function describes positions, not velocities. This approach essentially treats the figure a purely geometric entity rather than a physical one. It does not take into account overtly the figure's mass and inertia, though we will see that it can be used to control the center of mass. Our implementation of inverse kinematics is highly efficient and is capable of controlling a complex figure model with large numbers of constraints.

The approach that we advocate for defining postures is somewhat similar to the energy constraints of Witkin, Fleischer, and Barr [WFB87], although there are some important distinctions. Their approach models connections between objects using energy functions. The energy functions do not measure mechanical energy, but are simply constructed to have zeros at the proper locations and have smooth gradients so that the object can follow the gradient towards the solution. This provides the animation of the solution process. The user interface of Witkin, Fleischer, and Barr's system allows the user to specify these energy functions and turn them on. This causes a sudden increase in energy and thus causes the model to begin descending in the direction of the gradient of the energy function. One drawback of this approach is that the timestep of the iterative process must be sufficiently small to ensure convergence. This is particularly a problem in the case of articulated figures. Witkin, Fleischer, and Barr's formulation of the gradient descent algorithm does not permit jointed mechanisms, so they must model joints as constraints. The joints must have a very steep energy function to ensure that they never come apart. This means that the timestep must be very small, making the system of equations very stiff.

Another problem with Witkin, Fleischer, and Barr's approach, from a higher level point of view, is that the user's notion of time is embedded in the algorithm. The energy function defines the path of the end effectors towards their ultimate destinations because the gradient of the energy function determines the direction of movement. We remove the notion of time from the numerical process and put it in a higher level control scheme. This control scheme decides on discrete locations for the goals for end effector reference points, closely spaced in space and time — the user's notion of time. The inverse kinematics algorithm solves an instantaneous positioning problem. The control scheme has two components, the interactive manipulation previously discussed and the behaviors described in Chapter 4.

The inverse kinematics procedure itself has no notion of time. It uses the joint angles of the **Peabody** figures as the variables of optimization. This means that the search step can be significantly larger, and thus the convergence is faster. The control scheme guides the movement in cartesian space, rather than in the space of the energy function. It is still convenient to interpret the positioning criteria as a potential energy function, but the control scheme ensures that the figure is always very near a state of equilibrium.

Using the example of a human arm reaching for a switch, the technique of Witkin, Fleischer, and Barr would model the reach by defining an energy function with a zero when the hand is on the switch, that is, a constraint for the hand to lie at the switch. If the arm begins by the side of the body,

the arm would see a sudden increase in energy when the constraint becomes active and would immediately begin to move in a way to minimize the energy. This would cause the hand to begin to move towards the switch. The velocity would be controlled through the intensity of the potential energy function.

Using inverse kinematics, the higher level control scheme computes successive locations for the hand, starting at its current location and progressing towards the switch. The number of steps and the spacing between them is not the domain of the inverse kinematics algorithm but of the control scheme. A kinematic constraint describes the position of the hand and the algorithm's parameters at each time step. The inverse kinematics algorithm is invoked at each time step to place the hand at the desired location.

3.2.2 Constraints for Inverse Kinematics

The *Jack* notion of a kinematic *constraint* defines a *goal* coordinate frame, an *end effector* coordinate frame, a *set of joints* which control the end effector, and an *objective function* which measures the distance between the goal and the end effector. The set of joints is usually defined in terms of a *starting joint,* and the joint set then consists of the chain of joints between the starting joint and the end effector. We can think of individual constraints as defining a potential energy, measured by the objective function, much like that of Witkin, Fleischer, and Barr. The constraint is satisfied when the objective function is minimized, although the function need not be zero. The potential energy of several constraints is a weighted sum of the energies from the individual constraints.

The objective function of a constraint has separate position and orientation components. The potential energy of the constraint is a weighted combination of the two, according to the constraint's *position/orientation weight.* The weight may specify one or the other, or a blending of the two. Our inverse kinematics procedure provides the following objective types [ZB89]:

point The point-to-point distance between the end effector and the goal.

line The point-to-line distance between the end effector and a line defined in the goal coordinate frame.

plane The point-to-plane distance between the end effector and a plane defined in the goal coordinate frame.

frame The orientational difference between the end effector frame and the goal frame. This measures all three orientational DOFs.

direction The orientational difference between a vector defined in the end effector frame and a vector defined in the coordinate frame of the goal.

aim A combined position/orientation function designed to "aim" a reference vector in the coordinate frame of the end effector

towards the position of the goal. This is used mostly for eye and camera positioning, but it could also be used, for example, to point a gun. This should never be done in the presence of multiple human figures, of course, particularly children.

3.2.3 Features of Constraints

Each constraint in *Jack* has its own set of joint variables, so the control of each constraint can be localized to particular parts of the figure. Since the joint angles are the variables of optimization, this means that the algorithm operates on chains of joints, ultimately terminated at the root of the figure hierarchy. This implies that the root of the figure remains fixed during the positioning process. This makes the placement of the figure root particularly important. One of the major components of **Peabody** is that the actual definition of a figure does not include the figure root. Instead, the root is a dynamic property which can be changed from time to time. Since the inverse kinematics algorithm operates on chains emanating from the root, the inverse kinematics algorithm cannot change the location of the root.

Our inverse kinematics algorithm works very well provided that it doesn't have to search too far for a solution, although it will converge from any starting point. The farther it has to search, the more likely it is to produce large changes in the joint angles. In geometric terms, this means that the goals should never be too far away from their end effectors, lest the interior segments of the joint chains move too far. This also relieves the problem of getting trapped in a local minimum because hopefully the higher level control strategy which is specifying the goal positions will do so in a way to lead the figure away from such conditions.

The elegance of the potential energy approach, like that of Witkin, Fleischer, and Barr, is that the constraints can be overlapped. This means that it is acceptable to over-constrain the figure. The posture which the algorithm achieves in this case yields the minimum energy state according to the weighting factors between the constraints. This provides a way of resolving the redundancies in a massively redundant figure: use lots of constraints, and don't worry too much about whether the constraints specify conflicting information.

3.2.4 Inverse Kinematics and the Center of Mass

The center of mass of an object is one of its most important landmarks because it defines the focal point for forces and torques acting on it. The center of mass of an articulated figure such as a human figure is particularly significant because its location relative to the feet defines the state of balance. This is critical for human figures, because so many aspects of the movement of a human figure are dictated by the need to maintain balance. The center of mass of is, of course, a dynamic property, but it is possible to manipulate it

in a purely kinematic way and thus produce some of the effects of dynamic simulation without the extra cost.

Methods of Maintaining Balance

One approach to maintaining balance of an articulated figure is to root the figure through its center of mass. The center of mass is a dynamic feature of a figure, so rooting the figure through the center of mass means that each time the figure moves, the center of mass must be recomputed and the figure's location updated so that the center of mass remains at the same global location.

This approach works, but it does not give good control over the elevation of the center of mass, since the center of mass is effectively constrained to a constant elevation as well as location in the horizontal plane. The figure appears to dangle as if suspended from its waist with its feet reaching out for the floor. This is particularly true during an operation in which the center of mass normally goes down, such as bending over. In order for the balance behavior to function naturally, the elevation of the center of mass must be allowed to float up and down as required by the movement of the feet. This is more appropriately handled through a constraint.

Kinematic Constraints on the Center of Mass

Balancing a figure is achieved by constraining the center of mass to remain directly above a point in the support polygon. The constraint designates a single point as the balance point rather than using the entire support polygon because this gives control over the placement of the point within the polygon. This allows the figure's weight to shift side to side or forward and backward, without moving the feet.

Jack associates the center of mass logically with the lower torso region of the human figure, and it uses this as the end effector of the constraint, with the ankle, knee, and hip joints of the dominant leg as the constraint variables. During the constraint satisfaction process at each interactive iteration, the center of mass is not recomputed. Since the center of mass belongs logically to the lower torso, its position relative to the torso remains fixed as the inverse kinematics algorithm positions the ankle, knee, and hip so that the previously computed center of mass point lies above the balance point. There are generally other constraints active at the same time, along with other postural adjustments, so that several parts of the figure assume different postures during the process.

After the constraints are solved, *Jack* recomputes the center of mass. It will generally lie in a different location because of the postural adjustments, indicating that the figure is not balanced as it should be. Therefore, the constraints must be solved again, and the the process repeated until the balance condition is satisfied. In this case the structure of the human figure helps. Most of the postural adjustments take place on the first iteration, so on sub-

sequent iterations the changes in the center of mass relative to the rest of the body are quite minor. *Jack* measures the distance that the center of mass changes from one iteration to the next, and it accepts the posture when the change is below a certain threshold. Although it is difficult to guarantee the convergence theoretically, in practice it seldom takes more than two iterations to achieve balance.

3.2.5 Interactive Methodology

There are several possibilities for overcoming the problems with redundancies and local minima. One is to incorporate more information into the objective function, modeling such factors as strength, comfort, and agent preference (Section 5.3). This is an important addition, although it adds significantly to the computational complexity of the constraint solving procedure. *Jack*'s technique is to provide the positional input to the inverse kinematics algorithm with the 3D direct manipulation system. *Jack* allows the user to interactively "drag" goal positions and have the end effector follow[PZB90]. In this case, the geometric information obtained by the mouse at each iteration of the manipulation process is applied to the goal position of a constraint, and the inverse kinematics algorithm is called to solve the constraints before the graphics windows are redrawn. Alternatively, the user can move a figure which has end effectors constrained to other objects. Each case causes a relative displacement between the end effector and the goal.

Interactive Dragging

This dragging mechanism is a modified version of the basic direct manipulation scheme described in Section 3.1. After selecting the parameters of the constraint, the manipulation procedure works as shown in Figure 3.5. The inverse kinematics procedure is invoked at every iteration during the interactive manipulation.

This is a very effective and efficient tool for manipulation for several reasons. Because of the incremental nature of the interactive manipulation process, the goals never move very far from one iteration to the next, as necessary. The algorithm still suffers from problems of local minima, but since the user can drag the end effector around in space in a well-defined and easy to control way, it is relatively easy to overcome these problems by stretching the figure into temporary intermediate configurations to get one part of the figure positioned correctly, and then dragging the end effector itself into the final desired position. This takes advantage of the user's abilities, because local minima can be easy for the user to see but difficult for the system to detect and avoid.

A common example of this dragging technique involves the elbow. The user may initially position the hand at the proper place in space but then find that the elbow is too high. If this is the case, the user can extend the hand outwards to drag the elbow into the correct general region and then drag the hand back to the proper location.

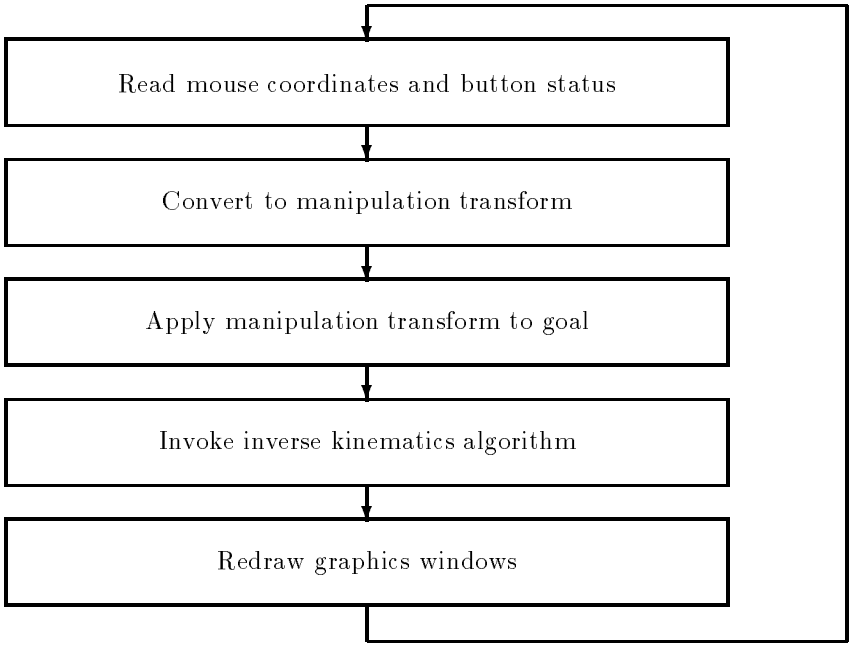

Figure 3.5: Interactive Dragging.

