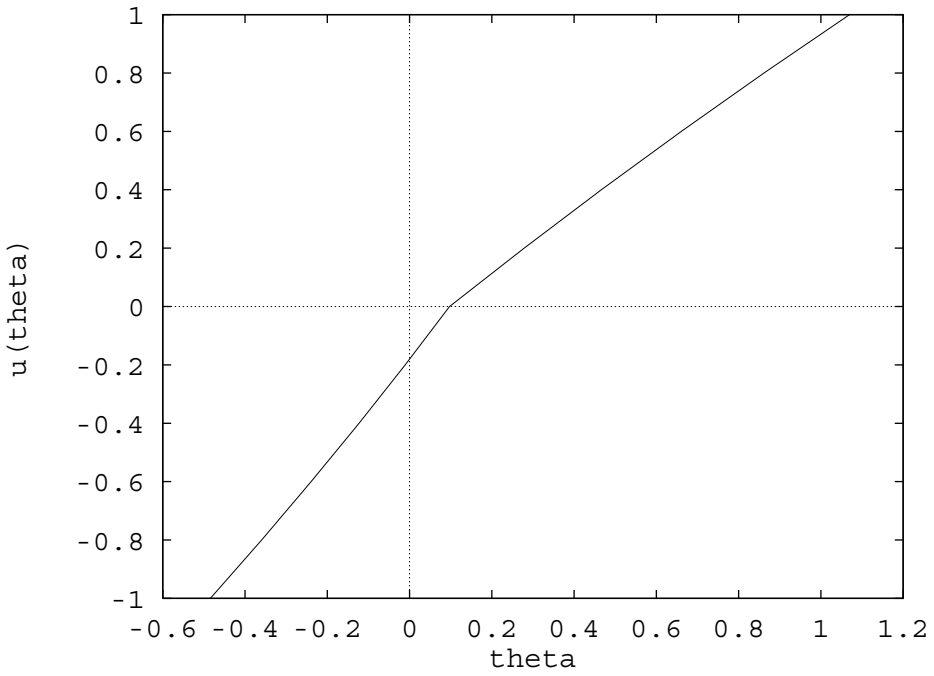

Figure 3.13: Function $u(\theta)$

1. For $u < 0$ (or $\theta < \theta_2$), we use points $(\theta_2, u_2), (\theta_1, u_1)$, and (θ_0, u_0) to fit the following function:

   $u_1(\theta) = a_1 * \theta^2 + b_1 * \theta + c_1$

2. For $u >= 0$ (or $\theta >= \theta_2$), we use points $(\theta_2, u_2), (\theta_3, u_3), (\theta_4, u_4)$ to fit the following function:

   $u_2(\theta) = a_2 * \theta^2 + b_2 * \theta + c_2$.

Then given a point (x, y, z), we can compute the corresponding spherical coordinates θ, β, and r. Also, using θ we can compute u using above formulas: if ($\theta < \theta_2$), we use function $u_1(\theta)$ to compute u; otherwise, we use function $u_2(\theta)$ to compute u.

If the torso flexion and side bending were independent, i.e., $\theta_0 = \theta(u_0 = -1)$ were independent of v, then the above computation would be correct. In reality, $\theta(u = constant)$ is a function of v. Fig. 3.14 shows a plot of $u(\theta)$ for $v = 0$ and $v = 1$. From the figure, it is easy to see that we have to consider the dependence of θ on v in our computation. That

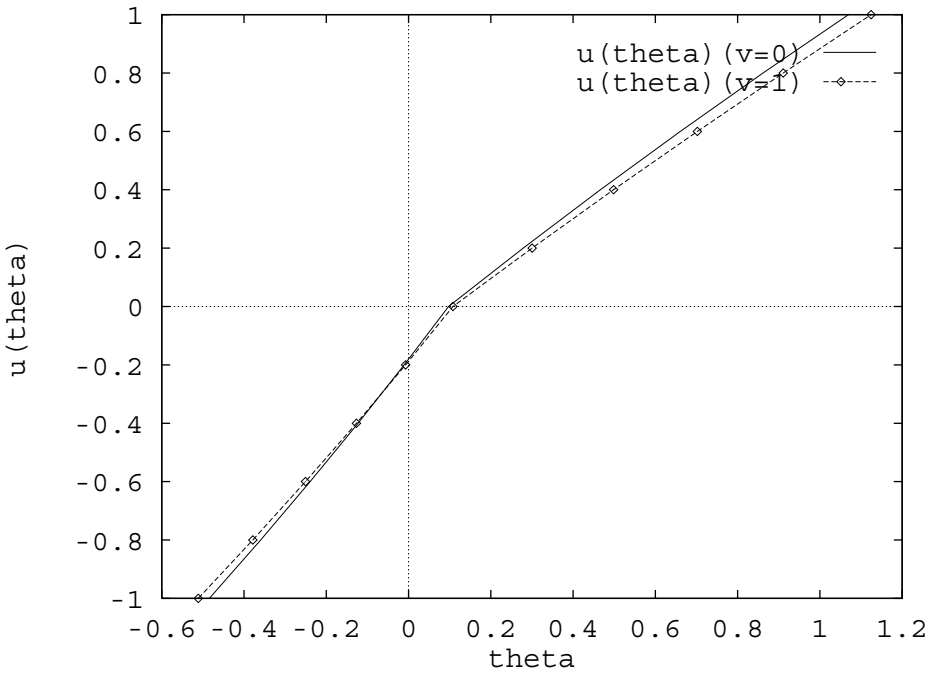

Figure 3.14: Function $u(\theta)$ with $v = 0$ and $v = 1$.

is, we have to compute the functions

$$\theta_0(v) = \theta(u_0 = -1)(v), \quad \theta_1(v) = \theta(u_1 = -0.6)(v)$$

$$\theta_2(v) = \theta(u_2 = 0)(v), \;\; \theta_3(v) = \theta(u = 0.6)(v), \;\; \theta_4(v) = \theta(u = 1)(v)$$

Even though we can approximate functions $\theta_0(v), \theta_1(v), \theta_2(v), \theta_3(v)$, and $\theta_4(v)$, this does not help because during the inverse kinematics computation, we do not know v. Our inputs are θ, β, and r. Since v depends primarily on β, instead of approximating $\theta(v)$ we approximate functions $\theta_i(\beta)$:

$$\theta_0(\beta) = \theta_0(v(\beta)) = \theta(u_0 = -1)(\beta(v))$$

$$\theta_1(\beta) = \theta_1(v(\beta)) = \theta(u_1 = -0.6)(\beta(v))$$

$$\theta_2(\beta) = \theta_2(v(\beta)) = \theta(u_2 = 0)(\beta(v))$$

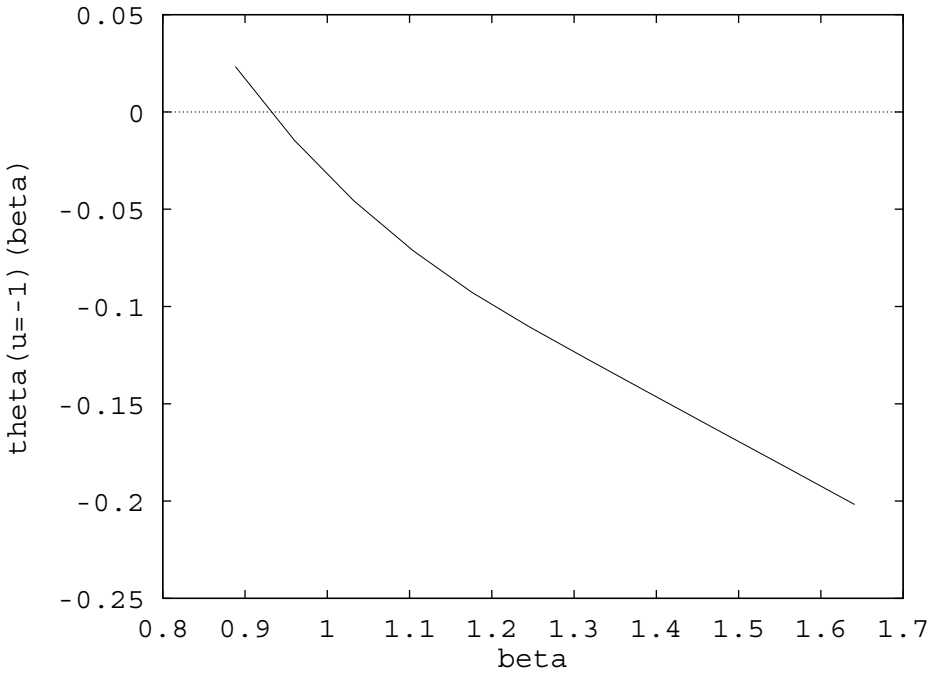

Figure 3.15: Function $\theta_0(\beta) = \theta(u = -1, \beta)$

$$\theta_3(\beta) = \theta_3(v(\beta)) = \theta(u = 0.6)(\beta(v))$$

$$\theta_4(\beta) = \theta_4(v(\beta)) = \theta(u = 1)(\beta(v))$$

From the plot of $\theta_0(\beta)$ (see Fig. 3.15), we can see that it can also be approximated by a quadratic function. Using 3 data points

$$(\beta(u = 0, v = -1), \theta(u = 0, v = -1))$$

$$(\beta(u = 0, v = 0), \theta(u = 0, v = 0))$$

$$(\beta(u = 0, v = 1), \theta(u = 0, v = 1))$$

we can approximate the quadratic functions $\theta_0(\beta)$ as follows:

$$\theta_0(\beta) = a_{\theta_0} * \beta^2 + b_{\theta_0} * \beta + c_{\theta_0}$$

and compute the fitting coefficients a_{θ_0}, b_{θ_0}, and c_{θ_0}. Similarly, we can approximate $\theta_1(\beta), \theta_2(\beta), \theta_3(\beta)$, and $\theta_4(\beta)$ using quadratic functions.

Given a point (x, y, z), we can compute θ, β, and r. Using β and the fitting function $\theta_i(\beta)$, we can compute $\theta_0, \theta_1, \theta_2, \theta_3$, and θ_4. Using these θ_i, we can compute the approximate functions $u_1(\theta)$ and $u_2(\theta)$. Then, using the input real θ and fitting functions $u_1(\theta)$ and $u_2(\theta)$, we can compute u.

