particular optimization problems. The condition monitor is the highest level of the three procedures in predicting a path.

Path Planning Scheme

The path planning scheme, guided by the condition monitor, determines the direction to move. In general, the output of any system is bounded by its headroom: the available range of a variable within a constraint. In the case when there is much strength in a system (a situation where indirect joint control applies) the headroom can be used to suggest incremental joint displacements, $d\theta$. A larger headroom allows a larger displacement. The mapping between the cartesian displacement and the joint displacement is

$$d\mathbf{x} = \mathbf{J}d\theta \tag{5.17}$$

where $\mathbf{J}$ is the $3 \times n$ Jacobian matrix and n is the number of joint displacements. If the headroom for each joint is represented by a weighting vector $\mathbf{w}$ proportional to $d\theta$, then

$$d\hat{\mathbf{x}} = \mathbf{J}\mathbf{w} \tag{5.18}$$

where $d\hat{\mathbf{x}}$ is a normalized direction of reach. The direction $d\hat{\mathbf{x}}$ is then compared against a cone representing the feasible directions of travel derived from perceived exertion. If $d\hat{\mathbf{x}}$ is within the cone then the direction of motion should be $d\hat{\mathbf{x}}$, otherwise the direction can be $d\hat{\mathbf{x}}$ projected onto the cone.

When the system is relatively weak, the suggested direction of motion must not violate the strength constraints. The decision process should shift importance from one strategy where the desirability to reach a goal is a major component of determining a suggested motion to an alternative strategy of avoiding positions where the joints are greatly strained. This leads to schemes where direct joint control is imperative to avoid positions where joints are strained [Lee92].

Rate Control Process

The rate control process, the most basic of the three procedures, resolves the speed with which a body moves along a prescribed end-effector path. This requires the use of dynamics, especially when the motion is fast. However, the incorporation of dynamics is difficult. When torques are specified to drive a motion (direct dynamics), control is a problem; when the driving forces are derived by kinematic specification (inverse dynamics), the forces are useful for only a short time interval and they may violate the body's torque capacity; and finally, when the forces optimize a particular function between two sets of positional constraints (boundary value problem), the method presumes that the optimization criteria is valid for the body's entire range of motion.

Dynamics equations can be interpreted as constraint equations solving for joint trajectories if they satisfy the conditions imposed by specific end-effector path and torque limits. The dynamics equations can be reformulated so that

they provide a mapping between an end-effector path and a binding torque constraint. A binding torque constraint is the maximum torque allowed to drive a body with maximum end-effector acceleration without the end-effector deviating from the prescribed path. A greater torque would cause excessive inertial force and therefore, undesirable path deviation. From the derivation of the reformulated dynamics equations originally derived to solve for path completion in minimum time [Bob88], joint trajectories can be found from the acceleration of an end-effector. In addition to finding the trajectories, the reformulated dynamic equations implicitly determine the force functions (joint torques) to guide an end-effector along a specified path.

Torque limits are established by the current discomfort constraint. The discomfort level variable *dcl* determines the torque limit at each joint by a simple relation:

$$dcl = \frac{\tau_{c,i}}{\tau(\theta)_{max,i}} \tag{5.19}$$

where $\tau_{c,i}$ is the torque load for a particular joint i. The value $\tau(\theta)_{max,i}$ is the maximum torque for the joint's current position obtained by querying the strength curves. When the value of *dcl* becomes greater than one, there is no more strength to accomplish a task and therefore the attempt to complete a task should cease. The discomfort level can be adjusted to achieve a desired motion. It influences both the rate of task completion and the direction of travel.

5.3.4 Motion Strategies

This is a catalogue of human motion strategies that are evaluated in the condition monitor and are executed in the path planner. The strategies are given in the order of increasing discomfort.

Available Torque

When a person moves, the tendency is to move the stronger joint. This is similar to the forces due to a spring or other types of potential forces. A stronger spring, based on the spring's stiffness coefficient, would yield a larger displacement per unit time than a weaker spring. Similarly, for a human, the amount of displacement for a joint depends not only on the strength that a joint is capable of, but mainly on the amount of strength that is currently available. The amount of strength available, which is based on the difference between the current required torque to support a particular position and the *effective maximum strength* (the maximum strength factored by comfort), is called *torque availability*. If torque availability is low, motion should not be encouraged. Conversely, if the torque availability is high, the joint should do more of the work. Torque availability is the driving factor for a joint to move and to thereby redistribute the joint torques so that the comfort level is more uniform.

Reducing Moment

As a joint approaches its effective maximum strength, the joint should move in a manner that avoids further stress (discomfort) while still trying to reach the goal. A path towards the goal is still possible as long as the maximum strength is not surpassed for any of the joints. As the body gets more stressed it should attempt to reduce the moment caused by a force trajectory by reducing the distance normal to the force trajectory's point of application. In addition, a reduction in moment increases the torque availability of (at least) the joint that is rapidly approaching its maximum strength. The reduction in the total moment involves examining the moments on a joint by joint basis. At each joint a virtual displacement is given to determine if that displacement provides sufficient moment reduction to continue moving in that direction. This strategy assumes that the body has enough effective strength to allow its joints to move to positions where the overall stress level of the body is smaller than if the joints were only guided by kinematic demands.

Pull Back

The two previous strategies depend on the current torque being less than the maximum strength. In these cases, maneuverability in torque space is high and therefore an end-effector can still consider moving toward a goal without exceeding any joint's maximum strength.

When a particular joint reaches its maximum strength, however, then that joint can no longer advance toward a goal from the current configuration. The Pull Back strategy proposes that the end-effector approach the goal from another configuration. In an effort to determine another approach to the goal, the constraint of moving toward a goal within a restricted path deviation can be relaxed. The emphasis of the strategy is one where the joints dictate an improved path in terms of torques. This can be accomplished by increasing the *ultimate available torque* – the difference of maximum strength to current torque – for a set of *weak joints* – joints that are between the joint which has no ultimate available strength and an end-effector.

In general, the joint with the least amount of ultimate available torque will reverse direction and cause the end-effector to pull back (move away from its goal). The idea is to increase the overall comfort level. When the joints form a configuration that has a greater level of comfort, there might be enough strength to complete the task. Then the governing strategy could return to Reducing Moment, which allows the end-effector to proceed toward the goal.

The Pull Back strategy leads to a *stable configuration*. This is a posture that a set of joints should form so that it can withstand large forces, such as those caused when changing from a near-static situation to one that is dynamic.

Added Joint, Recoil, and Jerk

When the three strategies, Available Torque, Reducing Moment, and Pull Back have been exhausted and an agent still cannot complete a task, it is obvious that the *active joints* – the joints that were initially assigned to the task – cannot supply sufficient strength. When this occurs it should be determined if the task should be aborted or if there are other means of acquiring additional strength. One mode of acquiring more strength is to add a joint to the chain of active joints. This assumes that the added joint is much stronger than any of the active joints.

Another mode to consider is to use the added joint to jerk – apply with maximum force – the set of active joints. Jerk reduces the forces necessary to complete a task for the set of active joints. Before jerking is initiated, a stable configuration should be formed by the active joints. After a stable configuration has been formed and the added joint has jerked, the active joints can then proceed to reach their goal since the required torques have decreased. A third possibility is to recoil another set of joints and then jerk with the recoiled set of joints in order to reduce the forces needed by the set of active joints to complete a task. For example, a weight lifter sometimes recoils his legs and then pushs off to reduce the force required in his arms.

5.3.5 Selecting the Active Constraints

The path determination process has been uncoupled into two active constraints: comfort and perceived exertion. In the rate control process involving dynamics, the two constraining parameters must be active to determine the joint rates.

At higher levels of control (such as in the path planner), both need not be active simultaneously. In fact, as the torque levels change the applicability of a particular constraint to predict motion also changes. We use a model that relates the comfort level to the constraints. The strategies bound various comfort levels.

High Comfort. The perceived exertion constraint is not active but the comfort constraint is, because any changes in acceleration (not necessarily large) may cause a joint to exceed the allowable discomfort level. In general, the force trajectory associated with a motion of high comfort is negligible, but dynamics is important because of the relatively large inertial effects of the body. This group is bounded by motions that are categorized by *zero jerk* condition [Gir91] and Available Torque.

Regular Comfort. The end-effector can advance toward the goal. Perceived exertion and comfort are loosely constraining and dynamics should be evaluated. Available Torque and Reducing Moment bounds this comfort level.

Discomfort. At this point the discomfort level for one or more joints are surpassed. The perceived exertion constraint needs to be changed so

that a larger path deviation is allowed. Motion should have slowed down considerably, therefore dynamics is not important and, most likely, is not meaningful. This group is formed by Reducing Moment and Pull Back.

Intolerable Discomfort Many of the joints' comfort levels have been exceeded and there may be other joints which could be approaching their maximum available torque. In such a situation, strategies can be combined. The pool of strategies are Added Joint, Pull Back, Recoil, and Jerk. Perceived exertion is relaxed and depending on the combination of the strategies, dynamics might be important.

5.3.6 Strength Guided Motion Examples

The strategies for Available Torque, Reducing Moment, Pull Back, and Added Joint are implemented in *Jack*. Figures 5.15, 5.16, 5.17, and 5.18 show the paths that were produced from these conditions. The task is to place an increasingly heavy load from various initial positions at a goal which is located above the body's head.

In Figure 5.15, there are two curves which outline the path of the hand. The right curve is for a task that involves lifting a 10 pound object; the left curve is for a 20 pound object. For the right curve, because the object is relatively light, a fast motion is predicted and the solution resembles a minimum time path. For the left curve, the heavier weight draws the hand closer to the body. This path is rough because it is at the boundary of a solution determined by Available Torque and Reducing Moment. In Figure 5.16, the right curve is for the 20 pound lift, and the left curve is for a lift of 30 pounds. Once again the algorithm predicts that a heavier object would bring the hand closer to the body.

Figure 5.17 shows the body with a heavier load (35 pounds). The body immediately executes Pull Back. In this case, the body pulls back to a region of high comfort and therefore the approach to the goal is smooth, without the rough path evident in the previous figures. In Figure 5.18, the joint chain, initially composed of the joints between the hand and the shoulder, is allowed to extend to the waist. The algorithm decides that it is better to distribute the weight with more joints. Figure 5.18 shows the advantage of including the waist in the set of active joints.

These algorithms can be applied to any type of task, as long as it is force-based: even rising from a chair. A force trajectory is used to represent the body weight and the shoulder's normal position when standing is used as the force goal. The body leans forward to balance its weight (a consequence of Reducing Moment) to reach the shoulder goal.

The average time of a path generation is under 10 seconds. Since our examples mainly involved heavy loads, static torque computations were used. The internal joint chain positions are determined by *Jack* inverse kinematics.

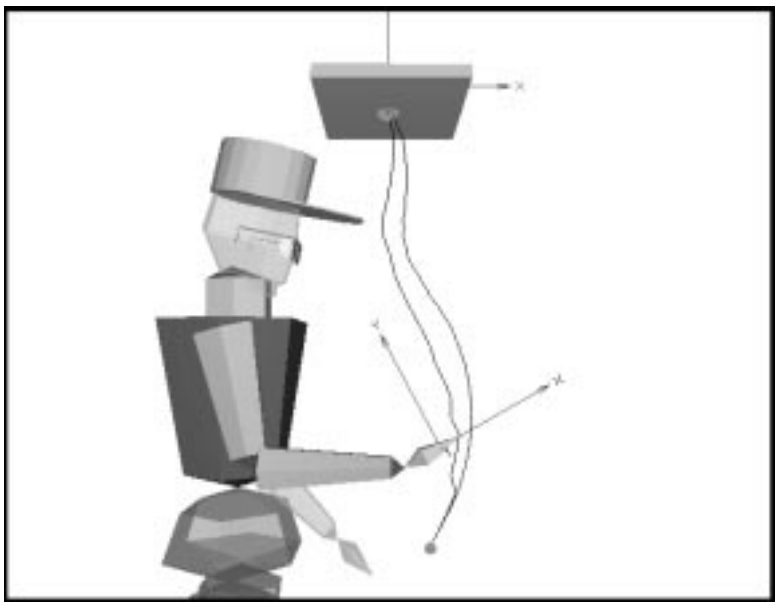

Figure 5.15: Lifting a 20 pound and 10 pound Object.

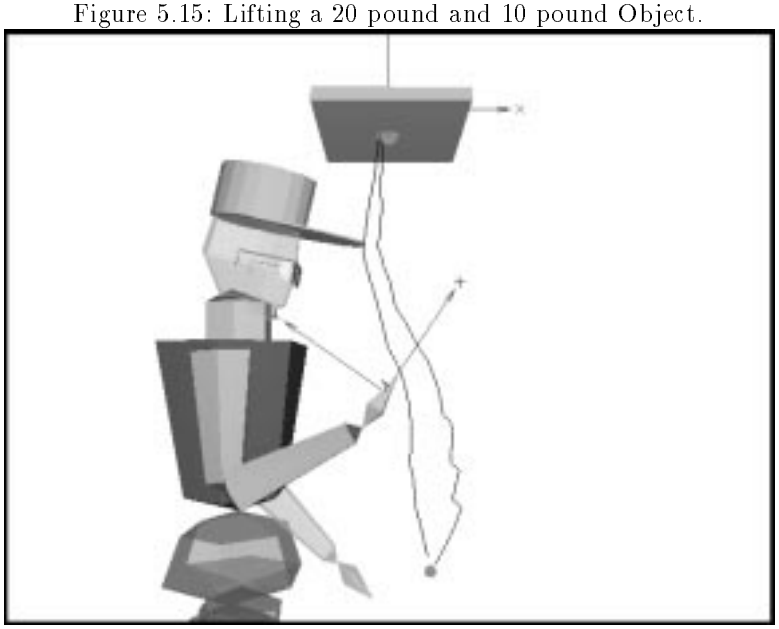

Figure 5.16: Lifting a 30 pound and 20 pound Object.