Interactive Twisting

Another effective feature of the direct manipulation interface is the use of orientation constraints, particularly the weighted combination of position and orientation. In this case, the orientation of the goal is significant as well as the position, so the user may manipulate segments in the interior of the joint chain by twisting the orientation of the goal and thus the end effector. This is especially helpful because of the difficulty the user encounters in visualizing and numerically describing rotations which will achieve a desired orientation. The above example of the elbow position may be handled this way, too. By twisting the desired orientation of the hand, the interior of the arm can be rotated up and down while the hand remains in the same location. This achieves in real-time a generalization of the "elbow circle" positioning scheme implemented by Korein [Kor85].

This raises an interface issue concerning the relationship between the actual orientation of the end effector coordinate frame and the manipulation transform. The manipulation technique described in Section 3.1 allows the user to translate and rotate a 6D quantity which now guides the position and orientation of the end effector. We noted, however, that this technique is not so good at choreographing smooth movements through space. The movement trajectory generated by the technique consists of intermittent seg-

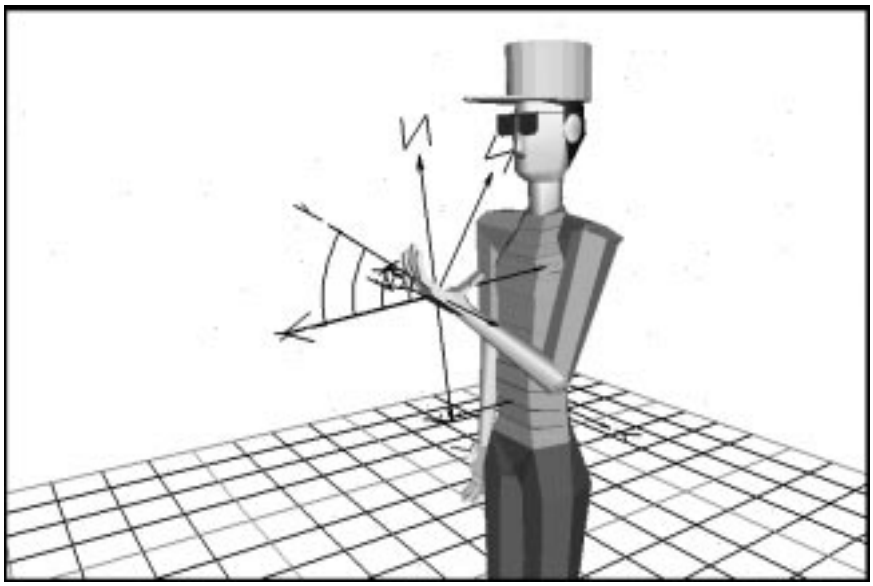

Figure 3.6: The Orientation Constraint Icon.

ments of straight-line translations and rotations. As the user translates the manipulation transform, its orientation remains fixed, and vice versa. Is this appropriate behavior for the end effector as well?

If the end effector is sensitive to orientation, then translating the manipulation transform means that the end effector will translate *but will try keep the same global orientation.* Typically, the user can quickly arrive at a goal position for the end effector which is not achievable.

Positional differences are easy to visualize; orientational differences are not. It is easy to manipulate a positional goal which is outside of the reachable space of the end effector. We can intuitively think of a spring or rubber band pulling the end effector towards the goal. Orientational differences are much harder to visualize. Even though it may be easy to conceptualize "rotational springs," in practice is it very difficult to apply that intuition to the geometric model. If the goal is very far outside of the reachable space of the end effector along *angular dimensions*, all correspondence between the orientation of the goal and the end effector gets quickly lost. *Jack* illustrates orientational differences through rotation *fans,* shown in Figure 3.6, which are icons to illustrate how an object must rotate to achieve a desired orientation, but no amount of graphical feedback can help when the differences are large. Therefore, it is absolutely essential that the orientation of the goal – the manipulation transform — not deviate too far from the end effector.

Jack solves this problem through an orientation *offset* to the goal which can be adjusted during the manipulation process. This offset is a relative transform which is applied to the manipulation transform to rotate it into the

true orientation of the goal as supplied to the inverse kinematics algorithm. The end effector dragging mechanism resets this offset each time a translation or rotation is completed during the manipulation process, that is, each time a mouse button comes up and the movement stops. This means that each time the user begins to rotate or translate the goal, the manipulation transform starts out from the current orientation of the end effector. This prevents the user from getting lost in the orientation difference.

This simulates the effect of a spring-loaded crank which applies a torque to an object, but only as long as the user holds down the mouse button. When the mouse button comes up, the torque disappears so that it doesn't continue to have a undesirable effect. This lets the user control the goal through a ratcheting technique of applying short bursts of rotation.

Manipulation with Constraints

The nature of the 3D direct manipulation mechanism allows the user to interactively manipulate only a single element at a time, although most positioning tasks involve several parts of the figure, such as both feet, both hands, etc. In addition to interactively dragging a single end effector, there may be any number of other kinematic constraints. These constraints are persistent relationships to be enforced as the figure is manipulated using any of the other manipulation tools. By first defining multiple constraints and then manipulating the figure, either directly or with the dragging mechanism, it is possible to control the figure's posture in a complex way.

This mechanism involves another slight modification to the direct manipulation loop, shown in Figure 3.7. Step #4 may cause the end effectors to move away from their goal positions. The inverse kinematics algorithm in step #5 repositions the joints so the goals are satisfied.

3.3 Inverse Kinematic Positioning

[1]Having modeled the articulated figure with segments and joints we need to deal with the issue of how to manipulate it. There are two basic problems: given all joint angles, how to compute the spatial configuration and, conversely, given a certain posture, what values should be assigned to joint angles. The first problem, forward kinematics, is simply a matter of straightforward transformation matrix multiplication [FvDFH90]. The second, inverse kinematic problem, is much harder to solve.

Inverse kinematics is extremely important in computer animation, since the spatial appearance, rather than the joint angles, interest an animator. A good interactive manipulation environment such as *Jack* may simply hide the numerical values of the joint angles from the user. In such an environment, where the user is concerned only with spatial configuration, the joint angles can become merely an internal representation. The transformation from the

[1] Jianmin Zhao.

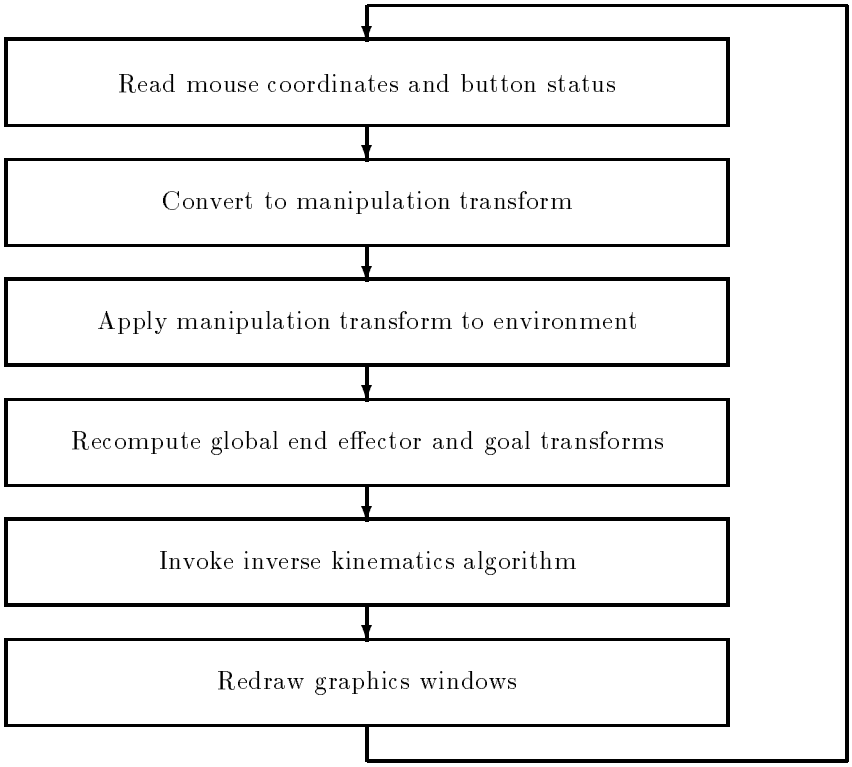

Figure 3.7: Manipulation with Constraints.

spatial configuration into the joint angles is carried out by the inverse kinematic computation so an efficient implementation is most desirable.

Inverse kinematics for determining mechanism motion is a common technique in mechanical engineering, particularly in robot research [Pau81]. In robots, however, redundant DOFs are usually avoided. Moreover, the computation is usually carried out on particular linkage systems [KH83, KTV+90]. In contrast, an interesting object in the computer animation domain – the human body – has many redundant DOFs when viewed as a kinematic mechanism. Therefore a means for specifying and solving underconstrained positioning of tree-like articulated figures is needed.

We first studied methods for kinematic chain positioning, especially in the context of joint limits and redundant DOFs [KB82, Kor85]. Later we tried position constraints to specify spatial configurations of articulated figures [BMW87]. A simple recursive solver computed joint angles of articulated figures satisfying multiple point-to-point position constraints.

About the same time, Girard and Maciejewski used the pseudo-inverse of

the Jacobian matrix to solve spatial constraints [GM85]. The main formula is

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

where $\Delta\theta$ is the increment of the joint angle vector, $\Delta\mathbf{r}$ is the increment of the spatial vector and J^{+} is the pseudo-inverse of the Jacobian $\partial\mathbf{r}/\partial\theta$. For a large step size the method is actually the well-known Newton-Raphson method, which is not globally convergent and often needs some special handling (e.g., hybrid methods [Pow70]). On the other hand, for a sufficiently small step size, as Girard and Maciejewski suggested, excessive iterations are required. The inverse operation is normally very expensive; moreover, they did not deal with joint limits.

Witkin *et al* used energy constraints for positioning [WFB87]. The energy function they adopted is the sum of all constraints including ones for position and orientation. Constraints are satisfied if and only if the energy function is zero. Their method of solving the constraint is to integrate the differential equation:

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

where θ is the parameter (e.g., joint angle) vector, E is the energy function of θ, and ∇ is the gradient operator. Clearly, if $\theta(t)$ is the integral with some initial condition, $E(\theta(t))$ monotonically decreases with time t. This integral not only gives a final configuration which satisfies the constraint in terms of θ, but also a possible motion from the initial configuration which is driven by the conservative force derived from the gradient of the energy function.

Instead of associating energy functions with constraints, Barzel and Barr introduced deviation functions such that a constraint is met if and only if the deviation function vanishes (= 0) [BB88]. They have presented various constraints, such as point-to-point, point-to-nail, etc., and their associated deviation functions. A set of dynamic differential equations are constructed to control the way by which the deviations vanish (e.g., exponentially in a certain amount of time). Constraint forces are solved which, along with other external forces, drive geometric models to achieve constraints. They dealt with rigid bodies which may be related by various constraints. Witkin and Welch [WW90] used a dynamic method on nonrigid bodies. In all these approaches articulated figure joints would be considered as point-to-point constraints, which are added to the system as algebraic equations. It is not unusual to have several dozen joints in an highly articulated figure, which would add to the number of constraint equations appreciably. Besides, a joint is meant to be an absolute constraint. In another words, it should not compete with other constraints which relate a point in a segment of the figure with a point in the space. This competition often gives rise to numerical instability. So these methods seem inappropriate to the highly articulated figure.

Although the energy or dynamics methods may be used to solve the spatial constraint problem, they are also concerned with computing plausible (dynamically-valid) paths. For articulated figure positioning, the additional work involved in path determination is not so critical. So, we first focus on

a more elementary version of the problem — to position the articulated figure into desired pose — hoping that the solution can be found much faster and more robustly, and joint angles limits can be dealt with effectively. We will see later in Chapter 5 how more natural motions may be generated by considering human strength models and collision avoidance.

3.3.1 Constraints as a Nonlinear Programming Problem

A spatial constraint involves two parts. The part on the figure is called the end-effector and its counterpart in the space is called the goal. A constraint is a demand that the end-effector be placed at the goal. Since a goal always implies a related end-effector, we may sometimes take goals as synonymous for constraints.