Similarly, we can approximate v using β and θ. First, we approximate

$$\beta_0(\theta) = \beta(v = -1)(\theta), \quad \beta_1(\theta) = \beta(v = -0.6)(\theta),$$

$$\beta_2(\theta) = \beta(v = 0)(\theta), \quad \beta_3(\theta) = \beta(v = 0.6)(\theta), \quad \beta_4(\theta) = \beta(v = 1)(\theta)$$

using quadratic functions. For example, we can use the following 3 points:

$$(\theta(u = -1, v = 0), \beta(u = -1, v = 0))$$

$$(\theta(u = 0, v = 0), \beta(u = 0, v = 0))$$

$$(\theta(u = 1, v = 0), \beta(u = 1, v = 0))$$

to approximate the function $\beta_0(\theta)$:

$$\beta_0(\theta) = a_{\beta_0} * \theta^2 + b_{\beta_0} * \theta + c_{\beta_0}$$

Then we can approximate function $v(\beta)$ using two quadratic functions and one for $\beta >= \beta(u = 0, v = 0)$, one for $\beta < \beta(u = 0, v = 0)$ as follows:

1. For $\beta < \beta(u = 0, v = 0) = \beta_2$, we use points $(\beta_2, v = 0)$, $(\beta_1, v = -0.6)$, and $(\beta_0, v = -1)$ to fit the following function:

   $v_1(\beta) = a_1 * \beta^2 + b_1 * \beta + c_1$.

2. For $\beta >= \beta_2$, we use points ($\beta_2, v = 0$), ($\beta_3, v = 0.6$), and ($\beta_4, v = 1$) to fit the following function:

   $v_2(\beta) = a_2 * \beta^2 + b_2 * \beta + c_2$

Given β, we use function $v_1(\beta)$ to compute v if $\beta < \beta_2$; otherwise, we use function $v_2(\beta)$.

3.5.4.3 Approximating $u(\theta, \beta)$ and $v(\theta, \beta)$ with torso twist

In the preceding discussion, we ignored the torso twist component w. While u largely corresponds to θ and v largely corresponds to v, w does not identify with a single component of θ, β, and r. Instead of computing w directly as we did with u and v, we assume $w = 0$ and compute the real w based on the error caused by this assumption.

For each given w, the end-effector (controlled by u and v) traverses a surface. From this point of view, w can be seen as a parameter controlling which surface patch the end-effector is on. For our purpose, we sample the interval of $w \in [-1, 1]$ uniformly with 11 points. Let the sample array be

$$ws = \{-1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1\}$$

Now we can organize the data points in the following fashion:

$$\beta[i][j][k],\ \beta[i][j][k],\ \beta[i][j][k],\ \beta[i][j][k],\ \beta[i][j][k]$$

$$\theta[i][j][k],\ \theta[i][j][k],\ \theta[i][j][k],\ \theta[i][j][k],\ \theta[i][j][k]$$

where $0 <= i <= 4$, $0 <= j <= 2$, $0 <= k <= 10$. Data point $\beta[i][j][k]$ corresponds to $(\beta(u = a_5[i], v = a_3[j], w = ws[k]), v[i])$, and data point $\theta[i][j][k]$ corresponds to $(\theta(v = a_5[i], u = a_3[j], w = w[k]), u[i])$. Here $a_5 = \{-1, -0.6, 0, 0.6, 1\}$ and $a_3 = \{-1, 0, 1\}$ are sampled u and v data arrays.

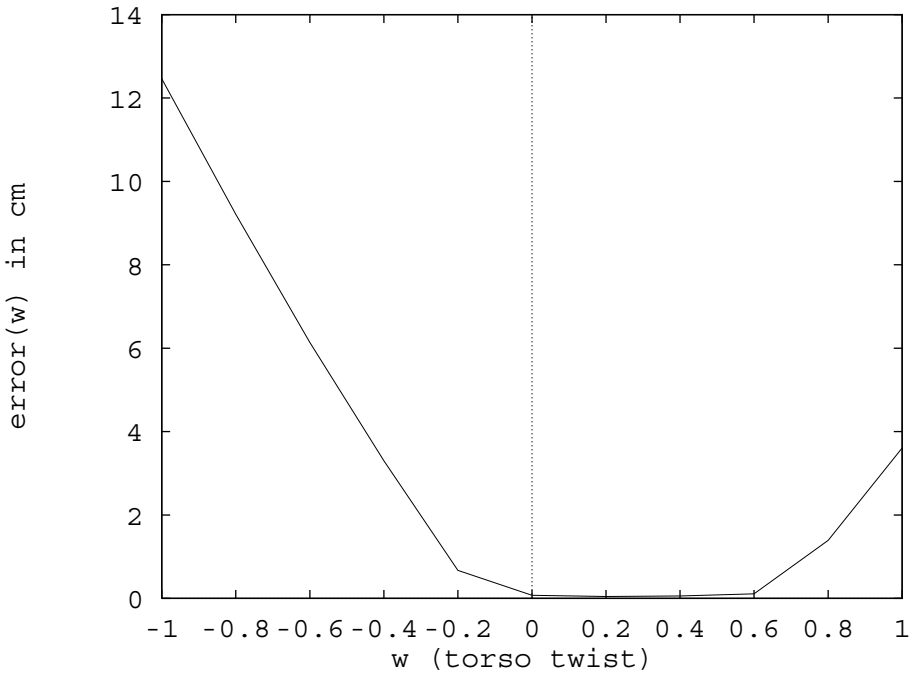

Figure 3.16: End-effector error (in cm) as a function of the torso twist parameter w

Using these data points, we can fit the functions $u(\theta, \beta)$ and $v(\theta, \beta)$ as described previously. Instead of putting w in the function's parameter list, we have a pair of $u(\theta, \beta)$ and $v(\theta, \beta)$ functions for each $w \in ws$.

For a given point (x, y, z) with θ and β, we compute the u parameter as follows:

1. Compute $\theta(u = a_5[i], \beta)$, 0 <= i <= 4.

2. Using the input θ, we choose 3 points from the computed $\theta(u = a_5[i])$ data points and fit a quadratic function $u_1(\theta)$ or $u_2(\theta)$, as described previously.

3. Using the fitting function $u_1(\theta)$ or $u_2(\theta)$ and the input θ, we compute the u parameter.

Note that we go through the above procedure for each $w \in ws$. As a result, we get one array of u parameters ua and one array of v parameters va. Each parameter in the array corresponds to a unique w parameter in the ws array.

To compute the w parameter, we use data points ($ua(w[k]), va(w[k]), w[k]$), 0 <= k <= 10 and forward kinematics to compute the end-effector position. The error (the

distance between the computed position and the goal position of the end-effector) is used to compute the value of the parameter w. One typical error curve is shown in Fig. 3.16 ($w = 0.2$). We approximate the error curve using a quadratic function (using the 3 points with smallest errors).

$$error(w) = a * w^2 + b * w + c$$

Using the fitted error function, we compute a $w \in [-1, 1]$ that makes $error(w)$ as close to zero as possible. This can be computed easily as follows:

1. If the equation $error(w) = 0$ has real roots, choose one in $[-1, 1]$. If the roots are outside $[-1, 1]$, project them inside.

2. If there are no real roots, we compute w by minimizing $error(w)$:

   $error'(w) = 0 = 2a * w + b = 0, => w = -b/2a.$

   Again, we have to project the solution inside $[-1, 1]$.

Using the computed w, we can compute the final values of u and v as follows:

1. From the computed w, we find an index k so that $ws[k]$ is the closest element to w in the array ws.

2. We choose 3 points $ua[k-1], ua[k]$, and $ua[k+1]$ around index k, and fit a quadratic function using the following data pairs:

   $(ws[k-1], ua[k-1]), (ws[k], ua[k]), (ws[k+1], ua[k+1])$.

   The approximate function

   $ua(w) = a * w^2 + b * w + c$

   and the computed w is used to compute the final u. We compute the final value of parameter v similarly.

Table 3.1: Average and maximum errors of the analytical torso inverse kinematics scheme

End-effector	*Average error (cm)*	*Maximum error (cm)*
Neck	0.047	0.25
Shoulder	0.15	0.94
Neck[4] (two hands)	0.047	0.33
Shoulder[4] (two hands)	0.22	1.35

3.5.4.4 Results

The scheme outlined above works fairly well in practice. We sampled 10,000 uniformly distributed random points in the (u, v, w) space and used the end-effector position $point = (xreal, yreal, zreal)$ to estimate the $(ufit, vfit, wfit)$ value. Using $(ufit, vfit, wfit)$ as the group joint angles, we computed forward kinematics for $pointfit = (xfit, yfit, zfit)$. The distance (in cm) between $point$ and $pointfit$ is the error measurement of our approximation scheme. The results are summarized in Table 3.1 (with 10,000 random samples) [5]. For the simulations of many human tasks, this magnitude of error is within the acceptable tolerance range.

3.5.4.5 Reachable space projection

If the input point is not in the reachable space of the end-effector, we need to project it into the reachable space to have accurate computation. For any given point (x, y, z), we can compute its spherical coordinates (θ, β, r). We project θ and β into the limits before using the approximation formula (there is no need to project r since it is not used in computation). From forward kinematics data, the lower and upper limits for θ and β are:

For the neck:

$$-0.555983 <= \theta <= 1.120201$$

[5] We use a different scheme to compute the waist joint angles when both the neck and the shoulder positions are given. First, we use the neck position (θ, β) to compute the first two dofs, u and v. Then, we use the error in the shoulder position to find the correct w. Using this scheme, the error for the neck position and the shoulder position are listed in the table.

$$1.094373 <= \beta <= 2.047220$$

For the left shoulder:

$$-0.970657 <= \theta <= 1.486352$$

$$0.671998 <= \beta <= 1.661777$$

For the right shoulder:

$$-0.973950 <= \theta <= 1.493876$$

$$1.477935 <= \beta <= 2.471256$$

3.6 Redundancy space posture representation

Traditionally, human postures are represented using joint angles. We can also use end-effector positions and orientations, and a set of redundancy parameters to represent human postures. However, this redundancy space representation RS is not really practical unless we can compute from it the corresponding joint space representation JS. This problem is solved with the development of an analytic human posture control algorithm in this chapter. It can convert representations from JS to RS and from RS to JS very efficiently. Because of this, the RS representation can be a real alternative to the JS representation.