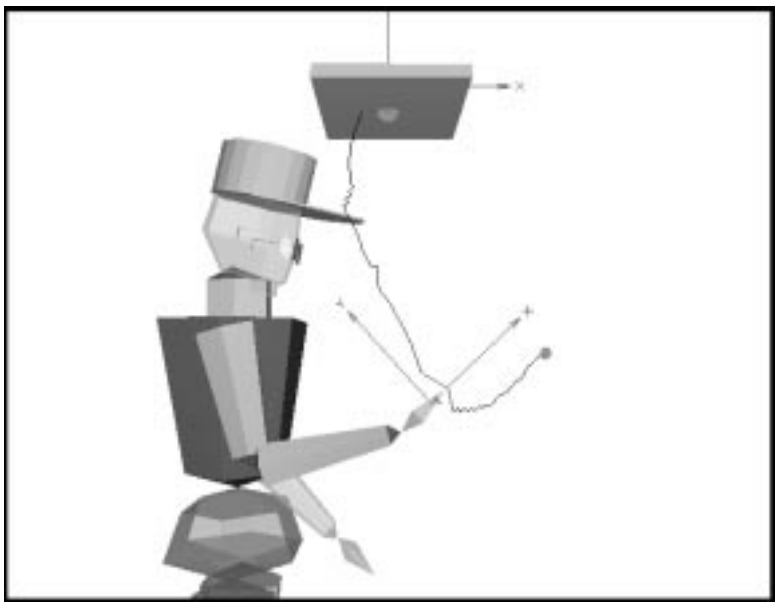

Figure 5.17: Pull Back.

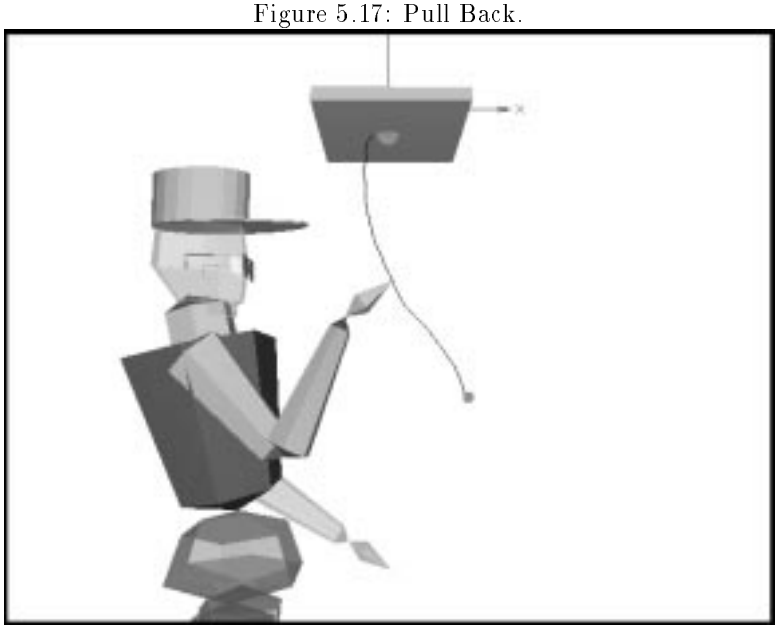

Figure 5.18: Added Joint.

5.3.7 Evaluation of this Approach

We have tried to generate a realistic motion path for a slow, weight-lifting task represented by a time- and position-dependent force trajectory. The method maps out an entire path automatically and incrementally for a force trajectory over a set of constraints based on comfort level, perceived exertion, and strength. Because the body state is constantly updated, these constraints can also be a function of an external model, such as fatigue.

Motion is generated by integrating a condition monitor which suggests basic motion strategies, a path planning scheme which locally plans the end-effector's path and a rate control process which controls the joint rates. The condition monitor offers strategies to pursue by balancing the goal of the task against the resources that are currently available. The path planning scheme proposes a direction of travel by executing the basic strategies. The elusive force function that previous investigators have sought can be found by changing the role of the dynamic equations to a constraint equation which is established with a dynamics model. By selecting the most restrictive constraint from the constraint equations, the maximum joint rates can be computed.

Altering the constraints used in this problem still gives a goal-directed motion that conforms to physical laws. We see this capability as an inherently model-driven, flexible, and crucial component for generating more natural human behaviors. We will see shortly in Section 5.4 how this approach can be extended to include collision avoidance during more complex actions.

5.3.8 Performance Graphs

Three types of graphs are developed for the user to analyze some pertinent mechanical relations of the agent's movements. The properties that are tracked and graphed are comfort levels, work, and energy (Figure 5.19, 5.20, and 5.21).

The histogram of comfort shows the current discomfort value as well as the user-specified desired maximum. The effective comfort level is the actual comfort level that controls the movements. It includes factors that may affect the desired comfort level. Currently, an exponential function representing fatigue is one of the factors that can alter the effective comfort level.

The graph of work the agent performs at each iteration is computed as

$$\Delta W_j = \sum_{i=n}^{n} \tau_i \ \Delta\theta_i \tag{5.20}$$

where n is the number of DOFs, τ_i is the current torque of the ith joint, and $\Delta\theta_i$ is the change in joint position $(\theta_j - \theta_{j-1})$.

The energy graph represents the sum of all the work performed by the joints since the beginning of the task. Because the amount of energy that is expended grows very fast, the curve that represents energy is automatically rescaled to fit within the designated plot area when necessary.

Figures 5.22, 5.23, and 5.24 show the comparative trajectories of the hand as load, comfort level, and perceived exertion are varied.

5.3.9 Coordinated Motion

Coordinated motion is when the execution of a task requires more than one set of linkages either from the same agent or from two separate agents. Our implementation of coordinated motion follows the same parameter control philosophy. As in the task specification of a single linkage, a coordinated task is made a function of comfort and path deviation.

The coordination between the two joint chains is based on a master and slave relation: the weaker chain is the master and the stronger chain is the slave. The algorithm first determines which is the weaker linkage (master), then its path is generated according to the strategy that is appropriate for its comfort level. The other joint chain (slave) must comply with the master's new position. Simple geometry from the master's new position, the dimension of the object, and the magnitude of the slave's incremental displacement is used to determine the slave's new position (see Figure 5.25). The magnitude of the master and slave's incremental path is a nominal distance factored by their respective comfort level; the master will travel less than the slave. The master-slave assignment switches whenever the relative comfort – the difference between the two linkages' comfort level – reverses.

The specification of a coordinated task begins by instantiating each of the joint chains as a separate task. Then the goal and the weight of the object to be moved is stated. Next, the kinematic relation between the two linkages is specified; this involves identifying the attachment points on the object (handle) with the corresponding end-effector. This is done for each of the chains that participates in the coordinated task. The end-effectors are automatically attached to the handle.

Finally, two control parameters which control the characteristics of the path need to be specified. The first control parameter, *allowable comfort difference*, specifies a tolerance in the relative comfort level before a master-slave switch is made. A graph can be called to display the difference in comfort between the two joint chains. The other parameter controls the amount of *allowable path deviation*. The user can control the amount of path deviation by entering a ratio that determines the relative propensity between the path of moving to the goal directly and the path that is sensitive to the agent's stress level. Unlike a task involving a single chain, the motion of each chain cannot deviate or switch strategy without considerating the other chain. This means that the strategies of the two linkages should coincide; both linkages performing Pull-Back would not be permitted. Because of this limitation, the strategies available in this current implementation are only Available-Torque and Reduce-Moment.

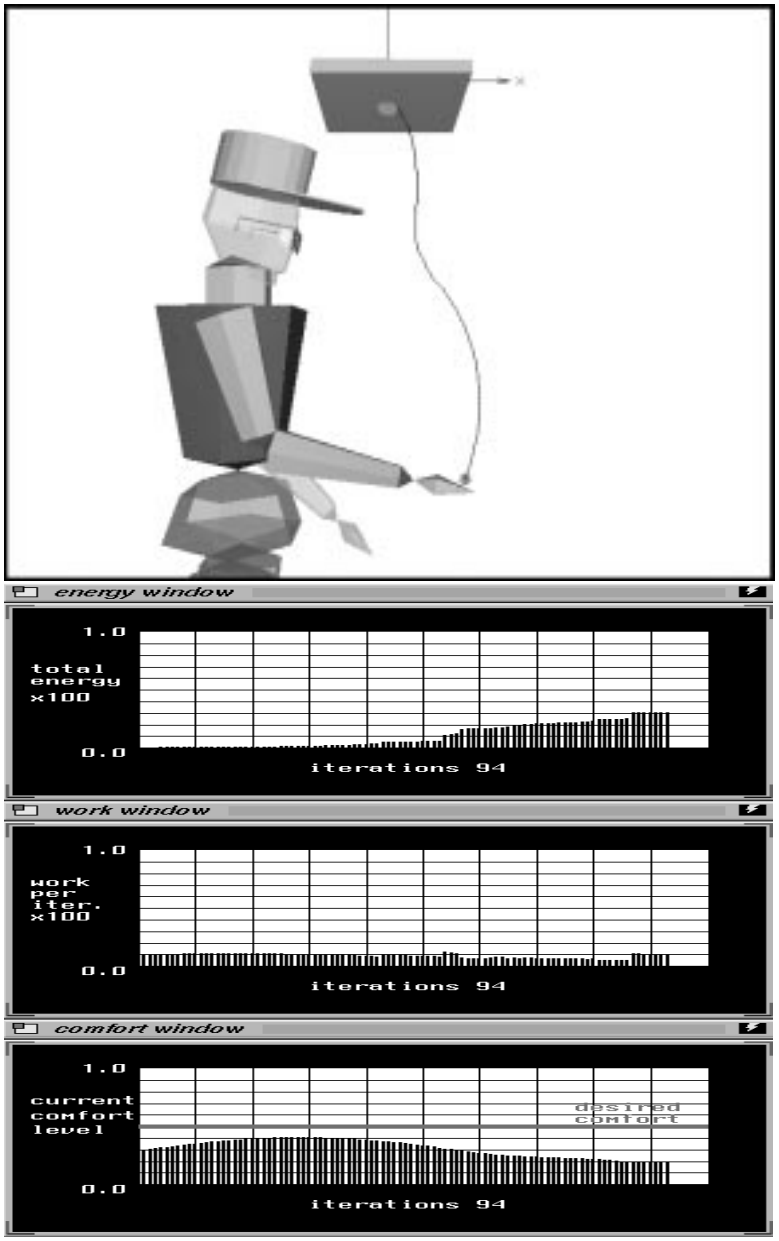

Figure 5.19: Plots of Comfort Level, Work, and Energy (10 lb., pe = 0.30, cl = 0.50)

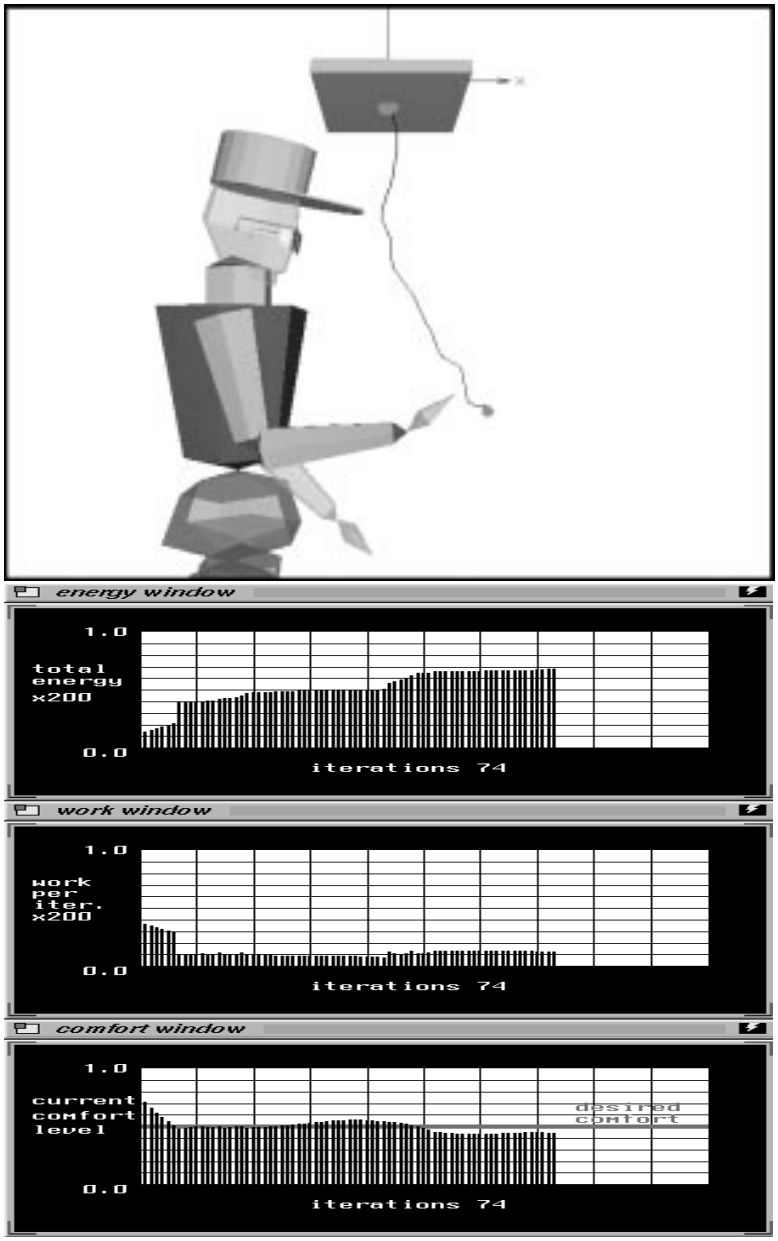

Figure 5.20: Plots of Comfort Level, Work, and Energy (20 lb., pe = 0.30, cl = 0.50)