As is done with energy constraints, we create a function with each goal such that its value, when applied to the end effector, represents the "distance" of the end-effector from the goal. This distance need not be only Euclidean distance. Let us call this function "potential," because it is a scalar function of spatial position or vectors. The vector field generated by the negation of the gradient of the function could be interpreted as a force field toward the goal. Depending on the goal types, an end-effector on a segment can be a point, a vector, a set of two vectors, or a combination of them, in the local coordinate system of the segment. Let P denote the potential function associated with a goal and $\mathbf{e}$ the "location" of the end-effector. The constraint is satisfied if and only if $P(\mathbf{e})$ vanishes. Clearly the end-effector is a function of the vector of all joint angles.

A single constraint may not be sufficient to specify a pose. For example, in addition to the position of the hand, the placement of the elbow is often desired. In positioning two arms, we may need to keep the torso in some orientation. The function associated with these conjunctively combined goals is defined as a weighted sum:

$$G(\theta) = \sum_{i=1}^{m} w_i G_i(\theta) \tag{3.1}$$

where m is the number of goals combined, and

$$G_i(\theta) = P_i(\mathbf{e}_i(\theta)) \tag{3.2}$$

where the subscript i refers to the ith goal, and w_is are weights on respective constraints.

Sometimes, goals may need be combined disjunctively. In other words, the combined goal is achieved if and only if either constituent goal is achieved. For example, the interior of a convex polyhedron is the conjunction of inner half-spaces defined by its surrounding faces, while the exterior is the disjunction of those opposite half-spaces. Obstacle avoidance is a situation where constraining an end-effector to the outside of a volume is useful. The function

associated with the disjunctively combined goal is defined as

$$G(\theta) = \min_{i \in \{1,\ldots m\}} G_i(\theta) \tag{3.3}$$

By definition of the function G, the overall constraint is met if and only if G vanishes. Due to the physical constraint of the figure and the redundancy of DOFs, there may not be θ such that $G(\theta) = 0$, or there may be many such θ's. Therefore we attempt to find a θ, subject to constraints on joint angles, which minimizes the function G. Although constraints on joint angles may be complicated, linear constraints suffice in most situations. Typical constraints are represented as a lower limit and an upper limit for each joint angle. Therefore we formulate our problem as a nonlinear programming problem subject to linear constraints, i.e.,

$$\begin{cases} \min G(\theta) \\ s.t. \quad \mathbf{a}_i^T \theta = b_i, i = 1, 2, \ldots, l \\ \qquad \mathbf{a}_i^T \theta \leq b_i, i = l+1, l+2, \ldots, k \end{cases} \tag{3.4}$$

where $\mathbf{a}_i, i = 1, 2, \ldots, k$ are column vectors whose dimension is the total number of DOFs. The equality constraints allow for linear relations among the joint angles. The lower limit l^i and upper limit u^i on θ^i, the ith joint angle, contribute to the set of inequality constraints on θ :

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

3.3.2 Solving the Nonlinear Programming Problem

There are many algorithms to solve problem (3.4). One efficient algorithm is based on Davidon's variable metric method with BFGS (Broyden, Fletcher, Goldfarb, Shanno) rank-two update formula [Fle70, Gol70, Sha70]. Rosen's projection method is used to handle the linear constraints on variables [Ros60, Gol69]. No matter what algorithm we choose, efficiency requires computation of the gradient as well as the value of the objective function when θ is given. So first we develop an efficient way to compute the gradient of, as well as the value of, the objective function $G(\theta)$ for figures with tree-like data structure and various sorts of goals.

Single Constraints

Since the human figure has a tree-like structure, an end-effector of a constraint depends only on those joints which lie along the path from the root of the figure tree to the distal segment containing the end-effector [BMW87]. Call this path the *constraint chain*. The constraint chain is illustrated in Figure 3.8. For simplicity, we assume each joint in Figure 3.8 has only one DOF. A joint with multiple DOFs can be decomposed conceptually into several joints with each having one DOF, i.e., zero distance from one joint to another is allowed.

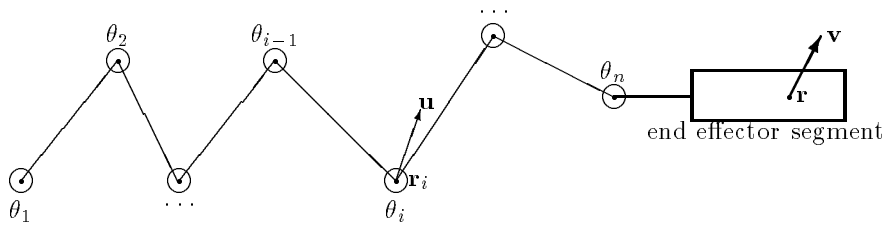

Figure 3.8: Constraint Chain.

The length of the constraint chain is the total number of joints along the chain; in Figure 3.8, it is n.

This module is to compute $G_i(\theta)$, as defined in equation (3.2) and its gradient for a single constraint chain and its corresponding goal. In this section, we only consider a single chain, so the subscript i is dropped.

Notice that

$$G(\theta) = P(\mathbf{e}(\theta)) \tag{3.5}$$

and

$$\begin{aligned} \mathbf{g}(\theta) &= \nabla_\theta G \\ &= (\frac{\partial \mathbf{e}}{\partial \theta})^T \nabla_{\mathbf{e}} P \end{aligned} \tag{3.6}$$

where $\frac{\partial \mathbf{e}}{\partial \theta}$ is the Jacobian matrix of the vector function $\mathbf{e}(\theta)$, i.e.,

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

These expressions suggest that end-effectors and goals can be abstracted as functions $\mathbf{e}(\cdot)$ and $P(\cdot)$ which, by pair, constitute constraints. The range of the function $\mathbf{e}$ depends on the type of the constraint, but it must be equal to the domain of its respective P function, which measures the distance of the current end-effector to the goal. For the sake of numerical efficiency, we require that those functions be differentiable.

By virtue of equations (3.5) and (3.6), two sub-modules, end-effector and goal modules, can be built separately. Note that the goal potential P does not depend on the figure model. Only the end-effector function $\mathbf{e}$ does.

End-effectors

This module is the only one which depends on the model of the articulated figure. It turns out that it is easy to compute $\mathbf{e}$ and $\frac{\partial \mathbf{e}}{\partial \theta}$, under our assumption of rigidity of segments, for many useful types of constraint.

We only consider constraints where end-effectors are points (for positions) or vectors (for orientations) on distal segments. So $\mathbf{e}$ consists of either a position vector $\mathbf{r}$, one or two unit vectors attached on the distal segment $\mathbf{v}$'s, or a combination of them (see Figure 3.8). Because all the joints of the human body are revolute joints, we discuss here only revolute joints. (Translational joints are simpler and can be treated similarly.)

Let the ith joint angle along the chain be θ^i, the axis of this joint be $\mathbf{u}$ (a unit vector), and the position vector of the ith joint be $\mathbf{r}_i$. The vectors $\mathbf{r}$ and $\mathbf{v}$ can be easily computed with cascaded multiplication of transformation matrices. The derivative can be computed as follows [Whi72]:

$$\frac{\partial \mathbf{r}}{\partial \theta^i} = \mathbf{u} \times (\mathbf{r} - \mathbf{r}_i) \tag{3.7}$$

$$\frac{\partial \mathbf{v}}{\partial \theta^i} = \mathbf{u} \times \mathbf{v} \quad . \tag{3.8}$$

These formulas are enough for the types of goals we have considered. The particular forms of $\mathbf{e}$ will be explained with particular goals, since they must match the arguments of the goal potential $P(\cdot)$.

Goal Types

We have implemented several useful, as well as simple, types of goals (or constraints).

Position Goal. The goal is a point $\mathbf{p}$ in 3D space. The end-effector is, correspondingly, a point $\mathbf{r}$ which sits on the distal segment of the constraint chain, but it is not necessarily a leaf segment in the figure tree (see Figure 3.8). The potential function is:

$$P(\mathbf{r}) = (\mathbf{p} - \mathbf{r})^2 \tag{3.9}$$

where p is the parameter of the function, and the gradient is:

$$\nabla_{\mathbf{r}} P(\mathbf{r}) = 2(\mathbf{r} - \mathbf{p}) \quad . \tag{3.10}$$

Orientation Goal. The orientation in the space is determined by two orthonormal vectors. So the goal is defined by a pair of orthonormal vectors, say,

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

Accordingly, the end effector is a pair of orthonormal vectors

$$\{\mathbf{x}_e, \mathbf{y}_e\}$$

attached on the distal segment of the constraint chain.

The potential function could be:

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

In combination with a positional goal, this function implies that one length unit is as important as about one radian in angle. To enforce one length unit compatible with d degrees in angle, we need to multiply the previous P by c_d such that

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

i. e. ,

$$c_d = 360/(2\pi d) \quad . \tag{3.11}$$

To be more general, our potential function is then

$$P(\mathbf{x}_e, \mathbf{y}_e) = c_{dx}^2(\mathbf{x}_g - \mathbf{x}_e)^2 + c_{dy}^2(\mathbf{y}_g - \mathbf{y}_e)^2 \quad . \tag{3.12}$$

The gradient is

$$\nabla_{\mathbf{x}_e} P(\mathbf{x}_e, \mathbf{y}_e) = 2c_{dx}^2(\mathbf{x}_e - \mathbf{x}_g) \tag{3.13}$$

$$\nabla_{\mathbf{y}_e} P(\mathbf{x}_e, \mathbf{y}_e) = 2c_{dy}^2(\mathbf{y}_e - \mathbf{y}_g) \quad . \tag{3.14}$$

Any direction, such as y, could be suppressed by setting c_{dy} to 0. This is useful, for example, to constrain a person holding a cup of water while attaining other constraints.

Position/Orientation Goals. Position and orientation goals can be treated separately, but sometimes it is convenient to combine them together as a single goal. The potential function for position/orientation goal is just the weighted sum of respectively goals:

$$P(\mathbf{r}, \mathbf{x}_e, \mathbf{y}_e) = w_p(\mathbf{p} - \mathbf{r})^2 + w_o c_{dx}^2(\mathbf{x}_g - \mathbf{x}_e)^2 + w_o c_{dy}^2(\mathbf{y}_g - \mathbf{y}_e)^2 \tag{3.15}$$

where w_p and w_o are weights put on position and orientation respectively such that

$$w_p + w_o = 1 \quad .$$

The gradients $\nabla_{\mathbf{r}} P, \nabla_{\mathbf{x}_e} P$ and $\nabla_{\mathbf{y}_e} P$ are obvious from above.

Aiming-at Goals. The goal is a point $\mathbf{p}$ in the space, but the end-effector is a vector $\mathbf{v}$ attached to the distal segment at $\mathbf{r}$ (see Figure 3.8). The goal is attained if the vector $\mathbf{v}$ points toward the point $\mathbf{p}$. This is useful, for example, when we want to make the head face toward a certain point.