In this section, first we briefly discuss converting postures between RS representation and JS representation. Then, we discuss the advantages the RS representation has over the JS representation.

3.6.1 Converting postures between joint space and redundancy space

Given a posture represented in joint space, we can convert it into redundancy space as follows:

1. Figure base position and orientation: given the angles of all joints in the body, we can compute the base (lower torso) position and orientation using forward kinematics.

2. Waist group joint parameters: they are identical in both representations.

3. Elbow control parameters: given the angles of all joints in the body, using forward kinematics we can compute the positions and orientations of the shoulder, the elbow, and the wrist. Then we can compute the elbow control parameter value using the method described in section 3.5.2.

4. Knee control parameters: they can be computed in the same way the elbow control parameters are computed.

Given a posture in redundancy space, we can convert it into joint space as follows:

1. Figure position and orientation: given the base (lower torso) position and orientation, and the joint angles computed in steps 2, 3, and 4 below, we can compute figure's global position and orientation using forward kinematics.

2. Waist group joint parameters: they are identical in both representations.

3. Shoulder, elbow and wrist joint angles: they are computed using the algorithm described in section 3.4.1.

4. Hip, knee and ankle joint angles: they are computed using the algorithm described in section 3.4.5.

Note that not only they are simple and straightforward, both conversions can also be done very efficiently since we only need forward kinematics computations.

3.6.2 Advantages of redundancy space posture representation

The RS representation has a number of advantages over the JS representation. For example, it is simpler to specify and manipulate postures using the RS representation than

using the JS representation. Using the JS representation, it is expensive and difficult to adjust the posture and keep the end-effector constrained at the goal. In the following we summarize a number of advantages the RS representation has over the JS representation when it is used in a distributed (networked) environment.

1. Less network traffic:

   In the RS representation, we may partially specify the posture by only giving the end-effector position and orientation. The redundancy parameters take default values when they are not specified, and we can still compute a posture that will place the end-effector at the goal. This cannot be done in joint space representation: the posture is not properly specified if any one of the joint angles is missing.

2. Higher reliability:

   The RS representation is more robust than the JS representation. A single joint angle error will cause the end-effector to be misplaced. On the other hand, errors in the redundancy parameters will not misplace the end-effector.

3. Looser coupling:

   In the RS representation, the posture is specified and manipulated by giving the end-effector goal and the redundancy parameter values. This specification is not directly coupled with the geometry of the figure. As a result, if link lengths or other geometric information are modified at the remote site, the modifications do not need to propagate back to the control site.

Chapter 4

Numerical Human Posture Control

In Chapter 3, we presented an analytical human posture control algorithm. It has several important advantages over existing methods: efficiency, no local minima problem, and convenient and effective redundancy control. But it also has a number of limitations:

1. It cannot handle over-constrained problems.
2. It cannot compute an approximate solution when there is no solution.
3. It cannot handle general geometrical constraint and performance criterion functions.

We discuss these problems in more details below.

Over-constrained problems and approximate solutions There are redundancies in the human body in performing many tasks. As discussed in the previous chapter, the number of redundancy dofs ranges from 4 to 9. These extra dofs can be used to accomplish other requirements such as collision avoidance and task-related constraints. There are situations where we may need more redundancy than is available. For example, if the shoulder position is fixed, then the arm has only one dof redundancy: the elbow height control parameter θ_{ec}. If we give the goal positions of more than one point on the arm, this problem becomes over-constrained. In this case, the analytical posture control algorithm

will not be able to compute a solution to put all the points at their goal positions. In many applications when there is no solution, we may wish to compute an approximate solution which puts the arm points as close to their goals as possible. Our analytical algorithm cannot compute an approximate solution.

General constraint and performance criterion functions Another limitation of the analytical algorithm is that it can only handle simple geometrical constraints and performance criterion functions. For example, it has no problem handling the geometrical constraints describing the positions of points on the arm. It can also handle minimal torso and minimal knee bending performance criteria easily. However, if the constraints and the performance criteria are general nonlinear functions of the geometrical parameters of the body, the analytical algorithm will not be able to handle them. For example, let

$$c(p) = 0$$

be a geometrical constraint describing the goal position of a point p on the body. Assume that $c(p)$ is a general nonlinear function of the position p. Also assume that we cannot compute the goal position of p analytically. Then the analytical human posture control algorithm will not be able to enforce this constraint.

To overcome these limitations of the analytical algorithm, we develop a numerical human posture control algorithm in this chapter. It is similar to existing methods in that we also use an optimization-based approach. Unlike existing methods where the optimization techniques are applied directly in joint space, here we transform the human posture control problem from joint space to redundancy parameter space described in the previous chapter. Through this transformation, the search space has been reduced from 40 to 9 or less. Then, we apply nonlinear optimization techniques to the transformed problem. Because of the smaller search space of the transformed problem, we can use a search-based numerical algorithm to solve it, and at the same time avoid the deficiencies associated with search-based methods as discussed in Chapter 2.

This chapter is organized as follows. We first give a brief description of nonlinear optimization. Then we formulate the human posture control problem in redundancy parameter space. A numerical algorithm based on Newton's method for nonlinear optimization is proposed to solve the formulated problem.

4.1 Nonlinear optimization and Newton's method

Nonlinear Optimization (or Nonlinear Programming NLP) solves the following problem:

$\min_x \ f(x)$

subject to constraints: $c_i(x) = 0$, $1 \leq i \leq k$, $c_i(x) \geq 0$, $k+1 \leq i \leq m$, where $f(x)$ and $c_i(x)$ are linear and nonlinear functions.

A local minima solution to the problem is x^* such that $x^* \in FS = \{x \mid \text{x satisfies all the constraints}\}$ and $f(x^*) \leq f(x)$ for all x in the neighborhood of x^* and $x \in FS$.

Let us assume that the initial point x^0 is close to the goal point x^*, a local minimizer of the problem. Since x^* is a local minimizer, it satisfies the following necessary condition [11]:

$$\bigtriangledown \mathcal{L}(x, \lambda) = 0 = [g - A\lambda, -c]^T$$

where $\mathcal{L}(x, \lambda) = f(x) - \sum \lambda_i c_i(x)$, g is the gradient of the function $f(x)$, A is the Jacobian matrix of the constraints, $\lambda = [\lambda_1, \lambda_2, ...]^T$ is the vector of Lagrangian multipliers, and $c = [c_1(x), c_2(x), ...]^T$ is the vector of the constraint function values.

In order to compute the local minimizer x^*, we solve the above stationary point equation using Newton's method. Assume we already have an initial guess for x^0 and λ^0. The Taylor series around x^k is:

$$\bigtriangledown \mathcal{L}(x^k + \delta_x, \lambda^k + \delta_\lambda) = \bigtriangledown \mathcal{L}^k + [\bigtriangledown^2 \mathcal{L}] \ [\delta_x \ \delta_\lambda]^T + ... = 0$$

$$[\bigtriangledown^2 \mathcal{L}]\ \ [\delta_x\ \ \delta_\lambda]^T \approx -\bigtriangledown \mathcal{L}^k$$

$$[\bigtriangledown^2 \mathcal{L}] = \begin{pmatrix} \frac{\partial^2 \mathcal{L}}{\partial^2 x} & \frac{\partial^2 \mathcal{L}}{\partial x \partial \lambda} \\ \frac{\partial^2 \mathcal{L}}{\partial \lambda \partial x} & \frac{\partial^2 \mathcal{L}}{\partial^2 \lambda} \end{pmatrix} = \begin{pmatrix} W & -A \\ -A^T & 0 \end{pmatrix}$$

where W is the Hessian matrix of the Lagrange function $\mathcal{L}$ with respect to variable x. Then we have

$$\begin{pmatrix} W & -A \\ -A^T & 0 \end{pmatrix} \begin{pmatrix} \delta_x \\ \delta_\lambda \end{pmatrix} = \begin{pmatrix} -g + A\lambda_k \\ c \end{pmatrix}$$

Thus,

$$\begin{pmatrix} W & -A \\ -A^T & 0 \end{pmatrix} \begin{pmatrix} \delta_x \\ \lambda_{k+1} \end{pmatrix} = \begin{pmatrix} -g \\ c \end{pmatrix}$$

The above algorithm assumes that the Hessian matrix W is available. If it is not, we may approximate it using an updating formula like BFGS. The details are given in Bartholomew-Biggs [3], Fletcher [11], and Gill, et. al [12].

4.2 Human posture control using nonlinear optimization

In the previous section, we gave a brief introduction to nonlinear optimization. In this section, we apply nonlinear optimization techniques to solve the human posture control problem.

4.2.1 Human posture control in joint space

Let $q = [q_1, q_2, .., q_n]^T$ be the joint variables of the human figure. Let X_d be the desired end-effector position and orientation, and let CS be a set of geometrical constraints which specifies the restrictions on how the task should be performed. Let PC be a geometrical performance criterion function. As discussed in Chapter 2, the human posture control problem can be stated as: compute a posture q such that

1. $X(q) = X_d$, where $X(q)$ is the forward kinematics function of the end-effector.

2. All constraints $c \in CS$ are satisfied.

3. Performance criterion function PC is minimized subject to 1 and 2.

We can solve this problem using nonlinear optimization:

$\min_q \ PC(q)$

subject to constraints: $X(q) = X_d$, $c(q) = 0$ and $c(q) <= 0$ for $c \in CS$.

In theory, there is no problem in this formulation. Any NLP (nonlinear programming) solver can be used to solve it. However, computationally there are many drawbacks associated with this formulation as discussed in Chapter 2. In the next subsection, we propose a new formulation of this problem so that computationally it is easier to solve. The key to this reformulation is based on our analysis of the human body articulated structure and our analytical posture control algorithm presented in Chapter 3.