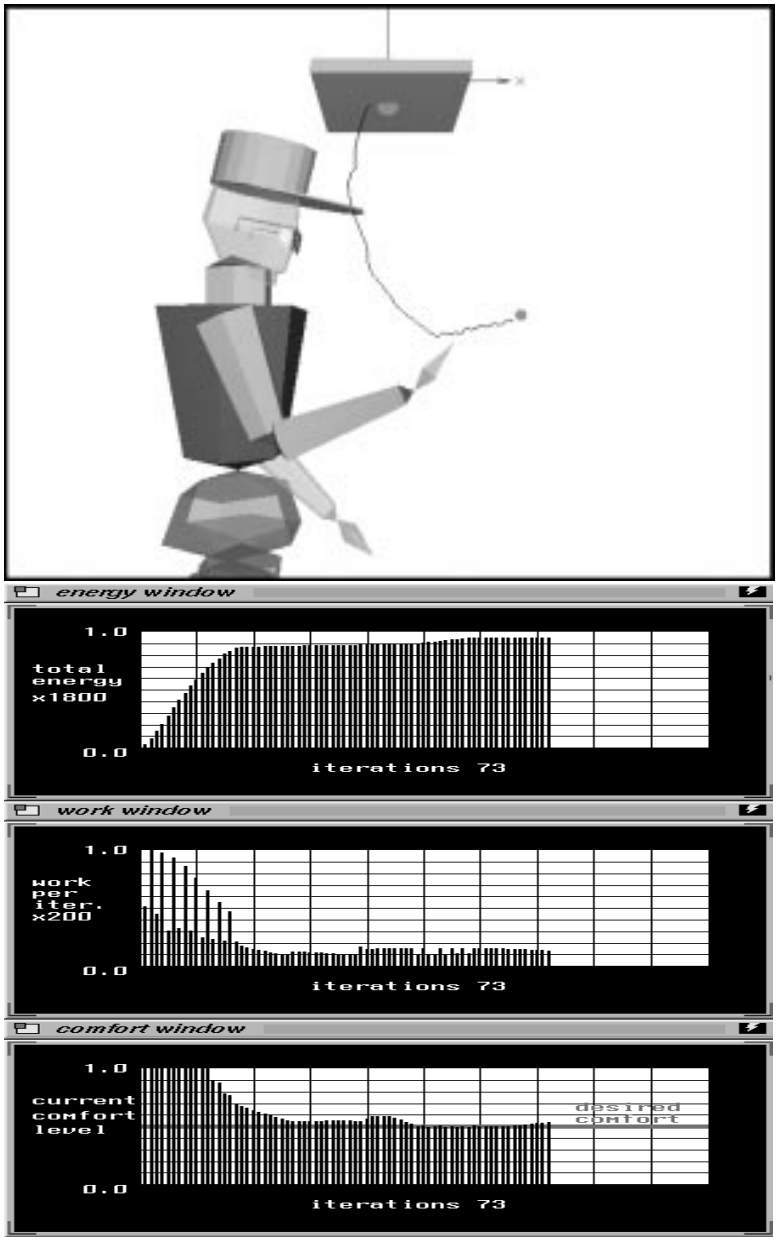

Figure 5.21: Plots of Comfort Level, Work, and Energy (30 lb., pe = 0.30, cl = 0.50)

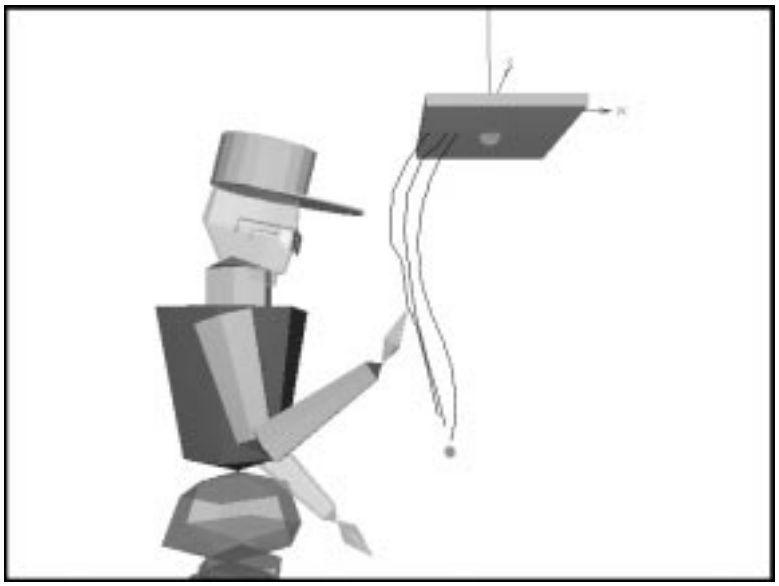

Figure 5.22: 30 lb., pe = 0.30, cl = (0.40, 0.50, 0.85)

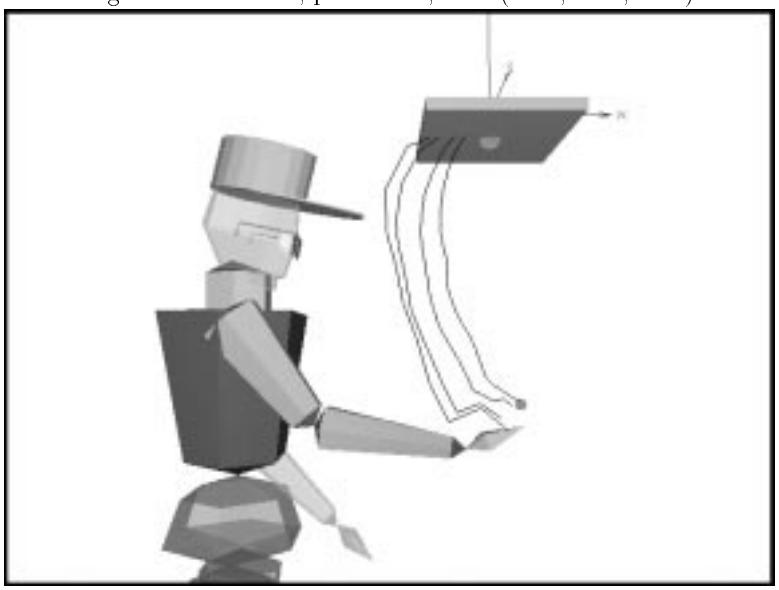

Figure 5.23: 30 lb., pe = 0.30, cl = (0.40, 0.50, 0.75, 0.85)

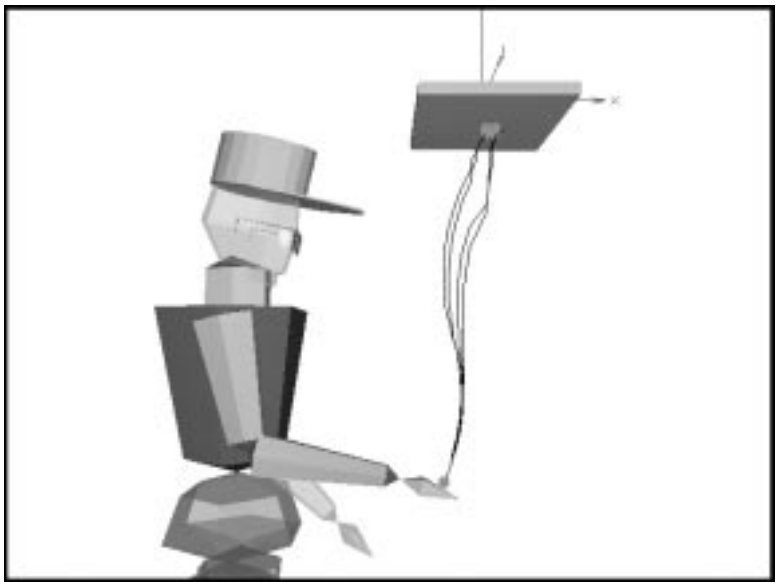

Figure 5.24: 5 lb., pe = (0.30, 0.20, 0.10, 0.05), cl = 0.50

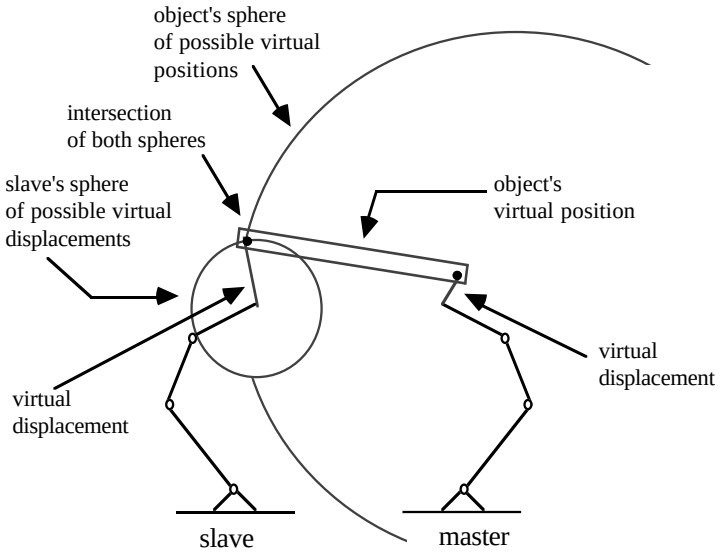

Figure 5.25: Construction of the Path for Coordinated Motion.

5.4 Collision-Free Path and Motion Planning

[4]Collision-free path planning has applications in a variety of fields such as robotics task planning, computer aided manufacturing, human figure motion studies and computer graphics simulations. A collision-free path for an articulated figure is the path along which the articulated figure moves from an initial configuration to a final configuration without hitting any obstacles residing in the same environment as the articulated figure.

A great deal of research has been devoted to the motion planning problem in the area of robotics within the last 10 years, e.g. [LPW79, LP87, LP83, Bro83b, BLP83, Bro83a, Don84, DX89, KZ86, Gou84]. However, despite the applicability of motion planning techniques to computer graphics simulations, the problem has not been addressed much in the computer graphics community [Bre89, DLRG91].

Articulated human figures are characterized by a branching tree structure with many DOFs. Existing algorithms in robotics fall short in handling some of the issues encountered when dealing with these types of figures. We present novel algorithms that can address all these issues. The basic idea is that instead of treating all the DOFs in the figure together, we divide them up into groups and treat these groups one by one and playback the path in a coordinated manner when all the groups are considered. Our motion planning system can also take into consideration the strength data of human figures so that the planned motion will obey strength availability criteria (Section 5.3).

5.4.1 Robotics Background

The major challenge of our problem is to handle a redundant branching articulated figure with many DOFs. Many of the robotics algorithms deal with manipulators with relatively few DOFs, e.g. mobile robots which typically have 3 DOFs or arm-like robots which have 6. Many of these algorithms are based on the use of the configuration space (C space) which is the space encompassing the DOFs of the robot [LPW79, LP83]. The inherent difficulty with this approach is due to the high dimensionality of the C space. It is well known that the worst case time bound for motion planning for a robot arm is exponential in the dimensionality of its C space [SS83a, SS83b]. It is only during the last few years that motion planning algorithms that can handle manipulators with many DOFs have been presented [BLL89b, BLL89a, BL89, Gup90, Fav84, Bre89].

Very few studies consider articulated figures with branches. Barraquand *et al* gave an example involving a manipulator with 2 branches [BLL89b, BLL89a, BL89]. In their work, they create an artificial potential field in the 3D workspace and the free path is found by tracking the valleys. A gain in efficiency is obtained as a result of the clever selection of potential functions and heuristics. However, it is not clear how these can be selected in general.

[4]Wallace Ching.

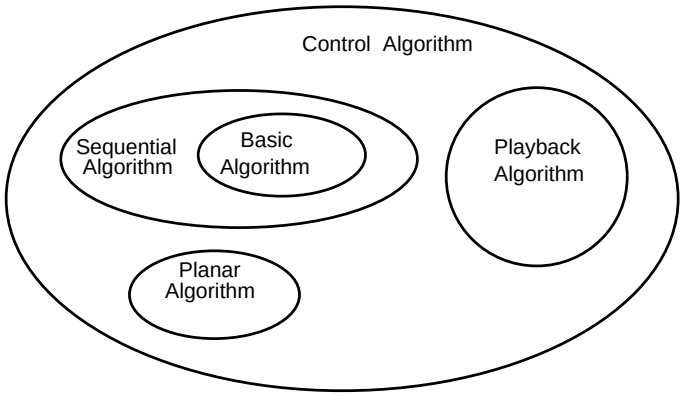

Figure 5.26: Different Modules and their Associated Algorithms in the Path Planning System.

Faverjon *et al* [Fav84] partition the free space into oct-trees and uses some probability measures to cut down the search tree during the A* search. Gupta [Gup90] handles sequential linkages with many DOFs using a sequential search technique which basically treats the individual DOFs one by one instead of considering all of them together. The initial stage of our path planner is based on his work.

5.4.2 Using Cspace Groups

The main idea of our path planner is to handle the DOFs of the articulated figure not all at once but a certain number at a time. The general principle of our path planner is first to divide the DOFs into a number of groups which we call *Cspace groups*. We then compute the collision-free motion for the DOFs in each group successively, starting from the first group. After the motion of the DOFs in group i has been planned, we parameterize the resulting motion with a parameter t. The motion for the DOFs in group $i+1$ will then be planned along this path by controlling the DOFs associated with it. The problem is then posed in a $t \times \theta^k$ space if there are k DOFs in this group. We proceed in this manner along the whole figure structure and solve for the motion for all the groups. Finally a playback routine is invoked to playback the final collision-free path for the figure.

Our system adopts a modular design in that it is made up of a number of modules each of which is based on an algorithm (Figure 5.26). Each module carries out a particular function and contributes to the whole path finding process.

On a more global perspective, the path finding procedure can be viewed as consisting of two phases: the *computation* phase and the *playback* phase. All of the steps involved in these phases are performed by the algorithms described in later sections.

The overall path planning procedure is outlined as follows:

- **Computation** Phase:

  1. Partition the articulated figure's DOFs into *Cspace groups* according to a grouping scheme.
  2. Impose an order of traversal among the Cgroups. For a human figure, we use a depth-first traversal. This means we plan the motion for one arm and then another.
  3. Invoke the *control algorithm* that handles traversal of the tree and finds the final collision-free path. This algorithm will actually call upon a subsidiary algorithm, *sequential algorithm*, to compute the free path along a branch of the tree structure. The *sequential algorithm* will in turn call another subsidiary algorithm, the *basic algorithm*, to compute the path for the DOFs within each Cgroup.