The potential function is:

$$P(\mathbf{r}, \mathbf{v}) = c_d^2\left(\frac{\mathbf{p} - \mathbf{r}}{\|\mathbf{p} - \mathbf{r}\|} - \mathbf{v}\right)^2 \tag{3.16}$$

where c_d is defined in (3.11) and $\|\cdot\|$ denotes the norm operation. The gradient is:

$$\nabla_{\mathbf{r}} P(\mathbf{r}, \mathbf{v}) = 2c_d^2(\|\mathbf{p} - \mathbf{r}\|^2 \mathbf{v} - (\mathbf{p} - \mathbf{r}) \cdot \mathbf{v}\,(\mathbf{p} - \mathbf{r}))/\|\mathbf{p} - \mathbf{r}\|^3 \tag{3.17}$$

$$\nabla_{\mathbf{v}} P(\mathbf{r}, \mathbf{v}) = -2c_d^2\left(\frac{\mathbf{p} - \mathbf{r}}{\|\mathbf{p} - \mathbf{r}\|} - \mathbf{v}\right) \quad . \tag{3.18}$$

Line Goals. The goal is a line and the end-effector is a point $\mathbf{r}$. The goal is to force the point to go along the line. Let the line be defined by point $\mathbf{p}$ and unit vector ν such that the parametric equation of the line is

$$\mathbf{p} + t\nu \quad .$$

The potential function is:

$$P(\mathbf{r}) = ((\mathbf{p} - \mathbf{r}) - (\mathbf{p} - \mathbf{r}) \cdot \nu \, \nu)^2 \tag{3.19}$$

and the gradient is:

$$\nabla_{\mathbf{r}} P(\mathbf{r}) = 2(\nu \cdot (\mathbf{p} - \mathbf{r}) \, \nu - (\mathbf{p} - \mathbf{r})) \quad . \tag{3.20}$$

Plane Goals. The goal is a plane and the end-effector is a point $\mathbf{r}$. The point must be forced to lie on the plane. Let a point on the plane be $\mathbf{p}$ and the normal of the plane be ν. The potential function is:

$$P(\mathbf{r}) = ((\mathbf{p} - \mathbf{r}) \cdot \nu)^2 \tag{3.21}$$

and the gradient is:

$$\nabla_{\mathbf{r}} P(\mathbf{r}) = -2\nu \cdot (\mathbf{p} - \mathbf{r}) \, \nu \quad . \tag{3.22}$$

Half-space Goals. The goal is one side of a plane and the end-effector is a point $\mathbf{r}$. The point is constrained to lie to one side of the plane. Let a point on the plane be $\mathbf{p}$ and the normal of the plane be ν which points to the half space the goal defines. The potential function is:

$$P(\mathbf{r}) = \begin{cases} 0 & \text{if } (\mathbf{p} - \mathbf{r}) \cdot \nu < 0 \\ ((\mathbf{p} - \mathbf{r}) \cdot \nu)^2 & \text{otherwise} \end{cases} \tag{3.23}$$

and the gradient is:

$$\nabla_{\mathbf{r}} P(\mathbf{r}) = \begin{cases} 0 & \text{if } (\mathbf{p} - \mathbf{r}) \cdot \nu < 0 \\ -2\nu \cdot (\mathbf{p} - \mathbf{r}) \, \nu & \text{otherwise} \end{cases} \quad . \tag{3.24}$$

As the number of joint angles n along the chain grows, the computational complexity of G and $\mathbf{g}$ is linear for the listed goal types, since the end-effector module needs $O(n)$ time and the goal module needs $O(1)$ time.

3.3.3 Assembling Multiple Constraints

In the single constraint module, we consider only those joint angles which lie on the constraint chain. To make our constraint system useful, multiple constraints would be combined to one constraint, as in equations (3.1) or (3.3). The arguments in these formulas, the vector θ, may involve joint angles along many paths in the figure tree. In other words, with respect to this

θ the gradient of G_i in (3.1) or (3.3) contains many zeros. For the sake of computational efficiency and program modularity, we choose not to pass the overall index of joint angles to the single constraint module.

Suppose there are m constraints. The ith constraint involves n_i joint angles, whereas the combined constraint involves n joint angles. Notice that this n is not simply the summation of n_is, since one joint angle may be involved in several chains. The relationship between the index used in the ith constraint chain and the overall index is represented by the mapping

$$M_i : \{1, 2, \ldots, n_i\} \longrightarrow \{1, 2, \ldots, n\} \quad ; \tag{3.25}$$

the jth joint angle in the ith constraint chain corresponds to the $M_i(j)$th joint angle in the overall index system.

Let θ_i denote the vector formed by the joint angles along the ith constraint chain. For disjunctively combined goals with G defined in (3.3) , one can always focus on the goal with minimal $G_i(\theta_i)$ at each iteration. So this is nothing but a collection of goals.

This module is designed for conjunctively combined goals with G defined in (3.1). The evaluation of G is simply a summation of the individual G_i. To compute the overall gradient

$$\begin{aligned} g &= \begin{pmatrix} g^1 & g^2 & \cdots & g^n \end{pmatrix}^T \\ &= \nabla_\theta G(\theta) \end{aligned}$$

by using the outputs of the single constraint module

$$\begin{aligned} g_i(\theta_i) &= \begin{pmatrix} g_i^1 & g_i^2 & \cdots & g_i^{n_i} \end{pmatrix}^T \\ &= \nabla_{\theta_i} G_i(\theta_i) \quad , \end{aligned}$$

we only need to do

1. $g^j \leftarrow 0$, for $j = 1, 2, \ldots, n$
2. For $i = 1$ to m do
   $g^{M_i(j)} \leftarrow g^{M_i(j)} + w_i g_i^j$, for $j = 1, 2, \ldots, n_i$.

From above one can see that all the G_i and their gradients g_i can be computed in parallel.

We assumed in Section 3.3.2 that the constraint chain went from the root of the figure tree to the distal segment. It is possible and sometimes useful that the chain go from a specified joint which is nearer to the root than the distal segment is. Then we must take care of those joints which affect an end-effector, but are not in the constraint chain. For example, suppose that the torso is the root of the figure, one constraint chain is from the right shoulder to the right hand, and the other is from the torso to the left hand. Although the torso is not assigned to the end-effector right hand, it will affect the right hand when it moves the left one. The system should add joints from the torso to the left shoulder to the constraint chain for the right hand [BOK80].

Constraints may exist in a hierarchy. With multiple constraints, there may be one (or more) constraint(s) which consists of several constraints disjunctively combined. In general, a logic formula which contains conjunctive and disjunctive connectors can be implemented. For example, the exterior of a convex polyhedron can be defined as the union of the outside half-space of all of its surrounding faces. This can be used to avoid obstacles: we may want the hand to get somewhere while keeping the elbow away from the obstacle. This does not solve *segment* obstacle avoidance, however.

Joints can be locked or added to the constraint chain for a particular task.

3.3.4 Stiffness of Individual Degrees of Freedom

A nonlinear programming algorithm which utilizes gradient information has a property that, given the same termination tolerance, a variable would undergo more displacement if the value of the objective function changes more due to the unit increment of that variable. This property can be used to control the stiffness of each DOF by assigning a scaling factor to each joint angle. These scaling factors change the unit of each joint angle.

For example, if the standard unit of joint angles is a radian, and 0.5 is assigned to a particular joint angle, which is to take half of the radian as the unit of that joint angle, then this joint angle will be more reluctant to move. One unit of change in this joint angle will have half of the effect on the end-effector as it would without that scaling factor. This makes the effect of other joint angles more apparent.

3.3.5 An Example

Jack contains an implementation of this multiple spatial constraint solver as a basic tool for the interactive manipulation of articulated figures. Constraints can be of any types listed above, or can be sets of simple constraints disjunctively combined. Users may create various constraints, any of which can be moved interactively, and the achieving configuration is solved and observed in real-time.

The pose in Figure 3.9 is achieved by using 6 constraints. Two Position/Orientation goals are used for two hands to hold the tube, where one direction of the orientation is suppressed so that only normals of the hand and the tube are aligned. Two Plane goals are used to constrain two elbows on two side planes. To have the person look down towards the end of the tube, we used two goals – a Line goal was used to constrain the view point on the central axis of the tube; an Aiming-at goal was used to point the view vector towards the end of the tube. In all, 22 DOFs are involved. To encourage forward bending, we set the rigidities of the lateral bending and axial rotation of the torso to a mid-range (0.5) value. Starting from an upright neutral position and moving from the waist, the solution took only 2 seconds on a Silicon Graphics Personal Iris (4D-25TG), not one of the faster machines of its type.

Figure 3.9: Looking Down Towards the End of the Tube.

3.4 Reachable Spaces

[2]The *workspace* is the volume or space encompassing all points that a reference point on the hand (or the end effector) traces as all the joints move through their respective ranges of motion [KW81]. An articulated chain is a series of links connected with either revolute or prismatic joints such as a robotic manipulator or a human limb. Visualizing the 3D workspace for articulated chains has applications in a variety of fields: in computer graphics and artificial intelligence systems that generate plans for approaching and grasping an object in complex environments, in CAD systems to design the interior layout of cars, vehicles, or space shuttles, in the evolving areas of telerobots and parallel manipulators, in the coordination of different manipulators to perform certain tasks, and, finally, in ergonomic studies to help understanding the effects of body size, joint limits, and limb length on the workspace volume.

Previous workspace algorithms [Kum80, Sug81, GR82, TS83, TS81, Tsa86, YL83, Vij85, Muj87] are only capable of displaying 2D workspace cross-

[2]Tarek Alameldin.

sections. This is not adequate for redundant (more than 6 DOFs) manipulators with joint limits. We describe how to compute the 3D workspace for redundant articulated chains with joint limits.

The first efforts to compute the manipulator workspace, based on its kinematic geometry, started in the mid 1970's [Rot75, Sug81]. The first result was that the extreme distance line between a chosen point on the first joint axis and the center point of the hand/end effector (extreme reach) intersects all intermediate joint axes of rotation. This is not valid, however, if any joint has limits, any intermediate joint axis is parallel to the extreme distance line, or two joint axes intersect.

Kumar and Waldron [KW81] presented another algorithm for the manipulator's workspace. In their analysis, an imaginary force is applied to the reference point at the end effector in order to achieve the maximum extension in the direction of the applied force. The manipulator reaches its maximum extension when the force's line of action intersects all joint axes of rotation (since the moment of the force about each axis of rotation must be zero). Every joint of the manipulator can settle in either of two possible positions under the force action. Hence, this algorithm results in 2^{n-1} different sets of joint variables for a manipulator of n joints in the direction of the applied force. Each set of joint variables results in a point on the workspace boundary. The concept of stable and unstable equilibrium is used to select the set of joint variables that result in the maximum extension in the force direction. This algorithm is used to generate a shell of points which lie on the workspace boundary by varying the direction of the applied force over a unit sphere. This algorithm has exponential time complexity and deals only with those manipulators that have ideal revolute joints.

Tsai and Soni [TS83] developed another algorithm to plot the contour of the workspace on an arbitrarily specified plane for a manipulator with n revolute joints. The robot hand is moved to the specified plane, then the tip of the hand is moved on the plane until it hits the workspace boundary, and finally the workspace boundary is traced by moving the hand from one position to its neighbor. Each of these three subproblems is formulated as a linear programming problem with some constraints and bounded variables (to account for the joint limits). Accordingly, this algorithm is just a 2D workspace cross-section computation and, moreover, has excessive computational cost.

Yang [YL83] and Lee [LY83] presented algorithms to detect the existence of holes and voids in the manipulator's workspace. A workspace is said to have a hole if there exist at least one straight line which is surrounded by the workspace yet without making contact with it. The hole in a donut is a simple example for the above definition. A workspace is said to have a void if there exist a closed region R, buried within the reachable workspace, such that all points inside the bounding surface of R are not reached by the manipulator. Gupta [Gup86, GR82] classified voids into two different types. The first one, called *central*, occurs around the first axis of rotation and is like the core of an apple. The second type, called *toroidal* or *noncentral*, occurs within the reachable workspace and is like a hollow ring. He [Gup86] also

presented qualitative reasoning about the transformation of holes to voids and vice versa. Both the qualitative method developed by Gupta [Gup86] and the analytical one developed by Yang and Lee [YL83, LY83] are based on mapping the workspace from the distal link to the proximal one and studying the relationship between the generated workspace and the new axis of rotation.

Tsai [Tsa86] presented another algorithm, based on the theory of reciprocal screws. In contrast to the above algorithms that only compute workspace points, this algorithm traces the 2D workspace boundary for a given manipulator. The use of reciprocal screw theory has made computing piecewise continuous boundary that consists of straight line segments and circular arcs possible. The manipulator's workspace is computed by performing the union operation on all the workspaces of the manipulator's aspects. An aspect of a robot is interpreted as a set of joint variables such that the manipulator can reach points inside the workspace at one configuration without hitting a joint limit [Tsa86]. The computed workspace has interior surfaces which are the boundaries of aspects. This algorithm is limited to manipulators which do not have holes or voids in their workspaces.

Korein [Kor85] created conservative approximations to 3D reach volumes by taking polyhedral unions of reach polyhedra, working along an articulated chain from the distal joint inwards. The major drawbacks of his approach are the high computational cost and numerical sensitivity of the polyhedral unions which are very difficult to perform once they become many-sided.