4.2.2 Human posture control in redundancy parameter space

To formulate the human posture control problem in redundancy parameter space, we need to describe the geometrical constraints and the performance criterion functions using the redundancy parameters discussed in Chapter 3. Since redundancy parameters (e.g., elbow height) are more intuitive to understand than joint angles, the geometrical constraints and the performance criterion functions can be modeled more simply in redundancy parameter space.

As discussed in Chapter 2, in this thesis we are interested in geometrical constraints and geometrical performance functions. These are functions which can be described using the position and orientation parameters of the human body segments. To transform these function from 3D space to redundancy control parameter space (i.e. represent them using redundancy parameters), we use the equations derived in Chapter 3 which relate

the position of the body and the redundancy control parameters. These equations are summarized below:

4.2.2.1 Arm point position

The position of a point on the arm as a function of the elbow height control parameter is as follows:

$$p = O_p + R_p * cos(\theta_{ec}) * O_pX + R_p * sin(\theta_{ec}) * O_pY \quad (4.1)$$

$$cos(\beta) = \begin{cases} \frac{d^2+LU^2-LL^2}{2*d*LU}, & \text{if point } p \text{ is on the upper arm;} \\ \frac{d^2+LL^2-LU^2}{2*d*LL}, & \text{if point } p \text{ is on the lower arm.} \end{cases} \quad (4.2)$$

$$R_p = r_p * \sqrt{1 - cos^2(\beta)} \quad (4.3)$$

$$O_p = \begin{cases} S^g + r_p * cos(\beta) * \mathbf{n}, & \text{if point } p \text{ is on the upper arm;} \\ W^g - r_p * cos(\beta) * \mathbf{n}, & \text{if point } p \text{ is on the lower arm.} \end{cases} \quad (4.4)$$

$$O_pX = \frac{\mathbf{v} \times \mathbf{n}}{||\mathbf{v} \times \mathbf{n}||} \quad (4.5)$$

$$O_pY = \mathbf{n} \times O_pX \quad (4.6)$$

$$\mathbf{n} = \frac{SW}{d} = \frac{W^g - S^g}{d} \quad (4.7)$$

$$d = \sqrt{(SW.x)^2 + (SW.y)^2 + (SW.z)^2} \quad (4.8)$$

$$(4.9)$$

In above equations, the only unknowns are the shoulder position S^g and orientation (which determines $\mathbf{v}$), and the elbow control parameter θ_{ec}. The shoulder position and orientation are controlled by torso bending parameters, as well as the base position and orientation. Thus, we have described the position of p as a function of the redundancy control parameters. The unknown functions of the shoulder position and orientation in terms of the torso bending parameters and the base parameters are discussed below.

4.2.2.2 Shoulder position and orientation

In Chapter 3, we outlined a scheme to approximate the inverse kinematics mapping of the shoulder and the neck positions (in terms of the torso bending parameters). In this numerical posture control algorithm, we do not need the inverse mapping directly. We only need to compute the forward mapping, which is the straight forward kinematics problem for the torso joint chain. Thus, we have the analytical functions

$S_p = S_p(u, v, w)$ and $S_o = S_o(u, v, w)$,

where S_p is the shoulder position, S_o is the shoulder orientation, and u, v, and w are the torso bending parameters. We can use these functions to compute their gradients (Jacobian matrices) as well.

4.2.2.3 Human posture control in redundancy parameter space

Having transformed the geometrical constraint and performance criterion functions from 3D space to redundancy control parameter space, we can reformulate the human posture control problem as follows:

$\min_{rcp}\ PC(rcq)$

subject to constraints: $X(rcp) = X_d$, $c(rcp) = 0$ or $c(rcp) <= 0$ for $c \in CS$.

where rcp represents variables in redundancy control parameter space. We assume that initially the human body is in a posture where $X(rcp) = X_d$. This posture can be computed using the algorithm developed in Chapter 3. Other constraints ($c \in CS$) may or may not be satisfied at the initial posture.

In order to make sure that the computed posture is valid, we need to enforce joint limit constraints for each dof i:

$$llimit_i <= \theta_i(rcp) <= ulimit_i$$

Because of the joint limit constraints, we may simplify the hand goal constraint $X(rcp) = X_d$ to

$$||S^g(rcp) - W^g|| <= LU + LL$$

where S^g and W^g are the shoulder and wrist positions, respectively. As a result, the constraints we are maintaining are:

$$c(rcp) = 0 \quad or \quad c(rcp) <= 0 \qquad \text{for } c \in CS \tag{4.10}$$

$$llimit_i <= \theta_i(rcp) <= ulimit_i \qquad \text{for each dof } i \tag{4.11}$$

$$||S^g(rcp) - W^g|| <= LU + LL \tag{4.12}$$

4.3 Solving the human posture control problem using an NLP solver

As described in section 4.1, nonlinear optimization theory and Newton's method in solving NLP problems are pretty easy to understand. On the other hand, it is difficult to implement a good NLP solver which is efficient and stable. There are many theoretical issues, numerical analysis issues, and practical implementation issues to consider. For these reasons, instead of implementing a new NLP solver, we use a publicly available package *LBFGSB* [5]. In the following, first we briefly discuss the *LBFGSB* solver. Then we show how it is used to solve the human posture control problem.

4.3.1 The *LBFGSB* solver

The *LBFGSB* nonlinear optimization algorithm [5] and solver [47] was developed by Ciyou Zhu, Peihuang Lu, and Jorge Nocedal at Northwestern University and Richard H. Byrd at University of Colorado at Boulder. It minimizes a nonlinear function of n variables

$$\min_x \ f(x)$$

subject to the simple bounds

$$l <= x <= u,$$

where the vectors l and u represent the lower and upper bounds of the variables. To use this solver, the user must supply the gradient of $f(x)$. No knowledge of the Hessian matrix of $f(x)$ is required.

The algorithm is discussed in detail in Byrd, et. al. [5]. It iterates through the following steps until a feasible minimizer is located (i.e., the convergence criterion is met):

1. First, it computes a limited memory BFGS approximation to the Hessian matrix.

2. Then it uses this limited memory approximation to define a quadratic model of the objective function $f(x)$.

3. The search direction is computed next:

   (a) A set of active variables is identified using the gradient projection method. These variables will be held at their bounds during this iteration. The other variables are free.

   (b) The quadratic model is then minimized with respect to the free variables, and an approximate minimizer x_{min} is obtained. The search direction is defined to be $x_{min} - x^c$, where x^c is the current value of x.

4. Finally, a line search is performed along the search direction to minimize function $f(x)$ subject to the bound constraints.

The main advantage of the *LBFGSB* solver is its efficiency, especially for large problems in which the Hessian matrix is either not sparse or difficult to compute. It is implemented in FORTRAN 77, in double precision. [1]

Since *LBFGSB* can only handle simple bound constraints, we use the penalty method to handle more general linear and nonlinear constraints in our system. *LBFGSB* uses a quasi-Newton algorithm, which requires the computation of the objective (performance criterion) and constraint functions and their gradients. We discuss their computations in sections 4.3.2, 4.3.3, and 4.3.4.

4.3.2 Objective function and its gradient

In this study, we are interested in geometrical performance criterion functions. These functions can be described using the positions and orientations of points on the body or in the environment. That is, a geometrical performance criterion function or a constraint function can be written as

$$f(p_1(rcp), p_2(rcp), ...)$$

where p_i is a point on the human body. To compute the gradient of f, $\frac{\partial f}{\partial rcp}$, we need to compute the Jacobian matrix $\frac{\partial p_i}{\partial rcp}$.

If the point p is on the body and not on the arm, its position computation is the standard forward kinematics problem, and its Jacobian matrix computation is also straightforward. Here we only consider the case when point p is on the arm. We discussed its position evaluation (given rcp) in the previous subsection. In the following, we briefly discuss its gradient (Jacobian matrix) computation.

The redundancy parameters rcp include elbow twist θ_{ec}, waist group joint angles θ_{waist} and the base position and orientation $base_{po}$. Let rcp^{wb} be a subset of the rcp including θ_{waist} and $base_{po}$. From equation 4.1, we have:

[1] The code can be obtained by anonymous ftp to $eecs.nwu.edu$, in directory $/pub/lbfgs$.

$$\frac{\partial p}{\partial \theta_{ec}} = -R_p * sin(\theta_{ec}) * O_pX + R_p * cos(\theta_{ec}) * O_pY \quad (4.13)$$

$$\frac{\partial p}{\partial rcp^{wb}} = \frac{\partial O_p}{\partial rcp^{wb}} + [cos(\theta_{ec}) * O_pX + sin(\theta_{ec}) * O_pY]\frac{\partial R_p}{\partial rcp^{wb}} + R_p * (cos(\theta_{ec}) * \frac{\partial O_pX}{\partial rcp^{wb}} + sin(\theta_{ec}) * \frac{\partial O_pY}{\partial rcp^{wb}}) \quad (4.14)$$

From equation 4.4, we have:

$$\frac{\partial O_p}{\partial rcp^{wb}} = \frac{\partial S^g}{\partial rcp^{wb}} + \frac{\partial SO_p}{\partial rcp^{wb}} \quad (4.15)$$

$$\frac{\partial SO_p}{\partial rcp^{wb}} = (\frac{r_p}{LU} - \frac{r_p * cos(\beta)}{d}) * (\mathbf{n}\frac{\partial d}{\partial rcp^{wb}}) + \ r_p * cos(\beta) * \frac{\partial \mathbf{n}}{\partial rcp^{wb}} \quad (4.16)$$

if point p is on the upper arm. If point p is on the lower arm, we have (note that $\frac{\partial W^g}{\partial rcp^{wb}} = 0$):

$$\frac{\partial O_p}{\partial rcp^{wb}} = \frac{\partial W^g}{\partial rcp^{wb}} + \frac{\partial WO_p}{\partial rcp^{wb}} = (-\frac{r_p}{LL} + \frac{r_p * cos(\beta)}{d}) * (\mathbf{n}\frac{\partial d}{\partial rcp^{wb}}) - \ r_p * cos(\beta) * \frac{\partial \mathbf{n}}{\partial rcp^{wb}} \quad (4.17)$$