- **Playback** Phase:

  After all the Cspace groups have been considered, a special *playback algorithm* will be called upon to traverse the tree structure in a reverse order, collect and coordinate all the computed information and finally playback the overall collision-free path in discrete time frames. These time frames can be further interpolated to produce smooth motion. Ideally the behavioral simulation loop controls the frame timing and hence the production of the output postures.

The translational movement of the articulated figure as a whole on a plane can also be generated with this planner. In this case, the figure resembles a mobile robot with two degrees of translational freedom and one degree of rotational freedom. The module that handles this case is named the *Planar Algorithm*.

Figure 5.27 shows the general redundant branching articulated structure and symbols that we will use for reference. We will mainly focus on the upper body of the human figure. The system can be easily applied to the legs to provide stepping or foothold movements as we will show later.

5.4.3 The Basic Algorithm

The particular algorithm we have chosen is the one presented by Lozano-Pérez in [LP87] due to its simplicity and intuitiveness. It first constructs the C space for the articulated figure. For the sake of completeness, the process is described below.

If the manipulator has n links, its configuration space can be constructed as follows:

1. $i = 1$.
2. While $(i < n)$ do

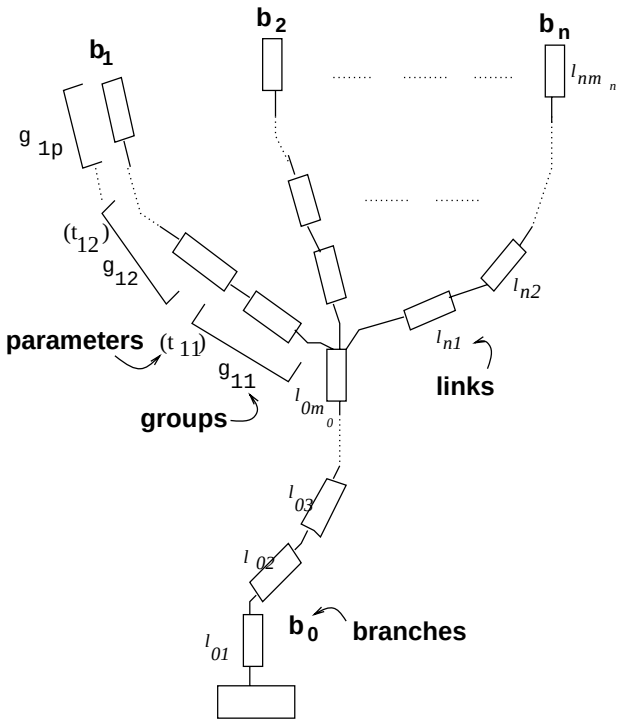

Figure 5.27: The Redundant Branching Articulated Figure.

(a) Ignore links beyond link i, and find the ranges of legal values of q_i by rotating link i around the position of joint i determined by the current value ranges of $q_1, ..., q_{i-1}$ and check for collision with the surrounding obstacles. Self collision can be avoided by checking collision with linkages from the same figure as well. Mark those joint values at which link i will have a collision as forbidden.

(b) Sample the legal range of q_i at the specified resolution.

(c) Increment i and repeat step (a) for each of these value ranges.

The free space is then represented by a *regions* data structure to explore the connectivity between the cells. A graph is then built on these region nodes and an A* search is conducted to search for a free path from the start node to the goal node.

5.4.4 The Sequential Algorithm

The *Sequential Algorithm* handles the motion planning problem for the Cspace groups along a sequential branch. This algorithm is based on but differs from Gupta's work.

Referring to Figure 5.28, let n be the total number of *Cspace groups* on this branch. Let the joint DOFs associated with the groups be represented by q_{ij} where i is the group number and j is from 1 to m_i where m_i is the maximum number of DOFs group i has. Let r_i be the reference vertex for group i. It is basically the distal vertex of the link associated with the DOFs in the group i. Let $r_i(t)$ denote the trajectory of the reference vertex r_i. The initial and goal configurations of the arm are given as q_{ij}^s and q_{ij}^g, i=1..n; j = 1..m_i.

The algorithm is as follows:

1. Compute a collision-free trajectory for the links associated with group 1. The trajectory of the reference vertex on its link will be $r_1(t)$.
2. $i = 2$.
3. While ($i < n$)
   (a) along $r_{i-1}(t)$, discretize the path according to a pre-specified resolution. Compute a collision-free trajectory for the DOFs in the ith group from q_{ij}^s to q_{ij}^g for $j = 1..m_i$ using the *basic* algorithm.
   (b) given $q_{1j}(t), q_{2j}(t), ..., q_{ij}(t)$, compute $r_i(t)$ using forward kinematics.
   (c) Increment i.

The parameter used in parameterizing the path already computed can be either interpreted as temporal or non-temporal. For a temporal interpretation of the parameter, the path computed has to be monotonic with respect to the parameter t simply because we cannot travel backward in time. Hence backtracking within the Cspace group is not allowed and the chance of finding a path is greatly restricted. In the example shown in Figure 5.29, we will not be able to come up with a path without backtracking. We have adopted a non-temporal interpretation of the parameter in most cases as this will increase the chance of finding a path.

Each C group deals with one parameter and a certain number of DOFs. The number of DOFs can vary between C groups so as to fit into the structure of the figure. For example, the shoulder joint can be handled by one C group with 3 DOFs.

The number of DOFs handled at a time also affects the degree of optimality of the resulting path (with respect to some criteria). Theoretically, the optimal path can only be obtained by searching through the n-dimensional C space built from considering all n DOFs together. However, such an algorithm has been proven to be exponential in the dimensionality of its C space [SS83a]. There is a customary trade off between speed and optimality.

Our choice of using the region graph instead of the visibility graph allows for the path to be positioned farther away from the obstacles, hence leaving more room for the next linkage (Figure 5.30).

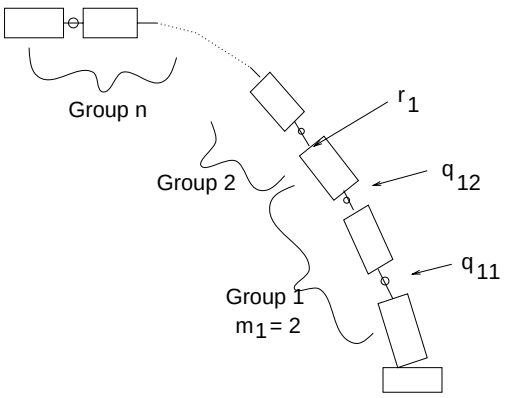

Figure 5.28: A Sequential Linkage.

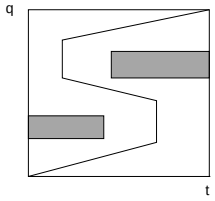

Figure 5.29: An Example Showing the Case that a Path can only be Found with Backtracking which Means the Parameter Takes on a Non-Temporal Interpretation.

5.4.5 The Control Algorithm

The control algorithm performs the entire path planning process:

1. Apply the Planar Algorithm to the whole figure to obtain the planar collision-free translational movement of the figure taken as a whole.
2. Parameterize the resulting motion.
3. Repeat the following to every branch, starting from the first.
   (a) Apply the *Sequential* algorithm to the branch using the last parameter in the first group.
   (b) Parameterize the resulting path computed for this branch according to some prespecified resolution.
   (c) Invoke the PlayBack Algorithm to the branch to obtain the sequence of joint angle values of the branch when moving along the computed path.
   (d) Record this sequence of joint angles in the array $FREEANGLES$.

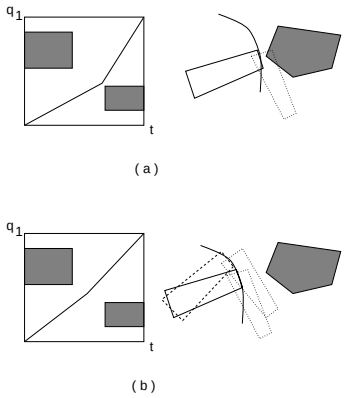

Figure 5.30: (a) A Path that is Too Close to an Obstacle. This Leaves Little Room for the Next Linkage to Maneuver. (b) A Better Path that is Farther Away from the Obstacle.

4. Apply the PlayBack Algorithm to the whole figure, starting from the last group of the very last branch.

5. The angle values obtained can then written into frames for continuous playback.

5.4.6 The Planar Algorithm

The articulated figure can translate and rotate on a plane, navigating around obstacles. The whole figure behaves just like a mobile robot. The path planning algorithm in this case deals with a three dimensional (2 translational, 1 rotational) Cspace. We can handle this case simply with our *basic* algorithm or other existing mobile robot path planning techniques.

5.4.7 Resolving Conflicts between Different Branches

Although the different branches are attached to the same rear link of branch b_0, we do not use the same parameter t that parameterizes the motion of branch b_0 in all these branches. The reason is that the parameters t_{ij} are interpreted as non-temporal in general. Hence, backtracking within the Cspace group is allowed and the values of t_{ij} along the computed path can be non-monotonic. If we use the same parameter in computing the motion for the first groups in all other branches, some of the joint angle values cannot be obtained uniquely during the final playback phase. This reasoning may become clear after looking at the playback algorithm.

Our solution to this problem is to further *parameterize* the already *parameterized path* of the previous branch and then use the new parameterization variable in the next branch.

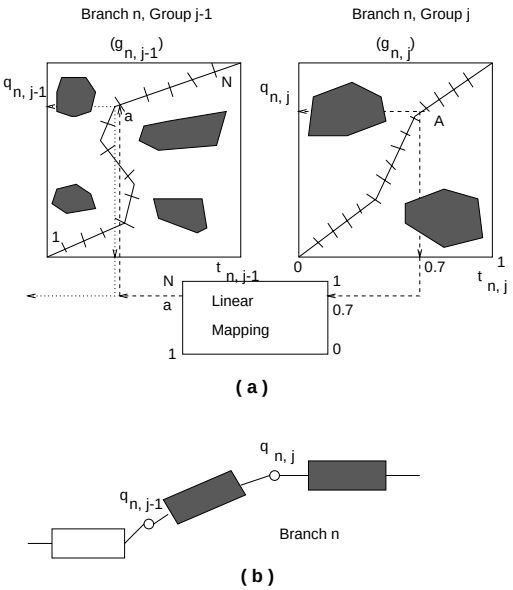

Figure 5.31: An Example Showing How the Final Joint Angle Values of the Whole Figure are Obtained from the Cspace Associated with the Cspace Groups.

5.4.8 Playing Back the Free Path

During the playback phase, we start from the *last group* of branch b_n and then traverse the branches in a backward manner along branch b_{n-1}, b_{n-2} and so on and finally to branch b_0. For example, let Figure 5.31 (a) represent the configuration space for the *last group* of the last branch, i.e. group g_{n,p_n} of branch b_n. We then discretize the free path according to a pre-specified playback resolution. The number of discretization intervals for this last group will be equal to the number of *time frames* for the final simulation.

At every discretized point, say A, there is a corresponding (q,t) pair: the q value is what we should set the last joint DOF to, and the *parameter* t is used to deduce the motion of the preceding group. We first set the last DOF to the q value. Then we use the parameter t in the pair to trace back to the *preceding* (proximal) group. Note that within this preceding group, the parameter t is *monotonic* by definition. Hence we can uniquely determine the corresponding (q,t) pair within this preceding group. By the same token, we can continue tracing back to the groups further preceding this one (Figure 5.31 (a)). We carry on in this fashion recursively until we come to the first group within this branch.

Note that at this point, all joint DOFs along this branch will have been set to their correct value for this simulation time frame. The sequence of joint values along the free path for all the other branches should have also been

recorded in the array $FREEANGLES_i$. The parameter value left unused will then be used as an index into the recorded joint angle array and to uniquely determine the set of angles corresponding to the movement of the preceding branch. The rest of the branches are processed similarly.

We will examine the *playback* algorithm by looking into its two components separately: the *Final Playback* algorithm and the *Single Branch Single Frame Playback* algorithm.

The Final Playback Algorithm

The Final Playback algorithm is driven by the behavior simulation cycle and generates the intermediate body postures culminating in the final goal achievement.

- Discretize the path computed for the last group in the last branch into N discrete points according to some pre-specified resolution. This number also determines the total number of key postures or time frames we will generate for the final simulation.
- For each time frame, do the following steps to get back the computed angles.

    1. Apply the **Single Branch Single Frame Playback Algorithm** to branch b_n, the last branch in the figure.
    2. A parameter value will be obtained at the termination of this algorithm. Use this parameter as an array index into $FREEANGLES$ for the next branch. The joint angles recorded for the next branch will be read from the array element pointed to by this parameter value.
    3. Set the joint angles in the next branch to the values read from the array.
    4. The last parameter value read from the array is used to index into the $FREEANGLES$ array for the next branch in a similar manner.
    5. Repeat the same process for the rest of the branches.
    6. Now all the joint angles in the articulated figure have been set to their appropriate values in this time frame. What is left is the *position* of the whole figure. The last parameter value obtained from the last step is used to index into the path computed from the Planar Algorithm. Then we set the whole figure location to that indexed position.

The Single Branch Single Frame Playback Algorithm

Let the branch index we are considering be i. Here branch i has a total of p_i groups. This playback algorithm is called only after the motion for the

last group in the branch is computed. This algorithm only deals with one discretized point, and hence only one time frame.

We start from the last group in the branch and go *down* the branch by doing the following on each group.

1. From the discretized point on the computed path, read the values of the $q_{i,j}$s associated with this Cspace group from the axes of the Cspace. This is illustrated in Figure 5.31 (a) with a 2D Cspace as an example.

2. Set the joints in the articulated chain corresponding to these q variables to the values just found.

3. Then read the normalized parameter value $t_{i,j}$ from the t axis.

4. Through a linear mapping, obtain the corresponding discretized point on the path computed for the next group down the branch from this parameter value.

Note that after this algorithm terminates, all the joint angles on this branch will be set to the appropriate values for this simulation time step.