3.4.1 Workspace Point Computation Module

The purpose of this module is to compute a suitably dense set of workspace points. The inputs to this module are a chain of linkages with a proximal and distal end, the joint limits associated with each DOF and the desired resolutions in the x, y and z directions (res_x, res_y, res_z) for the end effector position.

We classify the algorithms that can be used to implement this module as follows:

1. Algorithms based on forward kinematics. The basic idea is to generate end effector positions by cycling each DOF through some number of discrete angles (if revolute) or distances (if prismatic).

2. Algorithms based on nonlinear programming. Here a collection of points in space is provided as targets for the end effector and the linkage attempts to solve for a satisfying posture.

3. Algorithms based on force application at the end effector. A series of force directions is used to pull the chain to its maximum extension.

Each class is better than the others for some applications. Direct kinematics algorithms lend themselves easily to volume visualization applications since they require less time and space than the other algorithms. It is difficult,

however, to determine the adequate density (the number of points to be generated) that would compute a workspace with the given resolution (res_x, res_y, and res_z) . Hence, direct kinematics based algorithms cannot be used alone to compute the workspace volume since they are not guaranteed to compute all the reachable points. The resolution values divide the space into cells of dimensions $res_x \times res_y \times res_z$. A cell is marked with one if it contains a workspace point. A cell marked with zero does not necessarily mean that it is unreachable since the direct kinematics based algorithm might not have computed enough workspace points. This limitation is serious especially if the application requires surface visualization which use edge detection algorithm as will be described in the next section. On the other hand, algorithms based on nonlinear programming are more appropriate for applications that only require computing the envelope of the reachable workspace since these algorithms compute only points that lie on the workspace envelope. However, the cost of computing each point by the nonlinear programming based algorithms is higher than the cost using forward kinematics based algorithms. Finally, nonlinear programming based algorithms are more appropriate for applications that require partial surface computation in predetermined directions. They can also be used in volume visualization applications by dividing the space into voxels (of dimension $res_x \times res_y \times res_z$) and using the inverse kinematics algorithm to determine whether the cell is occupied or not. However, this operation is very costly and does not guarantee the correct result since the nonlinear programming algorithms do not necessarily return the global maximum or minimum (they might stop at local ones). Algorithms that are based on force application are very costly since they require exponential time to compute each point. All these algorithms can be hybridized to compute better quality volumes or surfaces [Ala91, ABS90].

3.4.2 Workspace Visualization

We believe that either surface based techniques or voxel based techniques can be used in workspace visualization depending on the application type. If the application goal is to compute the workspace boundary in order to find the intersection with other psurf objects in the environment, surface based techniques can be used. On the other hand, binary voxel based techniques lend themselves easily to applications that require computing cut-away views, changing the view point, finding the union or the intersection with other objects that are represented by voxels, and trading off computation time against image quality.

This module constructs a surface that encompasses the workspace points that were computed by the workspace point generation module. We have developed an algorithm that accepts the workspace contours computed by the direct kinematics algorithm. The algorithm can be summarized in four steps:

1. Region Filling. This step involves determining the number of regions in a given workspace contour. The number of holes and voids in the

given workspace contour can be determined. Region filling algorithms are a common graphics utility and are widely used in paint programs [Sha80, FB85, Fis90]. A region is a collection of pixels. There are two types of regions: 4-connected and 8-connected. A region is 4-connected if adjacent pixels share a horizontal or vertical edge. A region is 8-connected if adjacent pixels share an edge or a corner. Region filling algorithms start with a given seed point (x, y) and set this pixel and all of its neighbors with the same pixel value to a new pixel value. A good region-filling algorithm is one that reads as few pixels as possible. The algorithm computes the number of regions in a given workspace cross-section (contour). We search the contour for a reachable workspace cell (marked with 1) and use it as a seed point. The set of all cells connected to the seed point comprise a reachable region. The region filling algorithm sets those cells to a new value that greater than 2. The region filling algorithm is called as many times as necessary in order to set different regions with unique values.

2. Boundary Detection. The purpose of this step is to compute the boundary of different regions in a workspace cross-section. This is done by testing the neighbors of each cell in the workspace cross-section. An array element of the workspace cross-section is considered a boundary cell if it has a different value from its neighbor.

3. Contour Tracing. This step computes the edges that connect the boundary points for a given region.

4. Triangulation. This step constructs the 3D workspace by tiling adjacent contours with triangles. We have used the Fuchs' algorithm [FKU77] that interpolates the triangular faces between parallel slices in order to construct the 3D workspace surface from the different cross sections.

3.4.3 Criteria Selection

This module interacts with the user and selects the most suitable point computation and visualization algorithms based on the application requirements. Those requirements (parameters) include:

- *Surface/volume.* This parameter allows the user to select to either compute the workspace boundary (envelope) or compute the workspace volume based on the application requirements.

- *Complete/partial.* This parameter allows the user to compute either the full 3D workspace or just the portion of interest. If the user selects a partial workspace, it then asks for either the bounding cube or sphere that limits the portion of interest. The criteria selection would call the nonlinear-programming based algorithms from the point computation module since they are most suitable for computing partial workspaces.

- *Holes and voids.* This parameter allows the user to select computing holes and voids based on the application requirements. Computing the workspace envelope without holes and voids is important for some applications. On the other hand, computing the holes and voids is often unnecessary and anyway requires more time and space.

- *Resolution.* This parameter allows the user to select between a simple plot and a complex image based on the application deadline. The criteria selection asks the user to enter values for the required resolution in x, y, and z directions. These parameters are denoted res_x, res_y, and res_z respectively. The criteria selection passes those values to the point computation algorithms so that they can compute the right number of workspace points. If the user is interested in a quick response regardless of the image quality, low resolution values for the res_x, res_y, and res_z parameters can be entered.

We can now compute 3D workspaces for articulated chains with redundant DOFs and joint limits. The criteria selection module interacts with the user and selects the most suitable workspace computation and visualization algorithms based on the application requirements. The second module of that system computes workspace points for the given chain. The third module fits a 3D surface around the volume that encompasses the workspace points computed by the second module. An example of a 3D workspace for the left arm of a seated figure is illustrated in Plates 4 and 5. In Plate 4 the global shape of the workspace is visible as a translucent surface surrounding the body, while in Plate 5 we see what the figure can simultaneously reach with his left hand and *see* within the cockpit.

Chapter 4

Behavioral Control

The behaviors constitute a powerful vocabulary for postural control. The manipulation commands provide the stimuli; the behaviors determine the response. The rationale for using behavioral animation is its economy of description: a simple input from the user can generate very complex and realistic motion. By defining a simple set of rules for how objects behave, the user can control the objects through a much more intuitive and efficient language because much of the motion is generated automatically.

Several systems have used the notion of behaviors to describe and generate motion [Zel91]. The most prominent of this work is by Craig Reynolds, who used the notion of behavior models to generate animations of flocks of birds and schools of fish [Rey87]. The individual birds and fish operate using a simple set of rules which tell them how to react to the movement of the neighboring animals and the features of the environment. Some global parameters also guide the movement of the entire flock. William Reeves used the same basic idea but applied it very small inanimate objects, and he dubbed the result *particle systems* [Ree83].

Behaviors have also been applied to articulated figures. McKenna and Zeltzer [MPZ90] describe a computational environment for simulating virtual actors, principally designed to simulate an insect (a cockroach in particular) for animation purposes. Most of the action of the roach is in walking, and a gait controller generates the walking motion. Reflexes can modify the basic gait patterns. The stepping reflex triggers a leg to step if its upper body rotates beyond a certain angle. The load bearing reflex inhibits stepping if the leg is supporting the body. The over-reach reflex triggers a leg to move if it becomes over-extended. The system uses inverse kinematics to position the legs. *Jack* controls bipedal locomotion in a similar fashion (Section 5), but for now we focus on simpler though dramatically important postural behaviors.

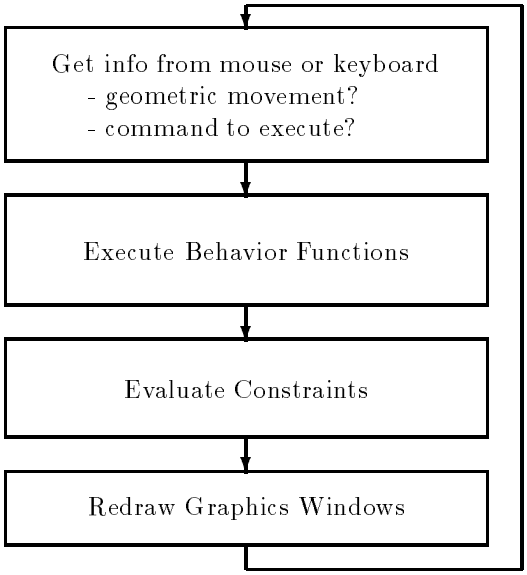

Figure 4.1: The Interactive System Architecture.

4.1 An Interactive System for Postural Control

The human figure in its natural state has constraints on its toes, heels, knees, pelvis, center of mass, hands, elbows, head and eyes. They correspond loosely to the columns of the staff in Labanotation, which designate the different parts of the body.

The heart of the interactive system is a control loop, shown in Figure 4.1. The system repeatedly evaluates the kinematic constraints and executes the *behavior functions* [PB91]. It also polls the user for information, which can be geometric movements through the direct manipulation operator described in Section 3.1, or commands to execute which can change the state of the system or the parameters of the constraints. The behavior functions can also modify the state of the environment or the parameters of the constraints, as described below. Each iteration of the control loop is a time step in a simulated movement process, although this is an imaginary sense of time.

There are four categories of controls for human figures, illustrated in Table 4.1. First, there are *behavioral parameters*. These are the parameters of the body constraints and govern things like whether the feet should remain planted on the floor or whether they should be allowed to twist as the body moves. Second, there are *passive behaviors*. These behaviors express relationships between different parts of the figure. These relationships are usually more complex than can be expressed through the behavioral parameters. An example of a passive behavior is the parametrization of the distribution of

weight between the feet. Third, there are *active behaviors*. Active behaviors have a temporal component. They wait for events or conditions to occur and then fire off a response which lasts for a specified duration. An example of an active behavior is the automatic stepping action the figure takes just before it loses its balance. Finally, there are *manipulation primitives*. These are the commands which allow the user to interactively drag or twist parts of a figure. These are the principal sources of input, the stimuli for the postural adjustments, although much of the movement during a postural adjustment usually comes from the response generated by the behavioral controls. Table 4.1 provides a summary of *Jack*'s current vocabulary for postural control.

4.1.1 Behavioral Parameters

The behavioral parameters are the properties of the constraints which model the posture. Mostly, these parameters describe the objective functions of the constraints, as in whether the constraint specifies position, orientation, or both. The parameters can include the goal values of the constraints as well. These are simple relationships which require no computation on the part of the system.

The *Jack* behaviors provide control over the position and orientation of the feet, the elevation of the center of mass, the global orientation of the torso, the orientation of the pelvis, and the gaze direction of the head and eyes. There are additional non-spatial controls on the knees and elbows. These controls were originally designed for the human figure in a rather ad hoc fashion, although they correspond loosely to the columns of the staff in Labanotation.

Most of *Jack*'s current behavioral parameter commands, listed in Table 4.1, allow the user to instruct the figure to maintain the current posture of the part of the body that it controls, such as the position and orientation of feet or the elevation of the center of mass. This means that this property will be maintained whenever possible even as the other parts of the figure change posture. This works well in an interactive manipulation context: the user manipulates the body part into place, and it stays there. This is the "what you see is what you get" approach. This also obviates the need for a complex syntax for describing positions and orientations grammatically, since the method of describing the location is graphical and interactive, through direct manipulation.