From equation 4.3, we have:

$$\frac{\partial R_p}{\partial rcp^{wb}} = -\frac{r_p * cos(\beta) * (d - LU * cos(\beta))}{LU * d * \sqrt{1 - cos^2(\beta)}} * \frac{\partial d}{\partial rcp^{wb}} \quad (4.18)$$

if point p is on the upper arm. If point p is on the lower arm, we have:

$$\frac{\partial R_p}{\partial rcp^{wb}} = -\frac{r_p * cos(\beta) * (d - LL * cos(\beta))}{LL * d * \sqrt{1 - cos^2(\beta)}} * \frac{\partial d}{\partial rcp^{wb}} \quad (4.19)$$

From equations 4.5 and 4.6, we have:

$$\frac{\partial \mathbf{vn}}{\partial rcp_i^{wb}} = \frac{\partial(\mathbf{v}\times\mathbf{n})}{\partial rcp_i^{wb}} = \frac{\partial \mathbf{v}}{\partial rcp_i^{wb}}\times\mathbf{n} + \mathbf{v}\times\frac{\partial \mathbf{n}}{\partial rcp_i^{wb}} \tag{4.20}$$

$$\frac{\partial O_pX}{\partial rcp_i^{wb}} = \frac{1}{||\mathbf{vn}||} * (\frac{\partial \mathbf{vn}}{\partial rcp_i^{wb}} - (\mathbf{vn}^T \cdot \frac{\partial \mathbf{vn}}{\partial rcp_i^{wb}}) * \frac{\mathbf{vn}}{||\mathbf{vn}||^2}) \tag{4.21}$$

$$\frac{\partial O_pY}{\partial rcp_i^{wb}} = \frac{\partial \mathbf{n}}{\partial rcp_i^{wb}}\times O_pX + \mathbf{n}\times\frac{\partial O_pX}{\partial rcp_i^{wb}} \tag{4.22}$$

where rcp_i^{wb} is the i-th variable in rcp^{wb}. From equation 4.7, we have:

$$\frac{\partial \mathbf{n}}{\partial rcp^{wb}} = -\frac{1}{d} * (\frac{\partial S^g}{\partial rcp^{wb}} + \mathbf{n}\frac{\partial d}{\partial rcp^{wb}}) \tag{4.23}$$

From equation 4.8, we have:

$$\frac{\partial d}{\partial rcp^{wb}} = \mathbf{n}^T\frac{\partial S}{\partial rcp^{wb}} \tag{4.24}$$

4.3.3 End-effector constraint

The end-effector constraint is described in equation 4.12, where S^g is the shoulder position and W^g is the wrist position specified by the user.

The shoulder position S^g is a function of the base position and orientation, as well as the waist group joint angles. The computation of S^g is the standard forward kinematics problem and its Jacobian matrix computation is also straightforward.

4.3.4 Joint limit constraints

In human body postures computation, we need to consider joint limits to ensure that the computed postures are valid. Since we are using redundancy parameters as the fundamental variables, we need to represent the joint angles as functions of redundancy space parameters. In Chapter 3, we derived the formulas for these functions. In the following, we compute their gradients.

4.3.4.1 Elbow joint angle

From equation 3.2 in Chapter 3, we have

$$-sin(\theta_e) * \frac{\partial \theta_e}{\partial rcp} = -\frac{1}{LL * LU} * \frac{\partial d}{\partial rcp}$$

$$\frac{\partial \theta_e}{\partial rcp} = \frac{d}{LL * LU * sin(\theta_e)} * \frac{\partial d}{\partial rcp} \tag{4.25}$$

4.3.4.2 Shoulder joint angles

From equations 3.3, 3.4, 3.5, and 3.16 in Chapter 3, we have:

$$\frac{\partial \theta_{sx}}{\partial rcp} = -\frac{\partial E^s.y}{\partial rcp} * \frac{1}{LU * cos(\theta_{sx})} \tag{4.26}$$

$$\frac{\partial \theta_{sy}}{\partial rcp} = \frac{\frac{\partial E^s.x}{\partial rcp} + LU * sin(\theta_{sx}) * sin(\theta_{sy}) * \frac{\partial \theta_{sx}}{\partial rcp}}{LU * cos(\theta_{sx}) * cos(\theta_{sy})} \tag{4.27}$$

$$\begin{aligned}\frac{\partial \theta_{sz}}{\partial rcp} = {} & (LL * cos(\theta_e) * cos(\theta_{sz}) * \frac{\partial \theta_e}{\partial rcp} - cos(\theta_{sy}) * \frac{\partial W^s.x}{\partial rcp} \\ & + (W^s.x * sin(\theta_{sy}) + W^s.z * cos(\theta_{sy})) * \frac{\partial \theta_{sy}}{\partial rcp} \\ & + sin(\theta_{sy}) * \frac{\partial W^s.z}{\partial rcp}) * \frac{1}{LL * sin(\theta_e) * sin(\theta_{sz})}\end{aligned} \tag{4.28}$$

To compute $\frac{\partial E^s}{\partial rcp}$, we note that $\mathbf{A}_s^g E^s = E^g$. Thus, we have:

$$\frac{\partial \mathbf{A}_s^g}{\partial rcp_i} E^s + \mathbf{A}_s^g \frac{\partial E^s}{\partial rcp_i} = \frac{\partial E^g}{\partial rcp_i}$$

$$\frac{\partial E^s}{\partial rcp_i} = (\mathbf{A}_s^g)^{-1} (\frac{\partial E^g}{\partial rcp_i} - \frac{\partial \mathbf{A}_s^g}{\partial rcp_i} E^s) \tag{4.29}$$

where rcp_i is the i-th variable in rcp. We can compute $\frac{\partial \mathbf{A}_s^g}{\partial rcp_i}$ using forward kinematics, and $\frac{\partial E^g}{\partial rcp_i}$ can be computed similar to the way $\frac{\partial p}{\partial rcp}$ is computed.

4.3.4.3 Wrist joint angles

Using equations 3.18, 3.26, and 3.27 in Chapter 3, we can compute the gradients of the wrist joint angles similarly to the gradients of the shoulder joint angles. In their computations, we need to evaluate $\frac{\partial W^s}{\partial rcp_i}$. Note that $\mathbf{A}_s^g W^s = W^g$. Thus,

$$\frac{\partial \mathbf{A}_s^g}{\partial rcp_i} W^s + \mathbf{A}_s^g \frac{\partial W^s}{\partial rcp_i} = \frac{\partial W^g}{\partial rcp_i} = 0$$

since W^g is independent of rcp_i.

$$\frac{\partial W^s}{\partial rcp_i} = -(\mathbf{A}_s^g)^{-1} \frac{\partial \mathbf{A}_s^g}{\partial rcp_i} W^s$$

The computation of $\frac{\partial \mathbf{A}_s^g}{\partial rcp_i}$ is straightforward using the forward kinematics computation.

Chapter 5

Experiments

In this chapter, we briefly describe our experiments with a prototype implementation of the analytical and numerical human posture control algorithms.

5.1 Experiments with the analytical human posture control algorithm

Computing human postures using the analytical algorithm described in Chapter 3 consists of two steps. First, we use it to compute a default human body posture to place the end-effector (hand) at the goal. We then adjust the posture to satisfy additional constraints such as collision avoidance, and optimize performance criteria such as minimal torso bending. We describe our experiments with both steps in sections 5.1.1 and 5.1.2.

5.1.1 Default posture computation

We start with the posture shown in the first frame of Fig. 5.1. The hand goals are the two small cubes. To compute a default posture, we check if the goal is inside the hand's reachable space. If it is, we only compute an arm posture to place the hand at the goal and

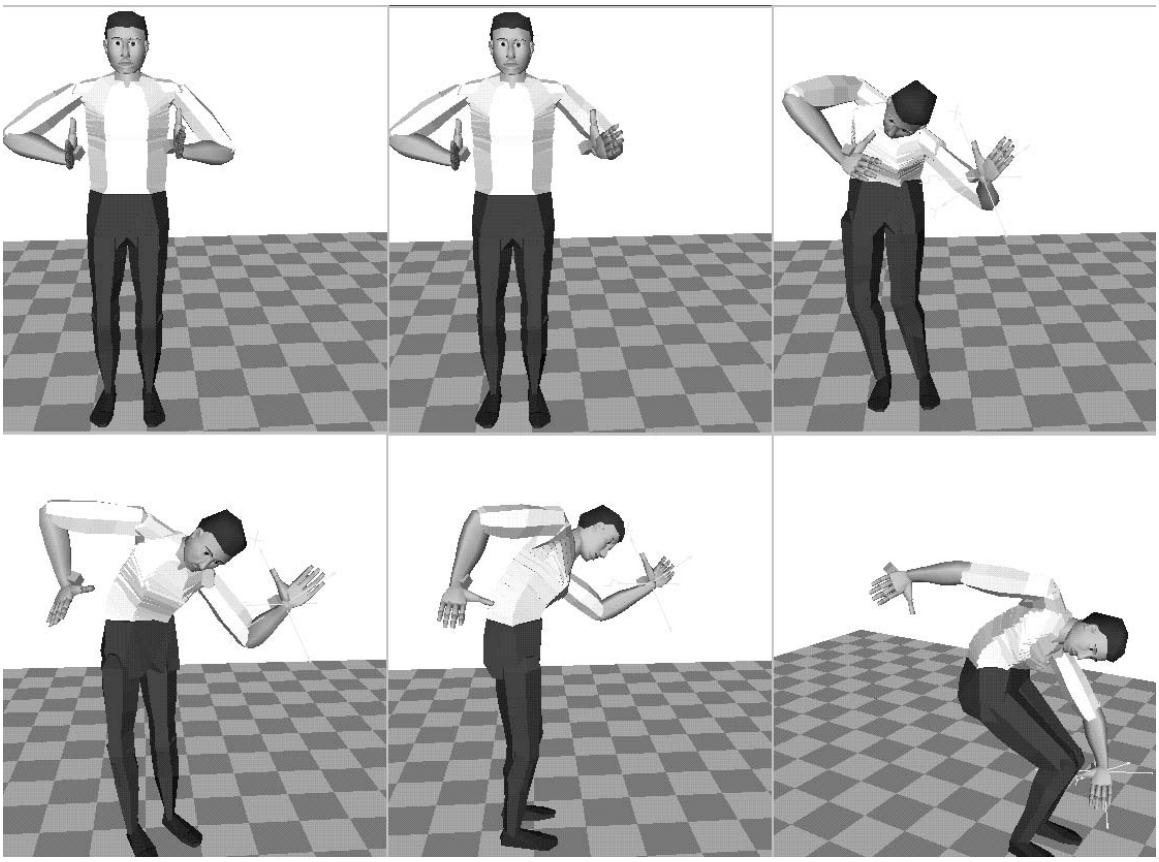

Figure 5.1: Step 1 of the analytical algorithm: default posture computation

the rest of the body does not move. This is shown in the second frame of Fig. 5.1, where the left hand goal moved to the left, and the right hand goal did not move.