5.4.9 Incorporating Strength Factors into the Planned Motion

In (Section 5.3) we demonstrated that realistic animation of lifting motions can be generated by considering the strength of the human figure. The basic premise of the method is that a person tends to operate within a *comfort* region which is defined by the amount of available torque. The static torque values at all the joints required to sustain a load is a function of figure configuration. The path planner incrementally updates the next joint angle values according to the *available torque* at the current configuration based on a number of motion strategies.

The planning methodology we have described so far divides the DOFs into groups and plans the motion for each group sequentially. Therefore, only after the control algorithm terminates do we have the complete path for each DOF. However, we need to make use of the strength information *during* the planning process. This requires values of all joint angles at a certain configuration. The solution is to use the value of the current parameter and project it over to the rest of the angles that have not yet been computed. The projection function $PROJ$ is arbitrary (since this is only an approximation of reality), so we use just a simple linear interpolation:

$$PROJ(t) = \theta_{initial} + t(\theta_{final} - \theta initial) \quad .$$

Since our searching process uses the A* algorithm, a heuristic function is evaluated at every step to find the node that has the smallest value. So far we have been using only a spatial distance measure in this heuristic function. The path found will be optimal up to the size of the *regions* used. Now,

however, we have a means to compute the required torque value of a particular configuration, so this heuristic function can include terms representing strength information. The weights attached to these terms represent the relative importance of the quantities they represent. Possible terms to include are:

- **Work Done**. The total work done or energy expended can be measured by the term $\int T(\vec{\theta}) \cdot d\vec{\theta}$. The integration is done over the path taken.
- **Comfort**. The *comfort level* of the resulting motion can be measured by the *available torque* which is the quantity obtained by subtracting the required torque from the strength limit at that particular joint configuration. We can sum up all the contributions along the path as $\int AvailTorque(\vec{\theta}) \cdot d\vec{\theta}$ where the available torque is defined in terms of its elements:

$$AvailTorque(\vec{\theta})_i = \begin{cases} Str(\vec{\theta})_i - T(\vec{\theta})_i & \text{if } Str(\vec{\theta})_i > T(\vec{\theta})_i \\ 0 & \text{otherwise} \end{cases}$$

  The subscript i stands for the i-th element in the vector. Str is the strength limit vector. This integral value will then represent the overall comfort level.

  This term will properly be useful only in the g function as it only affects future actions.

- **Fatigue**. Humans are not like robots: our strength will decrease with time as a result of fatigue. We may include a term such as $\int \|T(\vec{\theta})\| dt$ to avoid taking a path that has a high torque value maintained over a prolonged period of time.

The path found by the collision-free path planner is the best up to the size of the *regions* (the basic entities). Paths within regions are chosen by the strength guided motion heuristics. For example, in Figure 5.32 (a), the left path may be chosen by search to be better than the one on the right. This path can then be further refined by examining local comfort levels and invoking one of the motion heuristics such as *Available Torque, Reducing Moment* and *Pull Back*.

5.4.10 Examples

We have experimented with a variety of reaches involving shelves and apertures. Figure 5.33 shows a human figure reaching through two apertures with both arms. The path computed is collision-free and involves more than 20 DOFs. These and similar examples take about 12 to 20 minutes elapsed time to compute, depending on the complexity of the environment.

Simulating the wide range of human motions requires a number of different behavioral skills such as walking, grasping, and lifting. The path planner

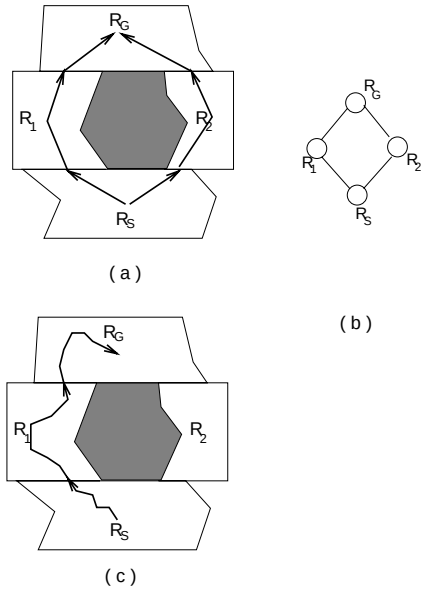

Figure 5.32: (a) Part of a Sample Cspace Showing Two Possible Paths Leading from the Start to the Goal Node. (b) The Corresponding Regions Graph. (c) Path Refinement after Considering Comfortable Motion Heuristics.

interfaces with other existing techniques in simulating more complex human behaviors. Figure 5.34 shows a human figure climbing up a rocky surface. The climbing movement of each limb and the torso translation are produced by the path planner. Each limb is considered in turn by the planner; the other three limbs are fixed by a point (reach) constraint. The spatial locations for each hand grasp and foothold must be pre-specified by the user. Even though our path planner cannot handle closed loop systems directly, such motions can be simulated with the help of other behaviors.

5.4.11 Completeness and Complexity

This planner is an approximate algorithm when backtracking among groups is not employed: it may fail to discover a path even though one exists. When backtracking is employed between groups, the algorithm is turned into a complete algorithm. Alternatively, the DOFs can be re-grouped differently if a path cannot be found for the previous grouping scheme.

The *basic* algorithm within a Cspace group has complexity $O(r^{k-1}(mn)^2)$ where k is the number of DOFs, r is the discretization interval, m is the number of faces and edges for the figure and n for the environment [LP87]. Since the number of DOFs in a Cspace group is bounded, the run time for the *basic* algorithm can be treated as a constant. Consequently the whole algorithm runs in $O(p)$ time where p is the total number of groups in the tree

structure without backtracking between groups. With backtracking, the worst case run time for the algorithm is exponential with respect to the number of DOFs. This is the same as a conventional exhaustive search algorithm. We believe that the average run time of the algorithm is fast enough for practical though not interactive use.

5.5 Posture Planning

[5]Motion generation algorithms for geometric figures typically start with initial, final, and sometimes intermediate configurations or postures. From these input postures, "natural" motion sequences are generated which satisfy given optimization criteria, e.g., time or energy. The assumption that the final joint angle configuration of a highly redundant articulated figure such as the human body is known – in advance – is rather unrealistic. Typically, only the task-level goals of the end effectors are known. In order for an agent to move in accordance with such task-level goals, we need to provide the agent with an ability to plan both intermediate and final postures. *Posture planning* uses explicit reasoning along with numerical procedures to find a relatively natural and collision-free sequence of postures that terminates at the desired goal.

The inputs to the posture planner are positional or orientational goals for end effectors. The posture planner finds the appropriate movements of relevant body parts needed to achieve the goals. It discovers a final global posture satisfying the given goals by finding intermediate motions that avoid collision. Typical motions used in the plan include stepping towards the workspace, spreading the legs, bending the upper body at the waist while the whole body remains balanced, and moving the upper arm with respect to the shoulder. Only the geometric aspects of body motion are considered at this time. The agent is assumed to be initially located in the vicinity of the target object so that an end effector goal can be achieved by taking only one or two steps. It is assumed that a given goal is not changed during the posture planning process. Collision-avoidance will be alluded to as necessary but the details are not addressed here [Jun92].

The fundamental problem in achieving postural goals is controlling the massively redundant skeletal DOFs. There are 88 DOFs in the *Jack* model (not counting fingers). Figure 5.35 shows a simplified tree containing only 36 DOFs that are necessary for gross motion. This excludes the vertebrae which account for most of the remaining freedom. To solve the redundancy problem, we should reduce the relevant DOFs to a manageable set at any given moment during motion planning. We use constraints to supplement the traditional methods of robot motion planning in joint space done solely using numeric computations [LPW79, LP81, LP83, BHJ+83, LP87, Bro83b, Bro83a, Don87, Kha86, MK85, BLL89a, Bre89, CB92]. We reduce the complexity of the numeric computations by applying cues from qualitative knowledge and reasoning.

[5]Moon Jung.

Figure 5.33: A Human Figure Reaching Through Two Apertures with Both Arms.

Figure 5.34: A Human Figure Climbing up a Rocky Surface. This Animation is Created with a Hybrid of Simulation Techniques.

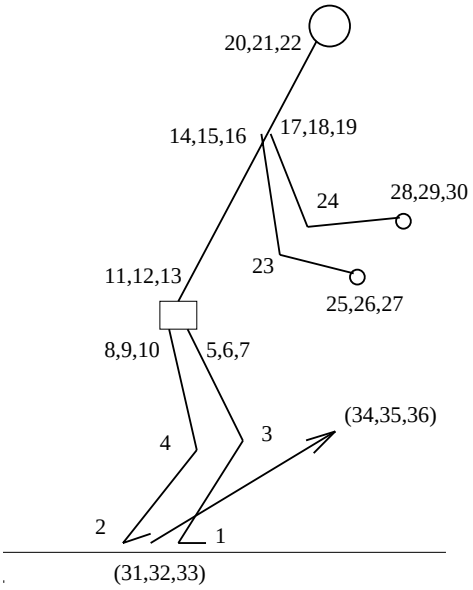

Figure 5.35: The Reduced Degrees of Freedom of the Human Body.

Here are the basic steps needed to determine postures needed to achieve a given end effector goal.

1. Postulate primitive motions for the end effector goal considering motion dependencies and a *minimum disturbance constraint*. For example, a motion dependency causes the palm to move forward and downward when the upper body is bent forward about the waist. While deciding on movement of body parts, the body is subjected to a constraint which requires minimal movements. For example, the agent does not bend if a relevant object can be grasped just by extending the hand. Primitive motions are defined in terms of salient parameters defined in the 3D Cartesian *task space*. These parameters are called *control points* and *control vectors*. They are points and vectors defined on parts of the body that allow the conversion from task-level goals into an intuitive subset of body DOFs.

2. Use *mental simulation* to bind parameters of primitive motions and to detect collisions. The body is subject to inherent constraints such as one that requires that the body remain balanced when a task is performed.

3. If the mental simulation indicates that a body part would collide with an obstacle, identify collision-avoiding spatial constraints or suggest intermediate collision-avoiding goals by reasoning about the qualitative geometric relations among the body part, the goal placement, and the obstacle.

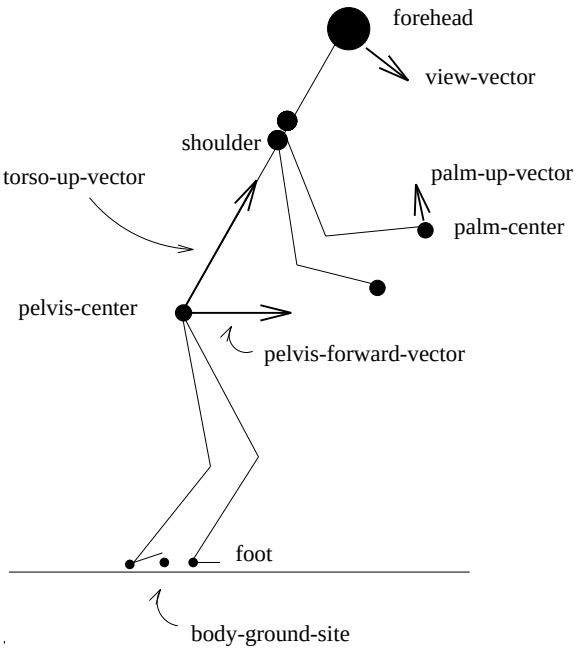

Figure 5.36: Task-Level Control Parameters: Control Points and Vectors.

4. If the postulated motions violate already achieved goals or collision-avoiding constraints, use the amount of violation to replan and compensate for the violation.

5.5.1 Functionally Relevant High-level Control Parameters

To control body postures effectively, we represent spatial configuations of the body in terms of "lumped" control parameters: control points and control vectors (Figure 5.36). Control points are points defined on important body parts such as feet, pelvis, head, and hands) to control their positions. Control vectors are vectors defined to control orientations. The task-space control points or vectors are considered to be the reduced set of DOFs of the body with respect to the task-space. The number of task-space DOFs that we consider is 14.

A control point or vector moves relative to a *base site* of the control point or vector. For example, the palm-center may be moved with respect to a base site at the shoulder, the pelvis-center, or the body-ground-site. Or it may be moved as the whole body moves with respect to the base site outside of the body, that is, the origin of the world coordinate frame. The pelvis-forward-vector controls the horizontal orientation of the body and so is confined to rotate along the horizontal circle centered at the pelvis-center. The torso-

up-vector controls the forward/backward orientation of the upper body. Note that the rough configuration of the body is defined by the four *primary* control parameters: body-ground-site, pelvis-center, pelvis-forward-vector, and torso-up-vector.

5.5.2 Motions and Primitive Motions

Motions are defined by specifying control parameters and the directions and the distances to move them. We use the three standard directions along which to move or rotate: sagittal (forward/backward), vertical (upward/downward), and sideward (leftward/rightward). These relative directions are considered unit vectors defined with respect to the body-centered coordinate frame. When we say that a control point moves *forward*, it means that the movement of the control point has the *forward* component. It may have other motion components. For example, we use the motion goal *move-by(forehead, downward, Dist)* to refer to a goal of moving the forehead downward by distance *Dist* and *orient-by(torso-up-vector, forward, Angle)* to refer to a goal of rotating the torso-up-vector forward by angle *Angle*. A motion along a given direction is decomposed into the three components along the standard directions. This is convenient for two reasons. First, the component motions can be independently planned, although interference among them must be taken into account. Second, it is easy to find cooperative motions that contribute to a given standard motion component.

Note that motion *move-by(forehead, downward, Dist)* does not specify the base site relative to which the forehead is to move. The forehead may move relative to the pelvis-center (by bending the torso-up-vector forward), or relative to the body-ground-site (by lowering the pelvis-center down), or do both in sequence or parallel. A motion which specifies the base site of a control point or vector is called a *primitive motion*. Examples of primitive motions are given in Table 5.1. Primitive *move-by(left.palm-center, downward, Dist, pelvis-center)* means that the agent moves the left palm center downward by distance *Dist* with respect to base site *pelvis-center*. When the body-ground-site is moved, its base site is the world origin *world*. For example, *move-by(body-ground-site, forward, Dist, world)* means that the body-ground-site is moved forward by distance *Dist* with respect to the world origin.