The Position and Orientation of the Feet

The foot behaviors are shown in Table 4.2. These are options to the set foot behavior command. The standard human figure model in *Jack* has a foot with two segments connected by a single toe joint, and two natural constraints, one on the toes and another on the heel. The toe constraint keeps the toes on the floor; the heel constraint can keep the heel on the floor or allow it to rise if necessary. For a standing figure, the pair of behaviors *keep heel on floor* and *allow heel to rise* control the height of the heel. The *pivot* behavior instructs

behavioral parameters
set foot behavior
pivot
hold global location
hold local location
keep heel on floor
allow heel to rise
set torso behavior
keep vertical
hold global orientation
set head behavior
fixate head
fixate eyes
release head
release eyes
set hand behavior
hands on hips
hands on knees
hold global location
hold local location
release hands
hand on site
grab object
passive behaviors
balance point follows feet
foot orientation follows balance line
pelvis follows feet orientation
hands maintain consistent orientation
root through center of mass
active behaviors
take step when losing balance
take step when pelvis is twisted
manipulation primitives
move foot
move center of mass
bend torso
rotate pelvis
move hand
move head
move eyes

Table 4.1: Behavioral Controls.

set foot behavior
pivot
hold global location
hold local location
keep heel on floor
allow heel to rise

Table 4.2: Foot Behaviors.

the toes to maintain the same position, and to maintain an orientation flat on the floor while allowing them to rotate through a vertical axis. The *hold global location* behavior disables the *pivot* behavior and fixes the toe orientation in space. This is the appropriate behavior when the foot is not on the floor. The *hold local location* behavior attaches the foot to an object such as a pedal. If the object moves, the foot will follow it and maintain the same relative displacement from it. If the figure is seated, then the heel behaviors and the *pivot* behavior have no effect, and the *hold* behaviors control the position and orientation of the heel instead of the toes.

The behavior of the feet is usually activated by the manipulation of some other part of the figure, such as the center of mass or the pelvis. A good example of the *pivot* behavior is when the center of mass is dragged towards one foot: should the other foot pivot in order to extend the leg, or should it remain planted and inhibit the movement of the center of mass? The behaviors say which should occur.

The Elevation of the Center of Mass

The horizontal location of the center of mass is a passive behavior which determines balance, as described below. The elevation of the center of mass is more straightforward. This concept has a direct analog in Labanotation: the *level of support* [Hut70]. A middle level of support is a natural standing posture, a low level of support is a squat, and a high level of support is standing on the tip-toes. The *hold current elevation* behavior is an option of the set balance behavior command. It instructs the figure to maintain the current elevation of the center of mass. This behavior is off by default. This behavior is necessary because under normal circumstances, the center of mass of the figure is free to rise and fall as necessary to meet the requirements of the feet. After adjusting the center of mass to an appropriate level, if no control holds it there, it may rise or fall unintentionally.

The Global Orientation of the Torso

The torso behaviors are listed in Table 4.3. The default behavior is *keep vertical*, which causes the torso to maintain a vertical orientation. Biomechanics research tells us that one of the most constant elements in simple human

set torso behavior
keep vertical
hold global orientation

Table 4.3: Torso Behaviors.

locomotor tasks is the global orientation of the head. One theory explaining this suggests that the head is the principle sensor of stability [BP88]. The *keep vertical* behavior mimics this nicely through a directional constraint on the chest to remain vertical, while not affecting its vertical rotation. This means that as the pelvis of the figure rotates forward, backward, or side to side, the torso will automatically compensate to keep the head up. Since the constraint is on the upper torso, not the head, the neck is free to move in order for the figure to look at certain reference points, as described below with the head and eye behaviors.

The *hold global orientation* behavior involves all three DOFs of the torso. This allows other parts of the body to be adjusted while the head and chest stay relatively fixed. This is particularly important in making adjustments to the pelvis and legs after positioning the torso acceptably. This behavior does not involve position, because it is usually acceptable to have the position float with the rest of the body.

Movements of the spine are described in terms of total bending angles in the forward, lateral, and axial directions. The technique uses weighting factors that distribute the total bending angle to the individual vertebrae in such a way that respects the proper coupling between the joints. Different weight distributions generate bends of different flavors, such as neck curls or bends confined to the lower back. These parameters are options to the torso behavior controls through the set torso behavior command because they govern how the torso behaves as it bends to maintain the proper orientation. The user can select one of the standard *curl from neck* or *bend from waist* options, or alternatively input the range of motion of the spine by selecting a top and bottom joint, and *initiator* and *resistor* joints which control the weighting between the vertebrae.

The Fixation Point for the Head and Eyes

The head and eyes can be controlled by specifying a fixation point, modeled through aiming constraints which orient them in the proper direction. The constraint on the head operates on a reference point between the eyes, oriented forwards of the head. The head constraint positions only the head, using the neck. The constraint on the eyes rotates only the eyeballs. The eyeballs rotate side to side and up and down in their sockets. The behavioral parameters control the head and eyes independently during postural adjustments. The active behaviors described below simulate the coupling between head and eye movement.

set head behavior
fixate head
fixate eyes
release head
release eyes

Table 4.4: Head Behaviors.

set hand behavior
hands on hips
hands on knees
hold global location
hold local location
release hands
hand on site
grab object

Table 4.5: Hand Behaviors.

The *fixate head* behavior option of the set head behavior command allows the user to select a fixation point for the head. The *fixate eyes* behavior does the same for the eyes. When these behaviors are active, the head and eyes will automatically adjust to remain focused on the fixation point as the body moves.

The Position and Orientation of the Hands

The principal control for the hands involves holding them at particular points in space as the body moves. Postural control of the arms and hands is usually a two step process. First, get the hands into position using the active manipulation facilities, and second, set a control to keep them there as some other part of the body moves. The *hold global location* and *hold local location* behaviors serve much the same for the hands as their counterparts for the feet. The desired geometric positions and orientations are either global or local to some other object. The set hand behavior command provides several standard postures. For a standing figure, a pleasing reference point is the figure's hips. The hands rest on the hips with the elbows out to the side, the arms akimbo posture. For a seated position, a pleasing reference point is the figure's knees. The *site* behavior moves the hand to a particular site, in both position and orientation. This simulates a reaching movement, but its real purpose is to hold the hand there once it reaches the site.

The controls for the hands are invoked whenever the body moves or whenever an object to which the hand is constrained changes location. A good

analogy is holding onto an object such as a doorknob. If the door closes, the arm goes with it. Likewise, if the body bends over, the door stays fixed and the arm adjusts accordingly.

Jack also allows the converse relationship which is more suited to the way a person holds a screwdriver. A screwdriver is controlled completely by the hand and is not fixed in space in the same sense as the door. If the body bends over, the screwdriver should move along with the arm, not remain in place like the doorknob. This type of a relationship comes from the *grab object* behavior. "Grab" in this context does not mean "grasp"; it doesn't mean the fingers will wrap around the object. Such an action is available as a type of motion (Section 4.2.7). It actually means that the object will subsequently be attached to the figure's hand, as if the figure grabbed it. Once again, the process has two-steps: first, position the hand and the object relative to each other, then specify the *grab* behavior to hold it there.

The constraints on the hands are logically separate from the other constraints on the body. *Jack* evaluates the hand constraints *after* the other constraints, not simultaneously. The reasons for structuring the arm behaviors this way are partly practical and partly philosophical.

In practice, the inverse kinematics algorithm does not perform well when the hand constraint is considered collectively with the other body constraints. With the human figure rooted through the toes, there are too many DOFs between the toes and the hands to be controlled effectively. Even though the constraints on the pelvis, center of mass, and opposite foot help to resolve this redundancy, if the hand constraint is on equal par with the other parts of the body, the hand constraint can frequently cause the other constraints to be pulled away from their goals. Since the potential energy function describing the equilibrium state for the figure is a weighted combination of all of the constraints, the center of mass and pelvis constraints must have significantly higher weight to avoid having the hand pull the body off balance. It has been difficult to arrive at a set of weights which give the right behavior. It has been much easier to simply localize the movements of the arms and isolate them from the rest of the body.

Philosophically, it is acceptable to consider the arm movements independently as well. Normally, a reaching task does not initiate much movement of the lower body, unless there are explicit instructions to do so. For example, consider what happens when a human being reaches for a nearby object such as a doorknob. If the door is near enough, this won't involve any bending of the waist, but if the door is farther away, it may be necessary to bend the waist. If the door is farther away still, it may be necessary to squat or counter-balance by raising a leg backwards. This system of behaviors requires an explicit control specifying which approach the figure takes. If the torso must bend or the center of mass must shift in order to perform a reaching task, then the user, or some higher level behavior function, must initiate it. Automatic generation of these intermediate postures is non-trivial. We discuss two rather different approaches in Sections 5.4 and 5.5.

The Knees and Elbows

The knees and elbows require special care to prevent them from becoming locked at full extension. The fully extended position not only appears awkward, but it tends to cause the inverse kinematics algorithm to get trapped in a local minimum. Because the algorithm uses a gradient descent approach, if an elbow or knee reaches its limit, it has a tendency to stay there[1]. To prevent this, *Jack* uses *limit spring* constraints to discourage the knees and elbows from reaching their limiting value. The springs give a high energy level to the fully extended angle. The springs can be tuned to any angle, but the default is 10°, and in practice, this tends to work well.

This gives the figure in its natural standing position a more pleasing posture, more like "at ease" than "attention". Biomechanics literature describes this in terms of the stresses on the muscles [Car72]. Labanotation uses this posture as the default: the elbows are "neither bent nor stretched" [Hut70] and, in the middle level of support, the knees are "straight but not taut" [Hut70].

The Pelvis

The pelvis and torso are intricatedly related. The torso includes all joints in the spine from the waist to the neck, and rotating these joints allows the figure to bend over. However, when human beings bend over, they generally bend their pelvis as well as their torso. This means manipulating the two hip joints as a unit, which can be a problem for a computer model because there is no single fixed point below the hips from which to rotate. However, this problem is easy to handle by controlling the orientation of the pelvis through a constraint.

4.1.2 Passive Behaviors

Passive behaviors can represent more complex relationships than the behavioral parameters. They are like little processes attached to each figure. The passive behavior functions are executed at each interactive iteration. A passive behavior can involve a global property of the figure such as the center of mass or the shape of the figure's support polygon. An example of this kind of behavior is the parametrization of the distribution of the weight between the figure's feet: when the feet move, the balance behavior function must compute the proper location for the balance point and register this with the constraint on the center of mass. Passive behaviors are *instantaneous* in that they explicitly define a relationship to be computed at each iteration.

Passive behaviors are easy to implement in this basic system architecture because their only job is to compute the necessary global information and supply it to the behavioral controls. Because of the general nature of the inverse

[1] Essentially the algorithm sees a zero gradient and hence finds no advantage to moving the locked joint as that does not decrease the overall distance to the goal: the required motion is in fact exactly perpendicular to the aligned segments.

kinematics constraints, the behaviors can overlap to a degree not possible with other systems, like Zeltzer's local motion processes [Zel82, Zel84].

Currently, *Jack* has implemented six basic passive behavior functions for human figures, and they illustrate a range of capabilities. They control the location of the balance point, the orientation of the feet, the orientation of the pelvis, and the orientation of the hands. The final two behaviors control the figure root.

Balance as a Passive Behavior

Probably the most important human postural behavior, and the one demanding the most coordination, is balance. The need to remain balanced dictates much of the subtle and elusive behavior of a human figure. The location of the balance point of a figure is significant in both cause and effect. The location of the balance point is dependent on other parts of the figure, namely, the feet. Also, the balance point sends information to the other parts of the body regarding the figure's state of balance. Requiring a process to handle balance in a global fashion was recognized long ago [BS79, BOK80], but significant progress in computer interactivity and posture behavior algorithms was needed to realize that design.

To parametrize the location of the balance point with respect to the feet we use the *balance line*, which is the line between a fixed reference point in the middle of each foot. Biomechanics literature [Car72] states that in the standing rest position, the body's vertical line passes 2-5cm in front of the ankle joint, midway through the arch of the foot. This line between the feet divides the support polygon down the middle.

Given the location of the center of mass, the balance point parameters, call them x and z, can be determined as shown in Figure 4.2. To do this, project the balance point on the $y = 0$ plane and call the point $\mathbf{b}$. Then find the point on the balance line closest to this point, and call it $\mathbf{p}$. z is the distance between $\mathbf{b}$ and $\mathbf{p}$, that is, the balance point's distance forward from the balance line. However, it is more convenient to normalize z between 0.0 and 1.0 according to the placement of $\mathbf{b}$ between the balance line and the front edge of the support polygon. Therefore, if $z > 1$ then the balance point lies outside the support polygon. If the balance point is behind the balance line, then let z be normalized between -1.0 and 0.0. Likewise, x is the interpolation factor which gives $\mathbf{p}$ in terms of the left and right foot reference points, normalized between 0.0 and 1.0, with $x = 0$ being the left foot. If x is outside of the $[0, 1]$, then the balance point is to the side of the support polygon.