If the goal is outside of the hand's reachable space, we put it inside by either bending the torso (frame 3), moving the figure towards the goal, (frame 4), adjusting its orientation (frame 5), or lowering its center of mass (frame 6).

Our experience with the default posture computation algorithm is that it does a good job at what it is designed to do: computing a default human body posture to place the hand at the goal. However, the computed default posture often may not be a good ("natural") posture. While we can certainly design better heuristics to compute more "natural" default postures, it is difficult to come up an algorithm which always produces "natural" postures. A different approach to solving this problem is to provide tools that allow the user to

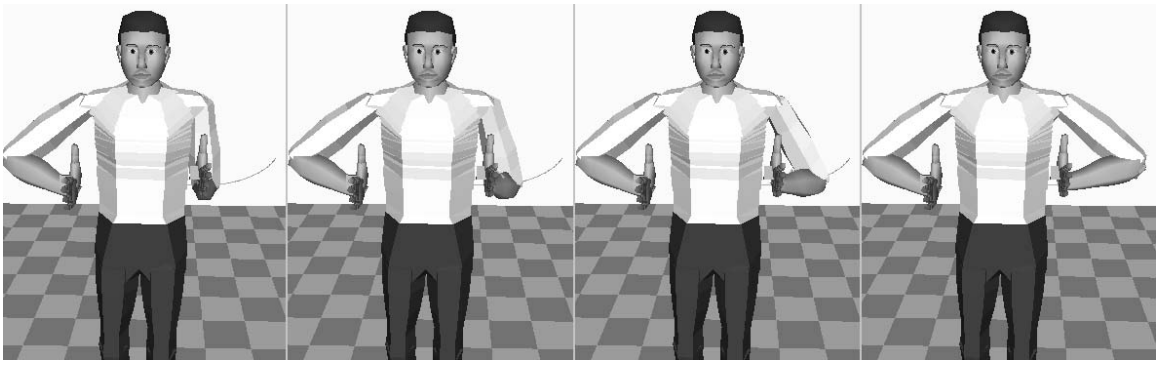

Figure 5.2: Redundancy control 1: adjusting elbow height control of the left arm

conveniently adjust the postures. This is accomplished in the second step of the analytical algorithm. We describe our experiments with it next.

5.1.2 Adjust posture in redundancy space

As mentioned in section 5.1.1, the default posture computation may not produce a "natural" or a desired posture. We can adjust the computed postures in redundancy space using the analytical algorithm.

As discussed in section 3.2, given the shoulder and wrist positions, the elbow traverses a circle. We can control the elbow position on the circle by giving the elbow height control parameter. This control is shown in Fig. 5.2. In the figure, the left elbow moves up while keeping the hand at the goal.

The other redundancy control parameters are torso bending and base position and orientation. We can adjust the torso posture as shown in Fig. 5.3, move the figure around and adjust its orientation as shown in Fig. 5.4.

Notice that while the body posture changes with new redundancy parameter values, both hands stay at their goals, and both feet stay on the ground. This means that we can concentrate on achieving the desired posture without worrying about the end-effector constraints. This is an important advantage over joint space manipulations where the user has to simultaneously manage two conflict goals of achieving the desired posture and

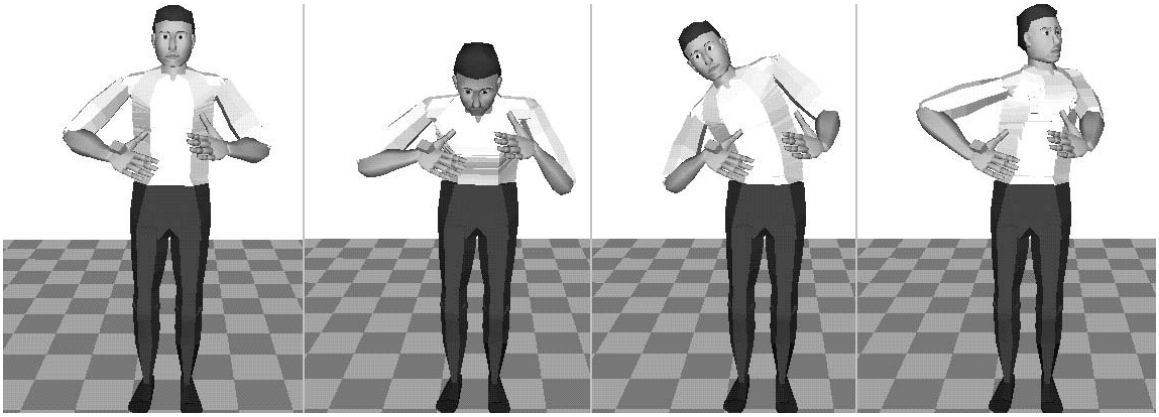

Figure 5.3: Redundancy control 2: adjusting torso flexion, side bending and twist

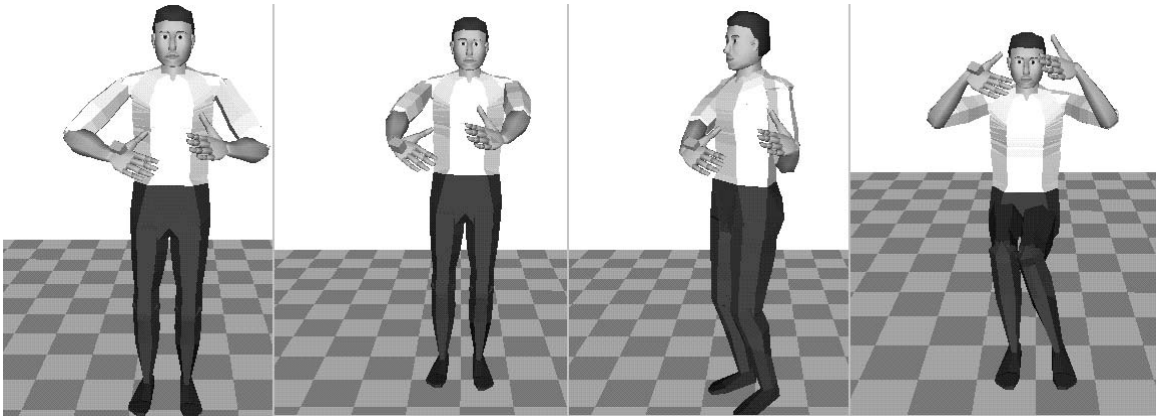

Figure 5.4: Redundancy control 3: adjusting base position and orientation

keeping the end-effectors at the goals.

5.1.3 Performance

At each frame, the analytical algorithm computes a whole body posture to place the end-effector at the goal. This includes computing the left and right arm postures, the torso posture, and the left and right leg postures. Arm and leg postures are computed using the algorithm described in sections 3.4.1 and 3.4.5. The torso posture computation is described in sections 3.4.2 and 3.5.4. In the following, we describe the performance of a prototype implementation of the analytical algorithm. The performance data, summarized in Table 5.1, is collected on a 200 MHZ SGI INDIGO-2 workstation.

Table 5.1: Performance data of the analytical human posture control algorithm

Posture	*Average computation time (sec)*	*Number of Human Figures controlled in real-time*
Arm	1.0E-4	330
Leg	1.0E-4	330
Torso (simplified)	3.0E-3	11
Torso	1.0E-2	3
Two Arms	2.0E-4	165
Whole Body (no joint limits)	3.4E-3	10
Whole Body (with joint limits)	1.1E-2	3

Given the shoulder and wrist positions and orientations, as well as the elbow control parameter, the arm posture computation takes 1.0E-4 seconds [1]. The leg posture computation takes the same amount of time since it uses the same algorithm. To compute arm and leg postures in real-time at 30 frames/second, during each frame we have 3.3E-2 seconds computation time. Thus, we can compute 330 arm or leg postures in real-time. If the task

[1]This is the average time obtained by running the program in a loop of 10,000 iterations

is to control one arm or leg movements, then we can control 330 human figures performing the task in real-time.

Given the base and shoulder (neck) positions and orientations, the torso posture computation takes 3.0E-3 seconds [2]. Note that we use a simplified, and less accurate version of the algorithm described in section 3.5.4. It does an adequate job for default postures computation where we do not need to position the neck or shoulder very accurately. To achieve the accuracy described in Table 3.1, the running time for the algorithm is 1.0E-2 seconds.

The human posture computation time varies from task to task and from frame to frame. For example, if the task only involves arm movements, then it takes 2.0E-4 seconds to control a human figure (time for computing two arm postures). That means we can control 165 human figures in real-time if the task only involves arm movements.

If the task involves whole body movements, i.e., we need to compute the arm postures, the torso posture, and the leg postures, then the total posture computation time is 3.0E-3 + 4*1.0E-4 = 3.4E-3 seconds. Thus, we can control whole body movements of 10 human figures in real-time. In this case, most of the time are used to compute the torso posture. The torso posture computation is expensive because the torso model in *Jack* has 17 joints with 51 dof. If we can reduce the torso posture computation time to 1.0E-4 by simplifying the model, we can reduce the whole body posture computation time to 5.0E-4 seconds. In this case, we can control whole body movements of 65 human figures in real-time.