The consequences of a primitive motion are not specified in terms of precondition-action-effect rules. Rather they are computed by incremental motion simulation by taking advantage of the underlying geometric properties of the body and the body balance constraint. Enumerating effects of primitive motions under different conditions is simply not feasible in the continuous domain of motion planning with massively redundant bodies.

5.5.3 Motion Dependencies

A positional goal is translated into a conjunction of simultaneous goals. For example, given a positional goal *positioned-at(right.palm.center, GoalPos)*,

Table 5.1: Primitive Motions for Manipulating Control Parameters.

move-by(left.palm-center, downward, Dist)
move-by(left.palm-center, downward, Dist, shoulder)
move-by(left.palm-center, downward, Dist, pelvis-center)
move-by(body-ground-site, forward, Dist, world)
orient-by(pelvis-forward-vector, leftward, Ang)
orient-by(pelvis-forward-vector, leftward, Ang, pelvis-center)
orient-by(pelvis-forward-vector, leftward, Ang, body-ground-site)

the difference vector from the current position of the palm center to the goal position *GoalPos* is decomposed into the forward, downward, and rightward component vectors, with their distances being F, D, and R respectively . That is, positional goal *positioned-at(right.palm-center, GoalPos)* is translated to the conjunction of the three component motion goals: *move-by(right.palm-center, forward, F)*, *move-by(right.palm-center, downward, D)*, and *move-by(right.palm-center, rightward, R)*.

There are several ways to achieve each component motion goal. They are determined based on the motion dependencies among different body parts. More precisely, they are relationships among control parameters described as follows:

1. move-by(right.palm-center, forward, Dist) has four contributors:

   (a) move-by(right.shoulder, right.palm-center, forward, D1),

   (b) rotate-by(pelvis-forward-vector, leftward, Ang2),

   (c) rotate-by(torso-up-vector, forward, Ang3, pelvis-center),

   (d) move-by(body-ground-site, forward, D4, world),

   such that Dist = D1 + D2(Ang2) + D3(Ang3) + D4, where D2(Ang2) is the distance that the right-palm-center travels due to the rotation of the pelvis-forward-vector leftward by angle Ang2 and D3(Ang3) is the distance that the right-palm-center travels due to the rotation of the torso-up-vector forward by angle Ang3. In general, these distances are dependent on the body context, the situation in which the body is placed. Hence it would be extremely hard to compute them analytically in advance without incremental simulation.

2. move-by(palm-center, downward, Dist) has two contributors:

   (a) move-by(palm-center, downward, D1, shoulder),

   (b) move-by(shoulder, downward, D2)

   such that Dist = D1 + D2.

3. move-by(shoulder, downward, Dist) has two contributors:

   (a) rotate-by(torso-up-vector, forward, Ang1, pelvis-center),

   (b) move-by(pelvis-center, downward, D2, body-ground-site)

   such that Dist = D1(Ang1) + D2, where D1(Ang1) is the distance that the shoulder travels due to the rotation of the torso-up-vector with respect to the pelvis-center by angle Ang1.

4. move-by(palm-center, leftward, Dist) has four contributors:

   (a) move-by(right.shoulder, right.palm-center, leftward, D1),

   (b) rotate-by(pelvis-forward-vector, leftward, Ang2),

   (c) rotate-by(torso-up-vector, leftward, Ang3, pelvis-center),

   (d) move-by(body-ground-site, leftward, D4, world),

   such that Dist = D1 + D2(Ang2) + D3(Ang3) + D4, where D2 is the distance that the right-palm-center travels due to the rotation of the pelvis-forward-vector leftward by angle Ang2 and D3 is the distance that the right-palm-center travels due to the rotation of the torso-up-vector leftward by angle Ang3.

How much to move or rotate the contributing control parameters to achieve a given goal is not part of what we call *motion dependencies*. They are determined by mental simulation.

5.5.4 The Control Structure of Posture Planning

The control structure of posture planning (Figure 5.37) is described. The planner maintains the partial plan (the plan under construction that has unordered goals and motions) and the list of constraints on goals or motions. The partial plan is initialized to the input goals: [*positioned-at(palm-center, GoalPos), aligned-with(palm-up-vector, GoalVector)*], which are subject to the proximity constraints. The **motion postulator** is invoked to suggest a sequence of primitive motions needed to achieve the current goal. The motions are first postulated without considering collisions, while respecting the previously identified constraints. When the motions are postulated, the **motion simulator** selects motions to achieve the positional goal. To do so, the motion dependencies among body parts are used. They permit many alternative ways of achieving a given goal. They are partly constrained by the *minimum disturbance* constraint which prescribes that body parts farther away from the end effector or the upper body are to move only when movements of body parts nearer the end effector or the upper body are not sufficient to achieve a given postural goal. The minimum disturbance constraint is procedurally incorporated into the motion postulator. To determine the values of distance parameters, selected motions are simulated by incrementally changing the

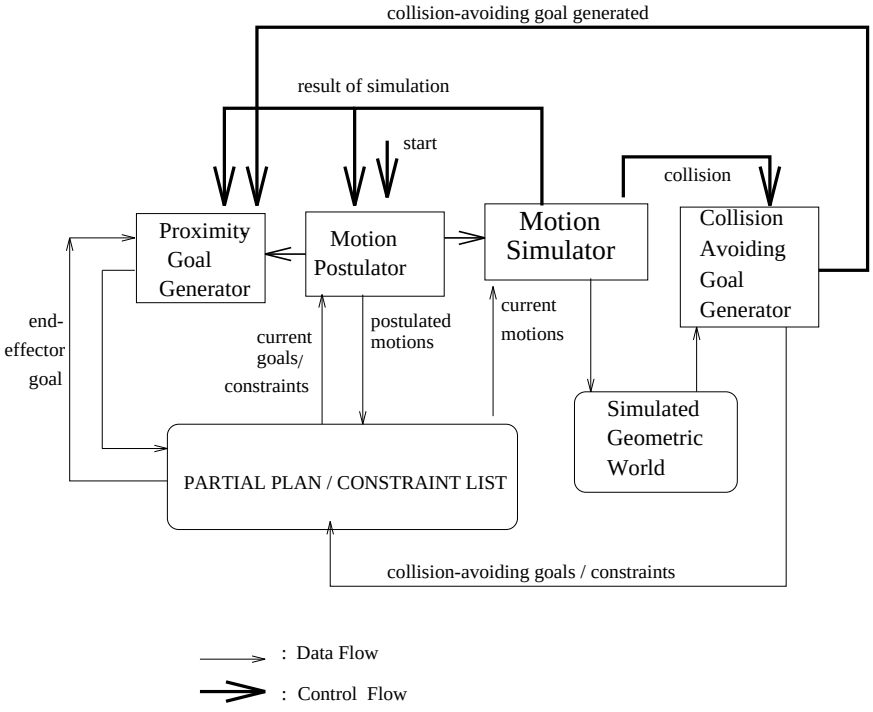

Figure 5.37: The Overall Control of Planning.

control parameters until the goal values are reached. The incremental step size for each control parameter is assumed to be fixed.

When the motions are determined, the **motion simulator** is again invoked to detect potential collisions of important body parts, hand(s), the upper body (including the head), and the lower body. Collision detection of important body parts seems sufficient for the purpose of finding a gross motion plan. If collisions are detected, the **motion simulator** marks the faces of the obstacles with which body parts collided for the first time and continues the simulation, even with object penetration, until the selected motions are fully achieved. This reveals all potential body part collisions and provides more lookahead without significant extra cost. If collision has occurred during the motion simulation the planner calls the **collision-avoiding goal generator** to generate intermediate goals. The achievement of the collision-avoiding goals is then planned by the **motion postulator**.

5.5.5 An Example of Posture Planning

The process of posture planning is best described using an example. Consider a positional goal *positioned-at(right.palm-center, GoalPos)*. Assume that the proximity constraints which the positional goal is subject to are already

achieved and are maintained. For simplicity of exposition, the end effector goal is assumed to have only the *forward* and *downward* components, ignoring the *sideward* component. Consider the situation in Figure 5.38.

1. Translate the input goal *positioned-at(right.palm-center, GoalPos)* to its contributor motions: *move-by(right.palm-center, forward, F)* and *move-by(right.palm-center, downward, D)*, where the goal vector from the palm center to goal position *GoalPos* is decomposed to vectors F and D.

2. According to the minimum disturbance constraint, the shoulder is tried first as the base joint of the palm-center motions. This will be done if the distance between the shoulder and the goal position is less than the arm length. Otherwise, the shoulder should be first moved so that the distance between the new shoulder position and the goal position will be less than the arm length. A plausible goal position of the shoulder can be simply found as shown in Figure 5.39. To find it, first get the vector *ArmLine* between the shoulder and the goal position of the end effector. Get the vector *Arm* whose length is equal to the arm length and which is parallel to vector *ArmLine*. Then the difference vector $Diff = ArmLine - Arm$ becomes the motion vector of the base site, the shoulder. Suggest parallel primitive motions *move-to(right.palm-center, forward, F0, right.shoulder)* and *move-to(right.palm-center, downward, D0, right.shoulder)* and add them to the goal tree. Then, extract the downward and forward motion components from the difference vector *Diff*. Let the downward component be *DDist* and the forward component *FDist* (Figure 5.39). Add two goals *move-by(right.shoulder, downward, DDist)* and *move-by(right.shoulder, forward, FDist)* to the goal tree. They are parallel goals to be achieved at the same time. They are added to the goal tree to be achieved *before* the end effector motions previously added.

3. Get the top two goals from the goal tree. Nonlinear planning is used to plan for the goals, that is, each goal is planned for independently starting from the current situation. If the goals interfere, they are resolved by finding and removing the causes of the conflict. According to the motion dependencies, goal *move-by(right.shoulder, forward, FDist)* can be achieved by (i) *orient-by(torso-up-vector, forward, Ang2, pelvis-center)* and (ii) *move-by(body-ground-site, forward, F1, world)* where $FDist = F1 + F2(Ang2)$. Goal *move-by(right.shoulder, downward, DDist)* is achieved by (a) *orient-by(torso-up-vector, forward, Ang1, pelvis-center)* and (b) *move-by(pelvis-center, downward, D2, body-ground-site)*, where $DDist = D1(Ang1) + D2$. Here $D1(Ang1)$ is the movement of the shoulder caused by bending the torso-up-vector by angle $Ang1$ and $D2$ is the movement of the shoulder caused by lowering the pelvis-center.

4. Consider subplan (i) *orient-by(torso-up-vector, forward, Ang1, pelvis-center)* and (ii) *move-to(body-ground-site, forward, FDist, world)* for

the forward component goal of the shoulder. Based on the minimum disturbance constraint, motion (i) alone is first tried, because its base site *pelvis-center* is more proximal to the control point, the shoulder, than is the base site (*world*) of motion (ii). Motion (i) *orient-by(torso-up-vector, forward, Ang2, pelvis-center)* helps achieve the first goal as long as the precondition holds that the shoulder is above the horizontal plane passing through the pelvis-center (Figure 5.40). Incrementally simulate motion (i) to determine angle parameter *Ang2* (Figure 5.39). This angle value is difficult to predict without incremental simulation, because this value is affected by the body balance constraint. Simulate the motion until the forward component goal of the shoulder is achieved or the precondition is violated. Because the precondition is violated before the forward component goal is achieved, it is concluded that motion (i) alone is not sufficient. Motion (ii) is used to fill the remaining gap. The distance *F1* that motion (ii) should achieve is easily predicted by simple arithmetic (Figure 5.41). The temporal relation between motions (i) and (ii) is determined, because the motion whose base site is the world origin *world* is to be achieved first.

Consider subplan (a) *orient-by(torso-up-vector, forward, Ang1, pelvis-center)* and (b) *move-by(pelvis-center, downward, D2, body-ground-site)*, where $DDist = D1(Ang1) + D2$, for the downward component goal of the shoulder. According the minimum disturbance constraint, motion (a) alone is chosen first, because its base site *pelvis-center* is more proximal to the hand than is the base site of motion (b), *body-ground-site*. This is compatible with the observation that lowering the pelvis-center requires bending the knees and is considered more difficult than bending the torso-up-vector. (This may not be possible if bending the torso causes collision. But in this example, collisions are not considered.) Incrementally simulate motion (b) to determine angle parameter *Ang2*. Simulate until the downward component goal of the shoulder is achieved, that is, $DDist = D1(Ang1)$. Because motion (a) for the downward component is the same (bending the torso) as motion (ii) for the forward component, the simulation of motion (a) will be done by augmenting or subtracting the simulation of motion (ii).

5. The above reasoning suggests two motions *move-by(body-ground-site, forward, F1, world)* and *orient-by(torso-up-vector, forward, Ang2, pelvis-center)* for the forward component goal of the shoulder, and one motion *orient-by(torso-up-vector, forward, Ang1, pelvis-center)* for the downward component goal of the shoulder. If the two sets of motions do not interfere with each other with respect to the achievement of the two parallel goals, the motion postulation is successful. But this is not the case. When motion *orient-by(torso-up-vector, forward, Ang1, pelvis-center)* is simulated to achieve the downward motion component *DDist*, the shoulder goes below the horizontal plane passing through the pelvis-center (Figure 5.42). It therefore also contributes negatively to the forward mo-

tion component *FDist*, partly violating the goal *move-by(right.shoulder, forward, FDist)*. The **motion postulator** first attempts to resolve this goal interference by modifying distance parameters of motions in the current partial plan (goal tree). (If the rebinding of motion parameters fails, the **motion postulator** will try other combinations of contributing motions.) When the **motion postulator** modifies the subplans, subplans with more contributing motions are tried earlier than are those with fewer motions. In particular, subplans with single motions cannot be rebound without destroying the goals they achieved. Consider a subplan with more than one contributing motion. If the distance of one motion is modified to avoid the goal interference, this also causes the violation of the goal of the subplan. But the distances of the other motions can be modified to compensate for that violation. In the current example, the motion postulator keeps the subplan for the downward goal of the shoulder, because it has only one motion *orient-by(torso-up-vector, forward, Ang2, pelvis-center)*. This decision affects motion *orient-by(torso-up-vector, forward, Ang1, pelvis-center)* in the subplan for the forward goal of the shoulder, with $Ang1 := Ang2$. (This decision in fact turns out to be the same as the initial arbitrary decision used to simulate the two subplans.) Simulate the two subplans thus obtained. Get the distance of the negative contribution from the resulting simulated situation. Let the distance be *NegDist*. The distance can be achieved by adding an additional motion *move-by(body-ground-site, forward, NegDist, world)*. This motion compensates for the distance *DegDist* without doing any harm to the downward goal of the shoulder. But the **motion postulator** is supposed to see first that modifying the distance parameters of the current plans will work, before trying to add other motions. That is possible in this case by modifying motion *move-by(body-ground-site, forward, FDist, world)* to *move-by(body-ground-site, forward, FDist + NegDist, world)*.