Once the system has the ability to measure balance, these parameters are available for the behavior functions to use. The *balance point follows feet* behavior, described in Section 4.1.2, falls directly out of this parametrization. This behavior causes the distribution of weight between the feet to remain constant even as the feet move. The active stepping behavior *take step when losing balance*, described in Section 4.1.3, uses this parameter as its trigger.

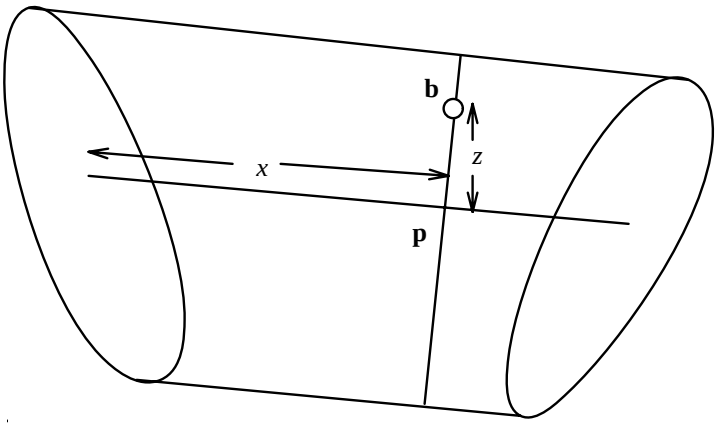

Figure 4.2: The Parametrization of the Balance Point.

Global Effects of Local Manipulations

Another capability of the passive behaviors in this system is to telegraph changes in the posture of a local part of the figure to the rest of the figure as a whole. This can provide coordination between the different parts of the figure. The behavioral parameters as described above generally hold the different parts of the figure in place, but sometimes it is better to have them move automatically. A good example of this is the *pelvis follows foot orientation* behavior, described in Section 4.1.2, in which the orientation of the pelvis automatically adjusts to the orientation of the feet. Whenever the feet change orientation, they radiate the change to the pelvis which mimics the rotational spring-like behavior of the legs.

Negotiating Position and Orientation

The passive behaviors offer a solution to the problem of negotiating the overlapping influence of position and orientation while interactively dragging part of the body. Because of the nature of the direct manipulation technique described in Section 3.1, it is not possible to rotate and translate during a single movement of the mouse. This has come up before, in Section 3.2.5: either the dragging procedure has *no* control over orientation, in which case the orientation is arbitrary and unpredictable, or the dragging procedure *does* have control over orientation, in which case the orientation remains globally fixed during spurts of translation. Fixing the orientation during translation can, for example, cause the hand to assume an awkward orientation as it is translated.

Passive behavior functions allow the direct manipulation operator to have control over the orientation and avoid awkward orientations. While the user is translating with the mouse, the behavior function can automatically determine a suitable orientation based on heuristic observations. While rotating, the user has complete control over the orientation. The heuristics can simply

be embedded in the behavior functions (Section 2.4).

The pair of behaviors *foot orientation follows balance line* and *hands maintain consistent orientation*, use heuristics taken from Labanotation to predict suitable orientations for the hands and the feet during their manipulation. This allows the user to position them mostly by translating them, making changes to the orientation only as necessary.

The Figure Root

One passive behavior deserves special attention: the figure root. The principal disadvantage of modeling an articulated figure as a hierarchy is that one point on the figure must be designated as the figure root. Section 3.2.3 explains the effect of the figure root on the inverse kinematics algorithm: the positioning algorithm itself cannot move the figure root. It can only manipulate chains emanating from the root. Any movement of the figure root must be programmed explicitly. Therefore, a major element of **Peabody** is the ability to change the setting of the figure root when necessary.

The figure "root" is an unnatural concept. It has no natural analog for a mobile figure like a human being, so it has no place in the language for controlling human figures. Since it is a necessary internal concept, can it be controlled internally as well? For certain postures of a human figure, there are distinct reference points on the figure which serve as good figure roots: the feet, the lower torso, and the center of mass. It should be possible to have the system choose these automatically and thus make the root transparent to the user.

There are several possibilities for the figure root of a human figure. Many systems which don't have the ability to change the root choose to locate it at the lower torso [BKK+85]. However, this complicates the process of moving the lower torso during balance adjustments. Using this approach, it can be very difficult to get the figure to bend over convincingly because the hips need to shift backwards and downwards in ways that are difficult to predict. However, for a seated posture, the lower torso is a good choice for the root. When a figure is standing, the feet are natural choices for the root.

The choice of the figure root can be handled by designing a behavior function which monitors the figure's posture and automatically changes the figure root when necessary to provide the best behavior. This behavior function uses the following rules:

- It roots the figure through a foot whenever the weight of the body is more than 60% on that foot. This ensures that if the figure is standing with more weight on one leg than the other, the supporting leg serves as the root. It also ensures that if the figure is standing with weight equally between the two legs but possibly swaying from side to side that the root doesn't rapidly vacillate between the legs.

- If the height of the center of mass above the feet dips below 70% of the length of the leg, then the root changes to the lower torso. This

predicts that the figure is sitting down. Heuristically, this proves to be a good choice even if the figure is only squatting, because the constraint on the non-support leg tends to behave badly when both knees are bent to their extremes.

Balance Point Follows Feet

Labanotation has a notion for the distribution of the weight between the feet and the shifting of the weight back and forth [Hut70]. This notion is well-defined regardless of the position of the feet: after specifying the distribution of weight between the feet, this proportion should remain fixed even if the placement of the feet need adjustment during a postural manipulation. This is the job of the *balance point follows feet* behavior.

Given these two parameters, a new balance point can be computed based on any new position of the feet. Holding these parameters fixed as the feet move ensures that the balance point maintains the same relationship to the feet, both laterally and in the forward/backward direction.

Foot Orientation Follows Balance Line

During the active manipulation of the feet with the move foot command, the user can intersperse translations and rotations of the feet, centered around the toes. Since it is not possible to rotate and translate during a single movement, either the dragging procedure has *no* control over orientation, in which case the orientation is arbitrary and unpredictable, or the dragging procedure *does* have control over orientation, in which case the orientation remains globally fixed during spurts of translation. The *foot orientation follows balance line* behavior offers a convenient alternative.

The solution which the behavior offers is to predict the proper orientation of the foot based on the balance line and adjust the orientation automatically as the foot is translated with the move foot command. The balance line, as described above, is an imaginary line between the middle of the feet. Actually, this rule fixes the orientation of the foot with respect to the balance line. As the foot translates, the balance line changes, and the orientation of the foot changes to keep the same relative orientation. This behavior is particularly appropriate when the figure is taking a step forward with the intention of turning to the side.

Pelvis Follows Feet Orientation

The muscles in the leg make the leg act like a rotational spring. The hip and ankle joints provide only a little more than 90° of rotation in the leg around the vertical axis. This means that the orientation of the feet and the orientation of the pelvis are linked together. If the orientation of the feet are fixed, the orientation of the pelvis is severely limited. What is more, the extreme limits of pelvis orientation place an uncomfortable twist on the legs. If the legs are rotational springs, then the "middle" orientation of the

pelvis can be determined by simply averaging the orientation of the feet. This seems to be in fact what happens when a person stands naturally: the pelvis is oriented to relieve stress on the legs. The *pelvis follows feet orientation* behavior simulates this.

Hands Maintain Consistent Orientation

The same problem with the orientation of the feet during the move foot command occurs with the hands with the move hand command. In fact, the problem is more intricate because the hands have a much greater range of movement than the feet. How is the orientation of the hand related to its position? How can this be determined automatically in order to predict reasonable postures when moving the hands?

Labanotation suggests an answer. Labanotation has a detailed system for describing arm postures and gestures, but what is most interesting here is what the notation does *not* say. To simplify the syntax, Labanotation has a standard set of orientations for the palms when the arms are in specific positions. Notations need be made only when the orientations differ from these defaults. The rules are [Hut70]:

- When the arms hang by the side of the body, the palms face in.
- When the arms are raised forward or upward, the palms face towards each other.
- When the arms are raised outward to the side, the palms face forward.
- When the arms cross the body, the palms face backward.

These rules are useful as defaults, but of course they do not dictate absolute behavior. These rules govern the orientation of the hands when the user translates them from one area to another without specifying any orientational change. These rules only take effect when the hand moves from one region to another.

Root Through Center of Mass

Most of the behaviors described so far are only appropriate for standing figures, which of course means that they are also only appropriate for earth-bound figures. But what about figures in zero-gravity space? This is actually quite easy to simulate by rooting the figure through the center of mass and disabling all other behaviors. The one constant element of zero-gravity is the center of mass. When the figure is rooted through the center of mass, the global location of the center of mass remains fixed as the figure moves.

4.1.3 Active Behaviors

Active behaviors mimic reflexive responses to certain conditions in the body. They can have temporal elements, so they can initiate movements of parts of

the body which last for a certain duration. These behaviors make use of the concept of a *motion primitive*. A motion primitive has a distinct duration in terms of interactive iterations, and it typically involves a constraint which changes over this time interval. An example of this is the stepping movement of the feet which is initiated when the figure's center of mass leaves its support area. The interactive system architecture maintains a list of currently triggered active behaviors, and it advances them at each iteration until they are complete. The behaviors terminate themselves, so the duration can be explicit in terms of a number of interactive iterations, or they can continue until a certain condition is met.

Active behaviors are like motor programs, or *schemas* [Kee82, Ros91, Sch82b, Sch82a]. Considerable physiological and psychological evidence suggests the existence of motor programs, which are preprogrammed motor response to certain conditions. The theory of schemas suggests that humans and animals have catalogs of preprogrammed motor responses that are fired off to produce coordinated movements in the body. Schemas are parametrized motor programs which can be instantiated with different settings. For some motor programs, there even seems to be very little feedback involved. Evidence of this comes from experiments which measure the excitation of the muscles of the arm during reaching exercises. The patterns of excitation remain constant even if the movement of the hand is impeded [Ros91].

The incorporation of active behaviors into the postural control process begins to blur the distinction between motion and manipulation. The purpose of the behaviors is predictive: if the user drags the center of mass of a figure away from the support polygon, this probably means that the desired posture has the feet in a different location. The job of the active behavior is to anticipate this and hopefully perform the positioning task automatically.

There are two active behaviors, both involving the placement of the feet. The *take step when losing balance* and *take step when pelvis is twisted* behaviors automatically reposition the feet just before the figure loses its balance. They use the balance point parameters described above as their triggers. The purpose of these behaviors is to predict a proper posture for the figure given that its center of mass is leaving the support polygon.

Active behaviors can be used to simulate movement even in the context of postural control. The entire process of interactive postural control can serve as a good approximation to motion anyway. The active behaviors provide a way in which motion primitives can be incorporated into the interactive system. To do this more effectively, the interactive system needs a more sophisticated notion of time and timed events (Section 4.3).

Take Step When Losing Balance

This behavior fires a stepping response to the loss of balance of a figure. When this behavior is active, it monitors the parametrization of the balance point of the figure as described with the *balance point follows feet* behavior. If the balance point leaves the support polygon to the front or back, the behavior

move foot
move center of mass
bend torso
rotate pelvis
move hand
move head
move eyes

Table 4.6: The Manipulation Primitives.

moves the non-support foot forward or backward to compensate. The non-support foot in this case is the one which bears less weight. The behavior computes the new foot location such that the current balance point will lie in the middle of the new support polygon once the foot arrives there. If the balance point leaves the support polygon to the side, the stepping motion moves the support foot instead. In this case, the support foot is the only one which can be repositioned in order to maintain balance.

Take Step When Pelvis Is Twisted

The discussion of the *pelvis follows feet orientation* behavior above described the relationship between the global orientations of the feet and pelvis, particularly in terms of determining an orientation for the pelvis from the orientation of the feet. The opposite relationship is possible as well. Consider standing with your feet slightly apart, and then begin to twist your body to the right. After about 45° of rotation, your legs will not be able to rotate any more. In order to continue rotating, you will be forced to take a step, a circular step with either your left or right foot.