The above performance analysis would be valid if there are no joint limits. In practice, the performance analysis is complicated by the presence of joint limits. For example, given the elbow control parameter, we can compute an arm posture (the shoulder, elbow, and wrist joint angles). However, the computed joint angles may be outside of the joint limits. In this case, we need to search for an elbow height control parameter value so that the computed posture is valid (within joint limits). Currently, we use a brutal force method to search for a valid elbow control parameter value. As a result, the arm and leg posture

[2]This is the average time obtained by running the program in a loop of 3,000 iterations

computation is about 20 times more expensive. Thus, each limb computation takes about 2.0E-3 seconds, and 4 limb computations take about 8E-3 seconds. If we include torso computation time, it takes 1.1E-2 seconds to compute a whole body posture.

Note that the prototype implementation is a straightforward one and it is not optimized for performance. There are a number of places where we can improve the performance considerably. For example, we can reduce the search time for a valid elbow (knee) control parameter value by using a better scheme (e.g., optimization). We can also simplify the torso model. As a result, we can reasonably assume that a good implementation of the analytical algorithm can control the movements of more than 10 human figures in real-time on today's workstations.

5.2 Experiments with the numerical human posture control algorithm

In section 5.1.2, we discussed interactive manipulations in redundancy space to obtain the desired posture. That is, the user finds the desired posture (i.e, a set of redundancy parameter values) by interactively adjusting the posture. We can also adjust postures automatically. The numerical posture control algorithm described in Chapter 4 optimizes postures in redundancy space to satisfy constraints and minimize performance criterion functions.

In performing many tasks, in addition to position the hands at given locations, we also need to control the positions and orientations of other parts of the body to avoid collisions or satisfy task-specific constraints. For example, in Fig. 5.5, *Jack*'s two hands are constrained to stay at current positions and orientations. We also want to position his left upper arm (the middle point of it) close to the small cube. Assume that the lower body and the base of the human figure do not move. We can use the numerical algorithm to automatically compute a posture that minimizes the distance between the left upper arm

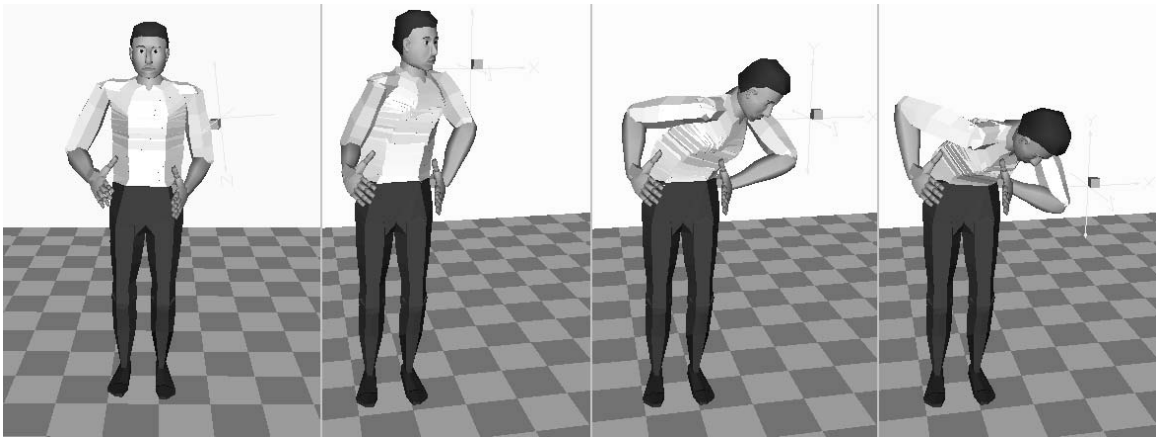

Figure 5.5: Minimizing the distance between the left upper arm and the small cube

and the small cube, and at the same time maintain the two hand constraints. As shown in Fig. 5.5, the small cube is being moved around, and *Jack* tries to position the left upper arm close to cube while maintaining the hand constraints.

Chapter 6

Conclusions

The goal of this thesis is to develop more efficient and more stable kinematic posture control techniques for human task simulations. To achieve this goal we have developed an analytical human posture control algorithm and a numerical human posture control algorithm in redundancy parameter space. The novel idea behind these two algorithms is a decoupling approach described in Chapter 3. Through decoupling we are able to greatly reduce the complexity of the human posture control problem and obtain an analytical solution. In the following we first summarize the contributions of this thesis. Then we briefly discuss future extensions to this work. We conclude this thesis with a brief discussion on the role the two posture control algorithms play in simulating human tasks.

6.1 Contributions

This thesis makes contributions to the areas of human factor analysis, virtual prototype evaluations, computer animation, and especially human and animal posture control. The contributions include:

1. A decoupling approach for complexity reduction:

We have proposed a decoupling approach to reduce the search space of the human posture control problem. Using this approach we have identified several decoupling schemes which make our analytical human posture control algorithm feasible.

2. An analytical human posture control algorithm:

   We have developed an analytical algorithm to control whole human body postures. The advantages of this algorithm over existing methods are that it eliminates the local minima problem, it is efficient enough to control whole human body postures in real-time, and it provides more effective and convenient control over redundant dofs.

3. A numerical human posture control algorithm in redundancy space:

   We have developed a numerical algorithm for human posture control. Like existing methods, it is based on nonlinear optimization theory. However, unlike existing methods which work in joint space, this algorithm works in redundancy parameter space. For the human posture control problem, redundancy parameter space is smaller than joint space. As a result, this numerical algorithm is better suited for solving the human posture control problem.

4. Animal posture control:

   While both the analytical and the numerical algorithms are designed for human posture control, they can also be used to control animal postures because of their similar articulated structure. The general idea of decoupling may be useful in studying the postures and movements of other articulated structures as well.

5. Redundancy space posture representation:

   We have proposed a new redundancy space representation for human postures. As discussed in section 3.6.1, we can efficiently convert postures between redundancy space representation and joint space representation. As a result, it can be a practical alternative to joint space representation. In addition, redundancy space posture

representation also has a number of advantages over joint space representation as discussed in section 3.6.2.

6.2 Future work

In this thesis, we have developed an analytical and a numerical human posture control algorithm. These two algorithms have several important advantages over existing methods as summarized above. There are also many important areas in which the two algorithms can be improved. We briefly discuss these improvements below.

6.2.1 Shoulder joint complex

In the *Jack* human model, the abstract 3-dof shoulder joint is a group joint encapsulating two real joints: the real shoulder joint with 3 dof, and the real clavicle joint with 2 dof. The shoulder group joint definition also specifies the constraints between the real shoulder and real clavicle joints.

In this thesis, we used a very simple scheme to decouple the two real joints in the shoulder group joint. First, the position of the elbow is used to compute the clavicle joint angles. After fixing the clavicle joint angles, the real shoulder joint angles can be computed easily since it is a standard spherical joint.

The problem with this scheme is that the proposed decoupling may violate the constraints that govern the relationship between the real shoulder and the real clavicle joints. In turn, it may result in unnatural postures in the shoulder joint complex. To overcome this problem, we can approximate the elbow position in terms of the shoulder group joint angles, similar to the way we handled the waist group joint.

6.2.2 Foot placement heuristics

Given the lower torso and the foot positions and orientations, we can compute the lower body posture analytically as discussed in Chapter 3. We have proposed a simple approach to compute the foot positions and orientations. It may be desirable to come up with a better method that takes into account the nature of the task, the upper body posture, the load distribution and the center of mass of the body.

6.2.3 Automatic decoupling

We have analyzed the articulated structure of the human body and proposed several decoupling schemes which make an analytical human posture control algorithm feasible. Due to similarities in body structure, these schemes are also useful for animal posture control. Currently we must identify these decoupling schemes manually for each articulated structure used. A valuable improvement would be to develop an algorithm that automatically identifies the decoupling schemes and sets up equations to solve the posture control problem analytically.

6.2.4 Posture interpolation in redundancy space

In this study, at each frame we compute the human body posture by considering the current hand goal and environment restrictions. This is sufficient for posture design. However, for movement generation, it is usually better to consider the coherence in the motion sequence and use the redundancy in the body to generate smooth motions to perform the task. In order to do this, we need to investigate the relationship between the coherence of the human posture in 3D space and its coherence in redundancy space. Using these relationships we can compute the redundancy parameters to ensure smooth human movements in 3D space.

Another possible approach to motion generation is through key framing. Given the starting and the ending postures, we can generate the in-between motions by interpolating

between these two postures. Unlike the traditional key-frame approach, where the interpolations occur in joint space or in 3D space, using our approach we can interpolate postures in redundancy parameter space. The advantage is that the hand position and orientation constraints are satisfied automatically. Also, since the redundancy parameters are intuitive, it is easier to come up an interpolation scheme that produces the desired movements.

6.2.5 Natural-looking human movements

In this thesis, we have developed new algorithms to solve the human posture control problem. The analytical algorithm provides an interface for exploring the redundancy in human postures while maintaining hand position and orientation constraints. The numerical algorithm allows the user to specify a performance criterion function to achieve the desired posture. While all these may help, we need to do much more to generate natural-looking human movements.

A possible approach is to first understand the human motor control mechanism [32] [9]. Based on the model of the human motor control mechanism, we can compute the muscle forces the human body generates for task performance. Then we can feed these forces into a dynamics simulator to synthesize natural-looking human movements. Unfortunately, our current understanding of the human motor control mechanism is not good enough to enable us to predict the correct muscle forces.

Another possible approach is to use optimal control. Biomechanical studies have revealed that the performance of human tasks often involves the optimization of some criteria such as minimization of work, reaction force, or torque. In [4] and [42], Brotman and Netravali, Witkin and Kass have demonstrated the usefulness of this technique for generating natural-looking motions for simple objects. Pandy et. al. [28] [29] used optimal control to study human movements. The drawback with this approach is inefficiency. Because of the complexity of the problem, it is computationally expensive to compute a solution. In [46], we have proposed a way to improve the efficiency and accuracy of

the solution procedure for certain types of problem. However, this improvement is small compared to the computational demand of the problem. A more serious problem is the local minima problem associated with optimization-based approaches. Until we have better ways to address these two problems, the optimal control approach can only be used in a limited domain of applications.