6. After the shoulder goals are achieved, the current motions in the goal tree to be considered are the downward and forward motions of the palm-center with respect to (the current position of) the shoulder. Simulate these motions. Both motion components are achieved with equal importance given to each of them. They will be achieved if there are no hindrances (constraints or obstacles) at all, because the position of the shoulder has been obtained based on that assumption. If there are hindrances, the base of the palm-center (the shoulder) needs to be relocated as the palm-center is moved to intermediate goals. The intermediate goals are found to avoid the hindrances while achieving the goal of the palm-center.

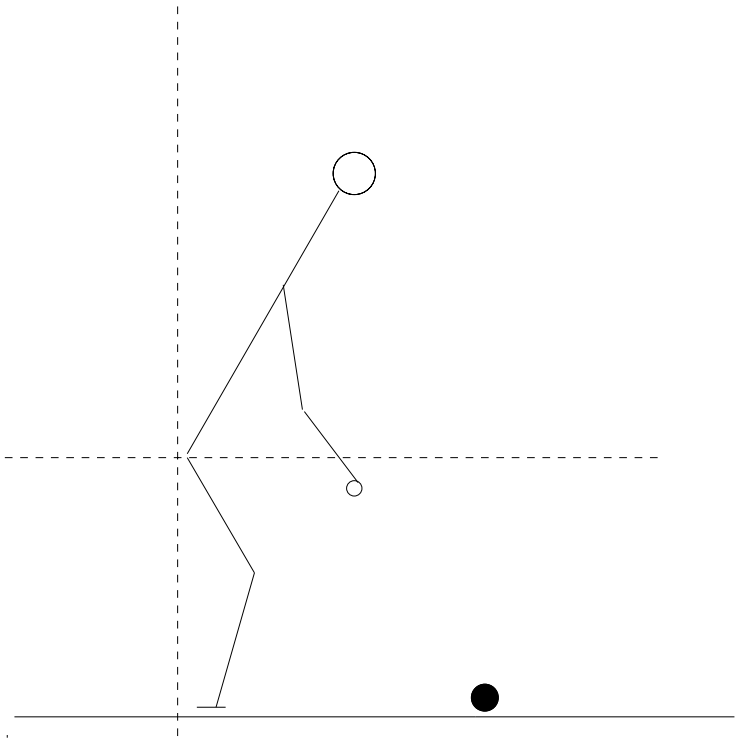

Figure 5.38: The Initial Situation for a Given Goal.

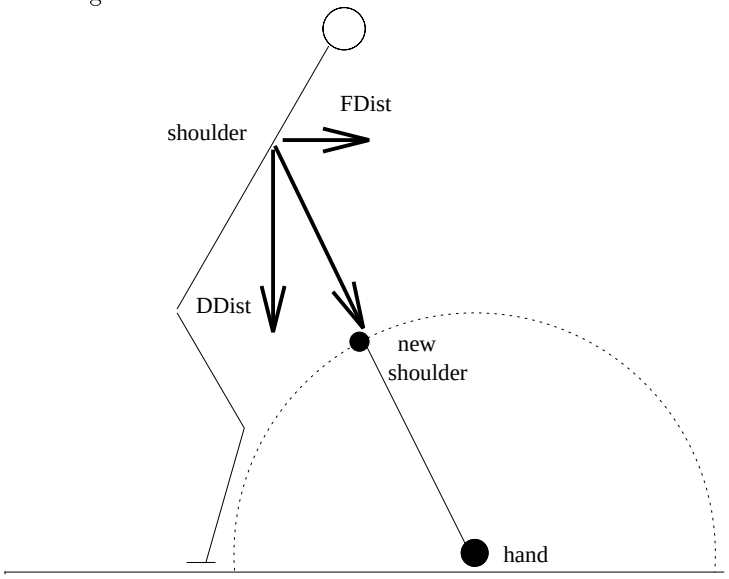

Figure 5.39: The Two Components of the Shoulder Difference Vector.

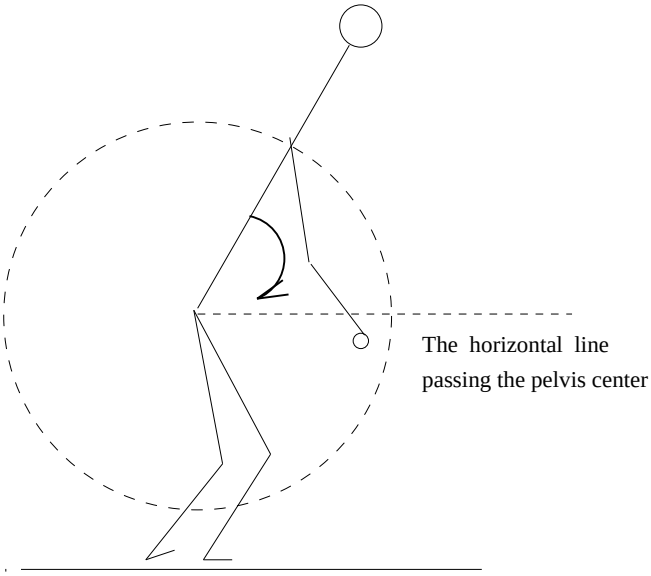

Figure 5.40: As Long as the Shoulder is Above the Horizontal Line, Bending the Torso-Up-Vector Moves the Shoulder Forward.

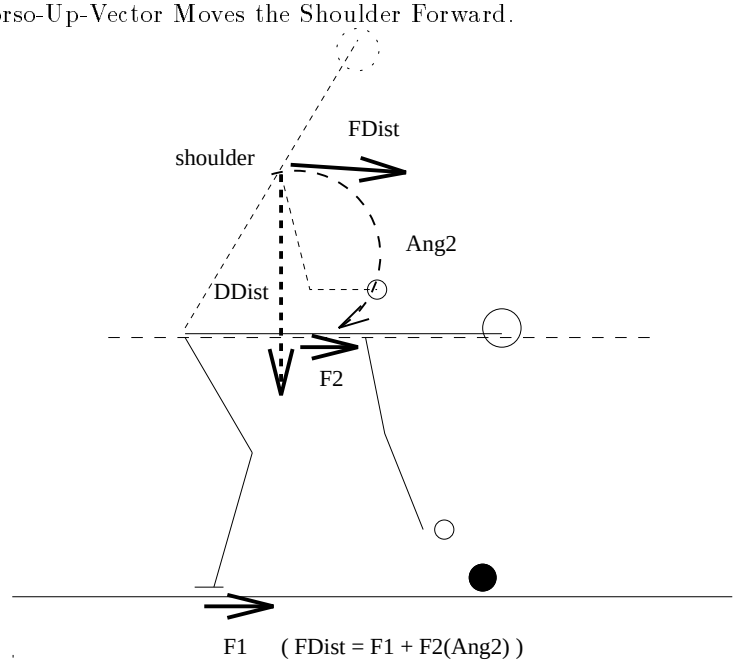

Figure 5.41: Achieving the Forward Component: By Moving the Body-Ground-Site and Bending the Torso.

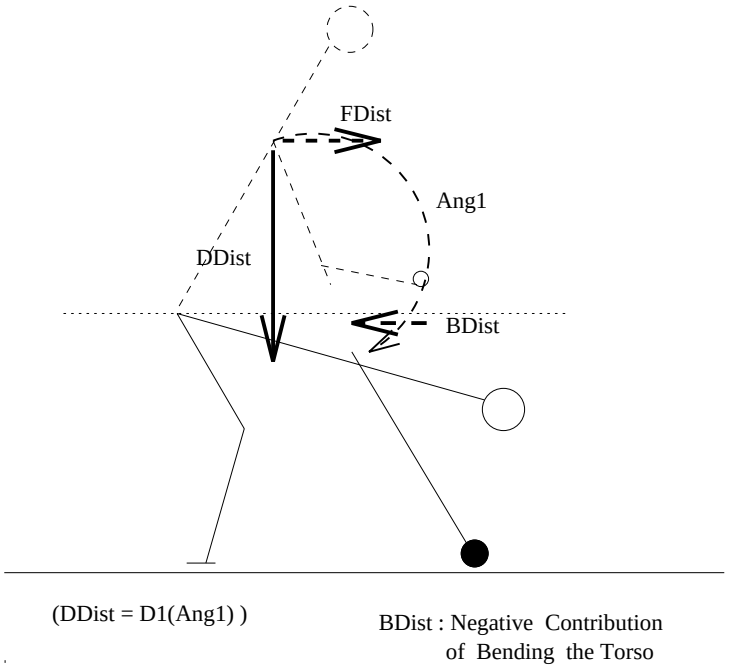

Figure 5.42: Achieving the Downward Component: By Rotating the Torso-Up-Vector Forward, Which Violates the Forward Component by Distance *BDist*.

Chapter 6

Task-Level Specifications

So far we have been talking about real-time interactive display and manipulation of human figures, with the goal of enabling human factors engineers to augment their analyses of designed environments by having human figures carry out tasks intended for those environments. This chapter explores the use of task-level specifications as an alternative to direct manipulation for generating task simulations.

By now, the reader should be convinced of the value of being able to simulate, observe and evaluate agents carrying out tasks. The question is what is added by being able to produce such simulations from high-level task specifications. The answer is efficient use of the designer's expertise and time. A designer views tasks primarily in terms of what needs to be accomplished, not in terms of moving objects or the agent's articulators in ways that will eventually produce an instance of that behavior – e.g., in terms of slowing down and making a left turn rather than in terms of attaching the agent's right hand to the clutch, moving the clutch forward, reattaching the agent's right hand to the steering wheel, then rotating the wheel to the left and then back an equal distance to the right. As was the case in moving programming from machine-code to high-level programming languages, it can be more efficient to leave it to some computer system to convert a designer's high-level goal-oriented view of a task into the agent behavior needed to accomplish it. Moreover, if that same computer system is flexible enough to produce agent behavior that is appropriate to the agent's size and strength and to the particulars of any given environment that the designer wants to test out, then the designer is freed from all other concerns than those of imagining and specifying the environments and agent characteristics that should be tested.

This chapter then will describe a progression of recent collaborative efforts between the University of Pennsylvania's Computer Graphics Research Lab and the LINC Lab (Language, INformation and Computation) to move towards true high-level task specifications embodying the communicative richness and efficiency of Natural Language instructions.

The first section will describe our earliest work on using simple Natural

Language commands to drive task animation. This work also dealt with an aspect of performance simulation that is rarely communicated in task specifications: how long an action will take an agent to perform. Section 6.2 describes our next effort at using Natural Language commands to drive task animation, focusing on how kinematic and spatial aspects of desired behavior are conveyed in Natural Language commands.

One of the consequences of these first two studies is understanding the value of a stratified approach to mapping from language to behavior: it is not efficient for, say, the language understanding components to make decisions that commit the agent moving a hand or a knee in a particular way, unless those movements are stated explicitly (but rarely) in the text. Because of this recognized need for intermediate representations between Natural Language descriptions and animation directives, an experiment was performed, described in Section 6.3, in which an intermediate level compositional language was created for specifying task-actions and generating task-level animated simulations from scripts written in this language. This demonstration paves the way for the ultimate connection between the behavioral simulation structure of the preceding Chapter and the conceptual structures of this one.

Each of these early efforts focused on individual commands in Natural Language. Task specifications, on the order of Natural Language instructions, go beyond individual commands in specifying what an agent should (and shouldn't) do. Since our current work is aimed at generating task animations from specifications as rich as Natural Language instructions, we devote the discussion in Section 6.4 to describing some features of instructions and what an understanding system requires in order to derive from them what a person would in understanding and acting in response to instructions.

6.1 Performing Simple Commands

[1]Our first work on driving task animation through Natural Language commands was a prototype system developed by Jeffrey Esakov that explored simple relations between language and behavior [EBJ89]. In this case, the desired behaviors were simple arm reaches and head orientation (view) changes on the part of the animated figures. While seemingly very easy, these tasks already demonstrate much of the essential complexity underlying language-based animation control.

6.1.1 Task Environment

In Esakov's work, the tasks to be animated center around a control panel (i.e. a finite region of more or less rigidly fixed manually-controllable objects) – here, the remote manipulator system control panel in the space shuttle with its variety of controls and indicators. Because Esakov was producing task animations for task performance analysis, he needed to allow performance

[1] Jeffrey Esakov.

to depend upon the anthropometry of the agent executing the task. In the experiments, all the controls were in fact reachable without torso motion by the agents being animated: failure situations were not investigated and the fully articulated torso was not yet available. An animation of one of the experiments can be found in [EB91].

6.1.2 Linking Language and Motion Generation

The primary focus of this work was to combine Natural Language task specification and animation in an application-independent manner. This approach used the following Natural Language script:

```
John is a 50 percentile male.
Jane is a 50 percentile female.
John, look at switch twf-1.
John, turn twf-1 to state 4.
Jane, look at twf-3.
Jane, look at tglJ-1.
Jane, turn tglJ-1 on.
John, look at tglJ-2.
Jane, look at twf-2.
Jane, turn twf-2 to state 1.
John, look at twf-2.
John, look at Jane.
Jane, look at John.
```

(The abbreviations denote *thumbwheels* such as `twf-1` and *toggle switches* such as `tglJ-1`. Thumbwheels have states set by rotation; toggles typically have two states, `on` or `off`.)