The *take step when pelvis twisted* behavior mimics this. When it senses that the orientation of the pelvis is near its limit relative to the feet, it repositions the non-support foot in an arc in front of or behind the other foot, twisted 90°.

4.2 Interactive Manipulation With Behaviors

This section discusses the *Jack* manipulation primitives, but in the process it describes the entire manipulation process, including the effect of all of the implemented behaviors. The effect of the manipulation commands cannot be treated in isolation. In fact, the very nature of the system of behaviors implies that nothing happens in isolation. This discussion serves as a good summary of these techniques because these are the commands (Table 4.6) which the user uses the most. These are the verbs in the postural control language.

The interactive postural control vocabulary includes manipulation primitives which allow the user to push, poke, and twist parts of the body, and behavior controls which govern the body's response. The manipulation commands are sufficiently intuitive to provide good handles on the figure, and the behavioral controls make the responses reasonable.

The structure of the behaviors for human figures did not come out of a magic hat. The rationale behind the behaviors comes partially from biomechanics and physiology literature, and partially from the semantics of movement notations such as Labanotation. Labanotation provides a good set of default values for the behaviors because it incorporates so many assumptions about normal human movement.

4.2.1 The Feet

The feet can be moved with the active manipulation command move foot. This command allows the user to drag the foot interactively. This automatically transfers the support of the figure to the other foot, provided the figure is standing. The control over the position of the feet is straightforward. The manipulation operator also gives control over the orientation. However, while translating the foot, its orientation depends upon the foot orientation behavior. The default behavior maintains a constant global orientation. The *foot orientation follows balance line* behavior causes the orientation of the foot to remain fixed with respect to the balance line during translation. This means that if the foot goes forward, it automatically rotates as if the figure is turning toward the direction of the stationary foot.

The move foot command automatically causes a change in the balance point according to the *balance point follows feet* behavior, which is the default. This means that the distribution of weight between the feet will remain constant as the foot moves. The location of the balance point within the support polygon, both side to side and forwards/backwards, will remain fixed as the support polygon changes shape. This is evident in Figure 4.3. The balance point shifts along with the foot. If this behavior is disabled, the balance point will remain fixed in space.

Manipulating the feet also telegraphs a change to the pelvis according to the *pelvis follows foot orientation* behavior, which is the default. This means that as the foot rotates, the pelvis automatically rotates as well. This keeps the body turned in the direction of the feet.

4.2.2 The Center of Mass and Balance

The move center of mass command allows the user to interactively drag the balance point of the figure, shifting its weight back and forth or forward and backward. This command changes the parametrization of the balance point in terms of the feet. If the *balance point follows feet* behavior is active, then when the move center of mass command terminates, the balance point will remain at its new location relative to the support polygon.

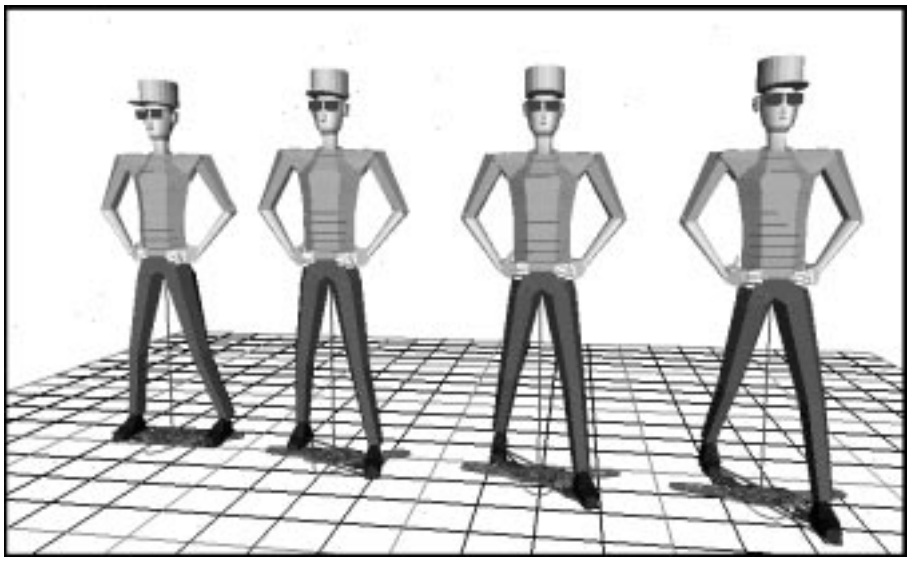

Figure 4.3: Moving the Left Foot, with Balance Point Following Feet.

The location of the balance point has a great effect on the feet. If the foot behavior is *pivot*, then shifting the weight laterally back and forth will cause the feet to twist back and forth as well. On the other hand, if the feet do not pivot, then they remain planted, possibly inhibiting the movement of the balance point. In Figure 4.4, the feet are held in place, not pivoting.

The move center of mass command also gives control over the elevation of the center of mass. Normally, the elevation of the center of mass is not controlled explicitly, except through the *hold current elevation* behavior option to the *set balance behavior* command. The move center of mass command gives control over the elevation, so moving the center of mass up and down allows the figure to stand on its tip-toes or squat down. Figure 4.5 shows the center of mass being lowered into a squatting posture. The constraint on the pelvis ensures that the hips remain square and straight.

The movement of the center of mass also tends to trigger the rooting behavior. This is mostly transparent, but to the trained eye, it is apparent in the movement of the feet. The support foot (the rooted one) is always very stationary.

The manipulation of the center of mass is the main instigator of the active stepping behavior. While the stepping behavior is active, if the balance point reaches the perimeter of the support polygon, the feet are automatically repositioned by the stepping behavior. Figure 4.6 illustrates the stepping behavior as the center of mass is dragged forward. When this occurs, the visual impression is of the figure being pulled and taking a step just to prevent a fall; it does not look like the figure is deliberately trying to walk somewhere. (In Section 5.2 more purposeful stepping and walking behaviors are utilized.)

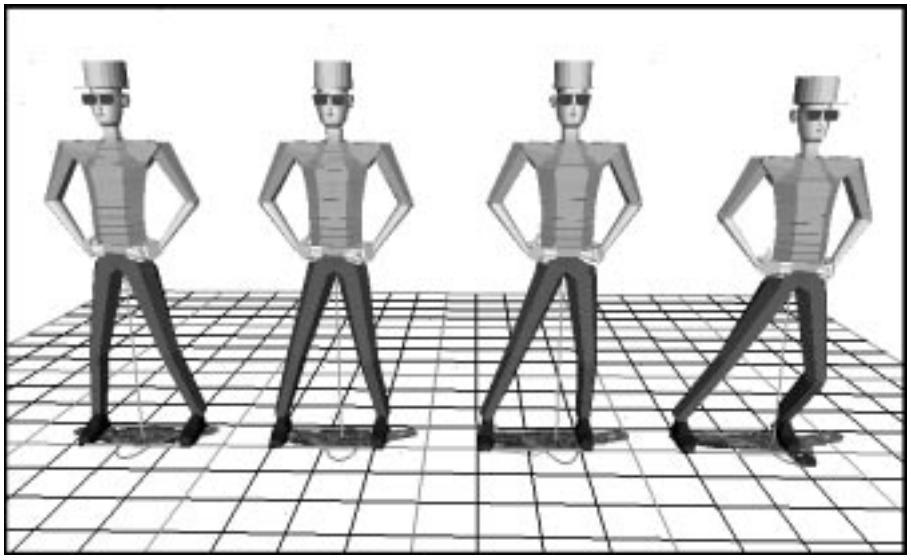

Figure 4.4: Shifting the Center of Mass.

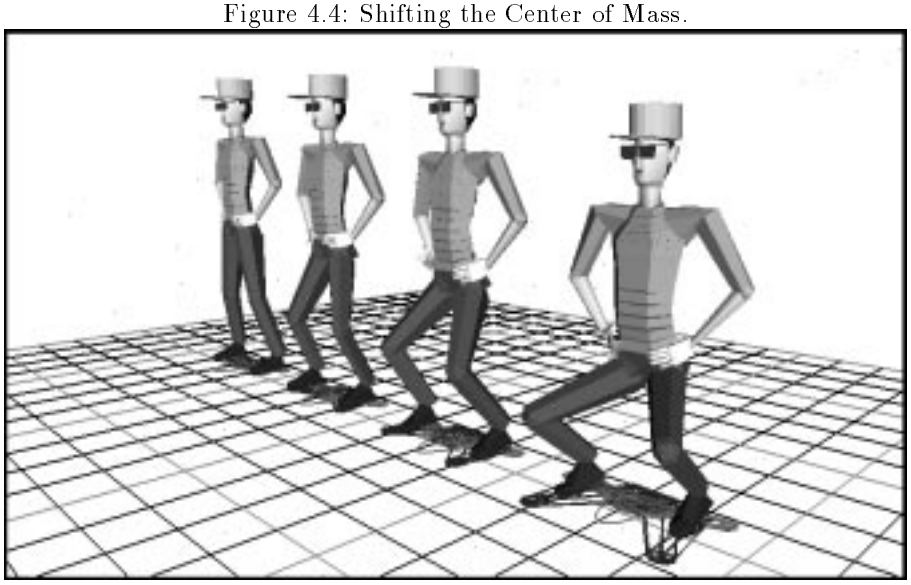

Figure 4.5: Lowering the Center of Mass.

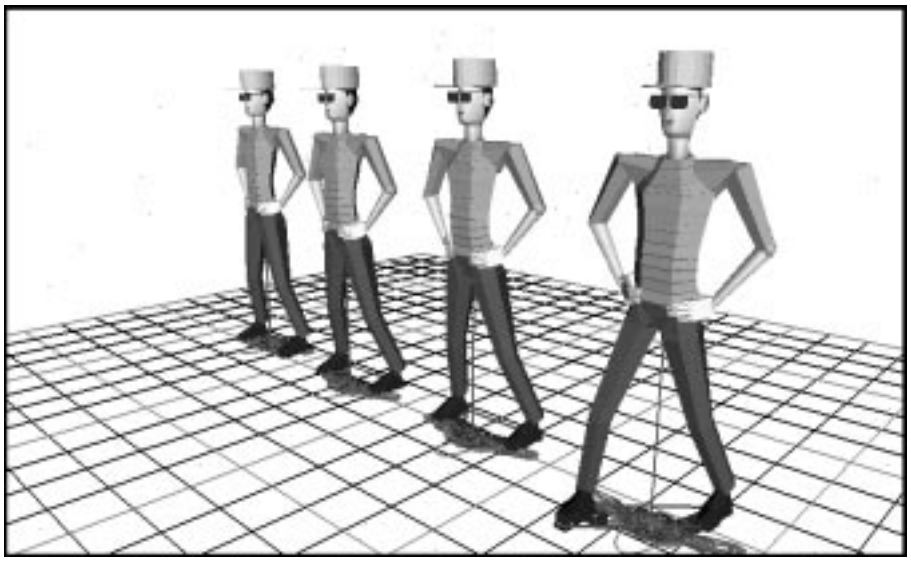

Figure 4.6: Taking a Step before Losing Balance.

4.2.3 The Torso

The *Jack* spine model provides a very important biomechanical feature for effective human behavioral control. Each vertebra has a current position defined by the three joint angles relative to its proximal vertebra. Also defined in the spinal database are joint rest positions and 6 joint limits for every joint. If each attribute is summed up for all joints, then 3D vectors are defined for current position, joint rest position, and two joint limits for the global spine. The target position – the 3D vector sum of final joint positions – is supplied as an input parameter. Movement towards the target position is either bending or unbending, meaning either towards the joint limits or towards the spine's rest position. Motion is defined as an interpolation between the current position and either the spine's position of maximum limit, or the spine's rest position.

Three rotations are calculated independently and then merged into one. For example, a 3D orientation vector (e.g. flex 45 degrees, rotate axially 20 degrees left, and lateral bend 15 degrees right) can be accomplished in one function with 3 loop iterations. It is assumed for the model that the maximum vertebral joint limit in one dimension will not affect the joint limits of another dimension.

The spine's rest position is included in the model, because it is a position of high comfort and stability. If the spine is unbending in one dimension of movement, it will move towards that position of highest comfort in that rotational dimension. The input parameters listed in Section 2.3 determine how much each vertebra bends as the spine moves. The three dimensions are