6.3 Conclusions

Human task simulation is important to virtual prototype evaluation, human factor analysis, and virtual environments. A key component of human task simulation is the human posture control problem. In this thesis, we have developed an analytical and a numerical algorithm for human posture control. These two algorithms are efficient and stable. They also provide convenient and computationally effective redundancy control.

While they are developed for controlling human body postures, the two posture control algorithms do not have any built-in knowledge about the human body other than its articulated structure. They solve the human posture control problem as a pure geometrical problem. The domain specific information about the particular task and the human body is coded in task specification. This information includes a hand goal location (or series of goals yielding a goal trajectory), a performance criterion minimization function, and a set of geometrical constraints. Thus, the two human posture control algorithms can be viewed as engines that enforce geometrical constraints and minimize geometrical performance criterion functions.

To build a human task simulation system, we need two higher-level systems that interface the posture control system. First, we need a system that relates body postures to human performance criteria such as comfort, fatigue, postural stability, visual perception, and hand manipulability (including the hand's mobility and its ability to exert force and torque). We call this part the *human performance* system. We also need a system which relates tasks to hand goal location (or hand goal trajectory), to geometrical restrictions,

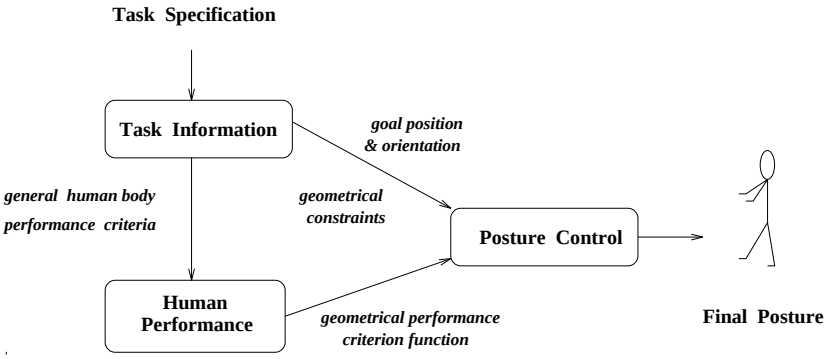

Figure 6.1: Architecture of a human task simulation system

and to human body performance criteria. We call this part the *task information* system. Using these systems, we can build a human task simulation system as shown in Fig. 6.1. The human task simulation process in this system is as follows:

1. Given a human task, the *task information* system determines a hand goal location (or a hand goal trajectory), a set of geometrical restrictions on the human body, and a set of human body performance criteria that are important to the successful completion of the task.

2. The *human performance* system takes the set of general performance criteria generated by the *task information* system and converts them to functions of the human body posture. These functions are then assembled into one scalar performance criterion function.

3. The *posture control* system takes inputs from both the *task information* system and the *human performance* system. It then computes the desired posture (or posture sequence) that reaches (follows) the given hand goal location (or the hand goal trajectory), satisfies the geometrical constraints, and minimizes the performance criterion function.

With the development of an analytical and a numerical human posture control algorithm, this work has laid a foundation for building fast and efficient human task simulation systems.

Bibliography

[1] Norman I. Badler, Cary B. Phillips, and Bonnie L. Webber. *Simulating Humans: Computer Graphics Animation and Control*. Oxford University Press, 1993.

[2] Jerome Barraquand and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, 10(6):628–649, December 1991.

[3] Mike C. Bartholomew-Biggs. Algorithms for general constrained nonlinear optimization. In Emilio Spedicato, editor, *Algorithms for Continuous Optimization - The state of Art*, pages 169–208, 1993.

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

[5] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Scientific Computing*, 16(5), 1995.

[6] H. Cheng and K. C. Gupta. A study of robot inverse kinematics based upon the solution of differential equations. *Journal of Robotic Systems*, 8(2):159–175, 1991.

[7] Wallace Ching and Norman I. Badler. Fast motion planning for anthropometric figures with many degrees of freedom. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2340–2345. IEEE, Apr 1992.

[8] R. Colbaugh and K. L. Glass. Obstacle avoidance for redundant robots using configuration control. *Journal of Robotic Systems*, 6(6):721–744, 1989.

[9] Paul Cord and Stevan Harnad. *Movement Control*. Cambridge University Press, 1994.

[10] A. S. Deo and I. D. Walker. Adaptive non-linear least squares for inverse kinematics. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, pages 186–193, 1993.

[11] Roger Fletcher. *Practical methods of optimization*. Wiley, 2nd edition, 1987.

[12] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical optimization*. Academic Press, 1981.

[13] K. Glass, R. Colbaugh, D. Lim, and H. Seraji. On-line collision avoidance for redundant manipulators. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, pages 36–43, 1993.

[14] G. Z. Grudic and P. D. Lawrence. Interactive inverse kinematics with manipulator configuration control. *IEEE Transactions On Robotics and Automation*, 9(4):476–483, August 1993.

[15] JOSEPH P. Hale. Virtual reality as a human factors design analysis tool. Macroergonomic application validation and assessment of the space station freedom payload control area. In *Proceedings of the 7th Annual Workshop on Space Operations Applications and Research (SOAR '93)*, page 388, Houston, TX, USA, Aug 1993. NASA Conference Publication, n 3240, pt 1.

[16] Josef Hoschek and Dieter Lasser. *Fundamentals of Computer Aided Geometric Design*. A K Peters, 1993.

[17] Y. K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Computing Surveys*, 24(3):219–291, September 1992.

[18] O. P. Jons, J. C. Ryan, and G. W. Jones. Using virtual environments in the design of ships. *Naval Engineers Journal*, pages 91–106, May 1994.

[19] Yoshihito Koga, Koichi Kondo, James Kuffner, and Jean-Claude Latombe. Planning motions with intentions. In *Proceedings of SIGGRAPH '94*, pages 395–408. ACM SIGGRAPH, ACM Press, July 1994.

[20] Koichi Kondo. Inverse kinematics of a human arm. (in preparation).

[21] James Urey Korein. *A geometric investigation of reach*. MIT Press, Cambridge, Mass., 1985.

[22] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston/Dordrecht/London, 1991.

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

[24] D. Manocha and Y. Zhu. A fast algorithm and system for the inverse kinematics of general serial manipulators. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 3348–3353, 1994.

[25] G. Monheit and N. I. Badler. A kinematic model of the human spine and torso. *IEEE Computer Graphics and Applications*, 11(2):29–38, March 1991.

[26] Yoshihiko Nakamura. *Advanced Robotics : Redundancy and Optimization*. Addison-Wesley Pub, 1991.

[27] Yoshihiko Nakamura and Hideo Hanafusa. Inverse kinematic solutions with singularity robustness for robot manipulator control. *Journal of Dynamic Systems, Measurement and Control*, pages 163–71, September 1986.

[28] M. G. Pandy, F. C. Anderson, and D. G. Hull. A parameter optimization approach for the optimal control of large-scale musculoskeletal systems. *Journal of Biomechanical Engineering*, 114:450–460, November 1992.

[29] M. G. Pandy, F. E. Zajac, E. Sim, and W. S. Levine. An optimal control model for maximum-height human jumping. *Journal of Biomechanics*, 23(12):1185–1198, 1990.

[30] Richard P. Paul. *Robot manipulators : mathematics, programming, and control : the computer control of robot manipulators*. MIT Press, Cambridge, Mass., 1981.

[31] F. G. Pin, P. F. R. Belmans, J. C. Culioli, D. D. Carlson, and F. A. Tulloch. A new solution method for the inverse kinematic joint velocity calculations of redundant manipulators. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 96–102, 1994.

[32] David A. Rosenbaum. *Human Motor Control*. Academic Press, 1991.

[33] S. Sasaki. On numerical techniques for kinematics problems of general serial-link robot manipulators. *Robotica*, 12(P4):309–322, JULY 1994.

[34] H. Seraji. Configuration control of redundant manipulators: Theory and implementation. *IEEE Transactions on Robotics and Automation*, 5(4):472–490, August 1989.

[35] H. Seraji. An on-line approach to coordinated mobility and manipulation. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, pages 28–35, 1993.

[36] H. Seraji and R. Colbaugh. Improved configuration control for redundant robots. *Journal of Robotic Systems*, 7(6):897–928, 1990.

[37] H. Seraji, M. K. Long, and T. S. Lee. Motion control of 7-dof arms: The configuration control approach. *IEEE Transactions On Robotics and Automation*, 9(2):125–138, April 1993.

[38] J. F. Soechting and M. Flanders. Errors in pointing are due to approximations in sensorimotor transformations. *Journal of Neurophysiology*, 62(2):595–608, Aug 1989.

[39] J. F. Soechting and M. Flanders. Sensorimotor representations for pointing to targets in three-dimensional space. *Journal of Neurophysiology*, 62(2):582–594, Aug 1989.

[40] Deepak Tolani and Norman I. Badler. Real-time inverse kinematics of the human arm. *Presence*, to appear.

[41] C. W. Wampler. Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *IEEE Transactions on Systems, Man and Cybernetics*, 16:93–101, 1986.

[42] Andrew Witkin and Michael Kass. Spacetime constraints. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 159–168, August 1988.

[43] Tsuneo Yoshikawa. *Foundations of Robotics : Analysis and Control*. MIT Press, 1990.

[44] Jianmin Zhao and Norman I. Badler. Inverse kinematics positioning using nonlinear-programming for highly articulated figures. *ACM Transactions on Graphics*, 13(4):313–336, 1994.

[45] Xinmin Zhao and Norman I. Badler. Interactive body awareness. *Computer Aided Design*, 26(12):861–867, 1994.

[46] Xinmin Zhao, Deepak Tolani, Bond-Jay Ting, and Norman I. Badler. Simulating human movements using optimal control. In *EUROGRAPHICS CAS'96: 7th Intl. Workshop on Computer Animation and Simulation*, Poitiers, France, August 1996.

[47] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. L-BFGS-B: FORTRAN subroutines for large-scale bound constrained optimization. Technical report, Electrical Engr. and Computer Sci., Northwestern University, December 1994.