This type of script, containing simple independent commands, is common to checklist procedures such as those done in airplanes or space shuttles [Cen81]. The verb "look at" requires a view change on the part of the figure, and the verb "turn" requires a simple reach. (Fine hand motions, such as finger and wrist rotations, were not animated as part of this work.) The two primary problems then are specifying reach and view goals, and connecting object references to their geometric instances.

6.1.3 Specifying Goals

For these reach tasks the goal is the 3D point which the fingertips of the hand should touch. A view goal is a point in space toward which one axis of an object must be aimed. Spatially, such goals are just Peabody sites and must be specified numerically with respect to a coordinate system. Within the natural language environment, goals are not seen as coordinates, but rather as the objects located there – for example,

```
John, look at switch twF-1.
Jane, turn switch tglJ-1 on.
```

Because the exact locations of the switches is unimportant at the language level, in creating an animation, the switch name `tglJ-1` must be mapped to the appropriate switch on the panel in the animation environment. The same process must be followed for the target object toward which an object axis must be aligned in a view change. This problem reduces to one of object referencing.

6.1.4 The Knowledge Base

In general, all objects have names. Since the names in the task specification environment may be different from those in the animation environment, there must be a mapping between the names. The knowledge base that Esakov used contained information about object names and hierarchies, but not actual geometry or location. He used a frame-like knowledge base called DC-RL to store symbolic information [Ceb87]. For example, the DC-RL code for an isolated toggle switch, `tglJ-1`, follows:

```
{ concept tglJ-1 from control
  having (
    [role name with [value = "TOGGLE J-1"]]
    [role locative with [value = panel1]]
    [role type-of with [value = switch]]
    [role sub-type with [value = tgl]]
    [role direction with [value = (down up)]]
    [role states with [value = (off on)]]
    [role movement with [value =
          (discrete mm linear ((off on) 20 5))]]
    [role current with [value = off]])
}
```

To reference this switch from within the animation environment, a mapping file was generated at the same time the graphical object was described.

```
{ concept ctrlpanel from panelfig
  having (
    [role twF-1 with
          [ value = ctrlpanel.panel.twf_1 ]]
    [role twF-2 with
          [ value = ctrlpanel.panel.twf_2 ]]
    [role twF-3 with
          [ value = ctrlpanel.panel.twf_3 ]]
    [role tglJ-1 with
          [ value = ctrlpanel.panel.tglj_1 ]]
    [role tglJ-2 with
          [ value = ctrlpanel.panel.tglj_2 ]]
  )
}
```

The names `twF-1`, `twF-2`, `tglJ-1` correspond to the names of switches in the existing knowledge base panel description called `panelfig`. These names are mapped to the corresponding names in the animation environment (e.g., `ctrlpanel.panel.twf_1`, etc.) and are guaranteed to match.

6.1.5 The Geometric Database

The underlying geometric database is just Peabody code. The salient toggle and thumbwheel locations were simply mapped to appropriate sites on a host segment representing the control panel object. The relevant part of the Peabody description of the panel figure is shown:

```
figure ctrlpanel {
  segment panel {
    psurf = "panel.pss";
    site base->location =
         trans(0.00cm,0.00cm,0.00cm);
    site twf_1->location =
         trans(13.25cm,163.02cm,80.86cm);
    site twf_2->location =
         trans(64.78cm,115.87cm,95.00cm);
    site twf_3->location =
         trans(52.84cm,129.09cm,91.43cm);
    site tglj_1->location =
         trans(72.36cm,158.77cm,81.46cm);
    site tglj_2->location =
         trans(9.15cm,115.93cm,94.98cm);
  }
}
```

This entire file is automatically generated by a modified paint program. Using the panel as a texture map, switch locations are interactively selected and the corresponding texture map coordinates are computed as the site transformation. The panel itself is rendered as a texture map over a simple polygon and the individual sites then refer to the appropriate visual features of the switches.

6.1.6 Creating an Animation

Linking the task level description to the animation requires linking both object references and actions. A table maps the names of objects from the task description environment into the psurf geometry of the animation environment. In this simple problem domain the language processor provides the other link by associating a single key pose with a single animation command. Each part of speech fills in slots in an animation command template. Simple *Jack* behaviors compute the final posture required by each command which are then strung together via simple joint angle interpolation.

6.1.7 Default Timing Constructs

Even though the basic key poses can be generated based upon a Natural Language task description, creating the overall animation can still be difficult. We have already discussed posture planning and collision avoidance issues, but there is yet another problem that bears comment. From the given command input, the *timing* of the key poses is either unknown, unspecified, or arbitrary.

Action timings could be explicitly specified in the input, but (language-based) task descriptions do not normally indicate time. Alternatively, defining the time at which actions occur can be arbitrarily decided and iterated until a reasonable task animation can be produced. In fact, much animator effort is normally required to temporally position key postures. There are, however, more computational ways of formulating a reasonable guess for possible task duration.

Several factors effect task performance times, for example: level of expertise, desire to perform the task, degree of fatigue (mental and physical), distance to be moved, and target size. Realistically speaking, all of these need to be considered in the model, yet some are difficult to quantify. Obviously, the farther the distance to be moved, the longer a task should take. Furthermore, it is intuitively accepted that performing a task which requires precision work should take longer than one not involving precision work: for example, threading a needle versus putting papers on a desk.

Fitts [Fit54] and Fitts and Peterson [FP64] investigated performance time with respect to two of the above factors, distance to be moved and target size. It was found that amplitude (A, distance to be moved) and target width (W) are related to time in a simple equation:

$$\text{Movement Time} = a + b \log \frac{2A}{W} \tag{6.1}$$

where a and b are task-dependent constants. In this formulation, an index of movement difficulty is manipulated by the ratio of target width to amplitude and is given by:

$$\text{ID} = \log \frac{2A}{W} \tag{6.2}$$

This index of difficulty shows the speed and accuracy tradeoff in movement. Since A is constant for any particular task, to decrease the performance time the only other variable in the equation W must be increased. That is, the faster a task is to be performed, the larger the target area and hence the movements are less accurate.

This equation (known as Fitts' Law) can be embedded in the animation system, since for any given reach task, both A and W are known. The constants a and b are linked to the other factors such training, desire, fatigue, and body segments to be moved; they must be determined empirically. For button tapping tasks, Fitts [FP64] determined the movement time (MT) to be

$$MT_{\text{arm}} = 74ID - 70msec \tag{6.3}$$

Task Duration Times (msec)				
Actor	Action	ID	Fitts Duration	Scaled Duration
John	Look twf-1	2.96	321	963
John	Turn twf-1	5.47	335	1004
John	Look tglJ-2	4.19	566	1697
John	Look twf-2	4.01	530	1590
John	Look Jane	4.64	655	1966
Jane	Look twf-3	4.28	584	876
Jane	Look tglJ-1	3.64	456	685
Jane	Turn tglJ-1	5.39	329	493
Jane	Look twf-2	4.16	560	840
Jane	Turn twf-2	4.99	299	449
Jane	Look John	4.33	594	891

Table 6.1: Task Durations Using Fitts' Law.

In determining this equation, it was necessary to filter out the extraneous factors. This was done by having the subjects press the button as quickly as possible and allowing them to control the amount of time between trials. Jagacinski and Monk [JM85] performed a similar experiment to determine the movement time for the head and obtained the following equation

$$\mathrm{MT}_{\mathrm{head}} = 199ID' - 268msec \tag{6.4}$$

$$\mathrm{ID}' = \log \frac{2A}{W - W_0} \tag{6.5}$$

This equation is the result of equating the task to inserting a peg of diameter W_0 into a hole of diameter W, and resulted in a better fit of the data.

For our purposes the above constants may not apply. Since it was our desire to have the man in our animation move sluggishly and the woman move quickly (but not too quickly), we scaled Equations 6.3 and 6.4 by differing constants.

$$\begin{aligned} MT_{\mathrm{man(arm)}} &= 3 * MT_{\mathrm{arm}} \\ MT_{\mathrm{man(head)}} &= 3 * MT_{\mathrm{head}} \\ MT_{\mathrm{woman(arm)}} &= 1.5 * MT_{\mathrm{arm}} \\ MT_{\mathrm{woman(head)}} &= 1.5 * MT_{\mathrm{head}} \end{aligned}$$

This width of the target, W in equation 6.2 was chosen to be 1cm. For head movements, we chose $W_0 = .33°$ after [JM85]. This results in the action durations shown in Table 6.1.

Although Fitts' Law has been found to be true for a variety of movements including arm movements ($A = 5 - 30$cm), wrist movements ($A = 1.3$cm)

[Dru75, JM85, LCF76], and head movements ($A = 2.45 - 7.50°$) [JM85] the application to 3D computer animation is only approximate. The constants differ for each limb and are only valid within a certain movement amplitude *in 2D space*, therefore the extrapolation of the data outside that range and into 3D space has no validated experimental basis. Nonetheless, Fitts' Law provides a reasonable and easily computed basis for approximating movement durations.

While it may seem odd at first to have attacked both Natural Language interpretation and timing constructs as part of the same research, Esakov's work foreshadows our more recent work on language and animation by focusing on the fact that the same instruction, given to agents with different abilities, will be carried out in different ways. Language tells an agent what he or she should attempt to do: how he or she does it depends on them.

6.2 Language Terms for Motion and Space

[2]The next piece of work that was done on driving task animation through Natural Language commands was Jugal Kalita's work on how Natural Language conveys desired motion and spatial aspects of an agent's behavior. In English, the primary burden falls on verbs and their modifiers. Kalita's work showed how verbs and their modifiers can be seen as conveying spatial and kinematic constraints on behavior, thereby enabling a computer to create an animated simulation of a task specified in a Natural Language utterance. This work is described at greater length in [Kal90, KB90, KB91].

6.2.1 Simple Commands

To understand Kalita's contribution, consider the following commands:

- *Put the block on the table.*
- *Turn the switch to position 6.*
- *Roll the ball across the table.*

Each of these commands specifies a task requested of an agent. Performing the task, requires *inter alia* that the agent understand and integrate the meanings of the verbs (*put, turn, open, roll*) and the prepositions (*on, to, across*). This requires understanding the significant geometric, kinematic or dynamic features of the actions they denote.

In Kalita' approach to physically based semantics, a motion verb denotes what may be called a *core* or *kernel* action(s). This kernel representation is then used with object knowledge and general knowledge about actions to obtain semantic representations and subsequent task performance plans which are specific to a context – for example,

[2]Jugal Kalita.

- The core meaning of the verb *put* (as in *Put the block on the table*) establishes a geometric constraint that the first object (here, the block) remains geometrically constrained to (or, brought in contact with and supported by) a desired position on the second object (here, the table).
- The core meaning of the verb *push* (as in *Push the block against the cone*) involves applying a force on the manipulated object (here, the block) through the instrument object (here, the hand). The prepositional phrase specifies the destination of the intended motion.
- The verb *roll* (as in *Roll the ball across the table*) involves two related motions occurring simultaneously—one rotational about some axis of the object, and the other translational, caused by the first motion. The rotational motion is repeated an arbitrary number of times.

6.2.2 Representational Formalism

Geometric relations and geometric constraints

The meanings of locative prepositions are represented using a template called a *geometric-relation*. A simple geometric relation is a frame-slot structure:

geometric-relation: spatial-type:
source-constraint-space:
destination-constraint-space:
selectional-restrictions:

Spatial-type refers to the type of the geometric relation specified. Its values may be *positional* or *orientational*. The two slots called *source-constraint-space* and *destination-constraint-space* refer to one or more objects, or parts or features thereof, which need to be related. For example, the command *Put the cup on the table* requires one to bring the bottom surface of the cup into contact with the top surface of the table. The command *Put the ball on the table* requires bringing an arbitrary point on the surface of the ball in contact with the surface of the table top. Since the items being related may be arbitrary geometric entities (i.e., points, surfaces, volumes, etc.), we call them *spaces*. The first space is called the *source-constraint space* and the second, the *destination-constraint space*. The slot *selectional-restrictions* refers to conditions (static, dynamic, global or object-specific) that need to be satisfied before the constraint can be executed.

More complex geometric relations require two or more geometric relations to be satisfied simultaneously:

geometric-relation:
{ **g-union**
g-relation-1
g-relation-2
...
g-relation-n }

where *g-relation-i* is simple or complex.

Geometric relations are also used in the specification of *geometric constraints*, which are geometric goals to be satisfied:

Geometric-constraint: execution-type:
geometric-relation:

Geometric constraints are distinguished by their *execution-type* slot, which can take one of four values: *achieve, break, maintain* or *modify.*

Kinematics

The frame used for specifying kinematic aspects of motion is the following:

Kinematics: motion-type:
source:
destination:
path-geometry:
velocity:
axis:

Motions are mainly of two types: *translational* and *rotational.* In order to describe a translational motion, we need to specify the source of the motion, its destination, the trajectory of its path (path-geometry), and the velocity of the motion. In the case of rotational motion, path-geometry is circular, and velocity, if specified, is angular. Rotational motion requires an axis of rotation. If not specified, it is inferred by consulting geometric knowledge about the object concerned.

Kernel actions

The central part of an action consists of one or more components: *dynamics, kinematics* and *geometric-constraints*—along with control structures stating its other features. The control structures used in the examples that follow are: *repeat-arbitrary-times* and *concurrent.* The keyword *concurrent* is specified when two or more components, be they kinematic, dynamic or geometric constraints, need to be satisfied or achieved at the same time. The keyword *repeat-arbitrary-times* provides a means for specifying the frequentation property of certain verbs where certain sub-action(s) are repeated several times. The verbs' semantic representation need not specify how many times the action or sub-action may need to be repeated. However, since every action is presumed to end, the number of repetitions of an action will have to be computed from simulation (based on tests for some suitable termination conditions), or by inference unless specified linguistically as in *Shake the block about fifty times.*

6.2.3 Sample Verb and Preposition Specifications

Many of the features of Kalita's representation formalism can be seen in his representation of the verbs "roll" and "open" and the prepositions "in" and "across". Others can be seen in the worked example in Section 6.2.4.

A kinematic verb: *roll*

The verb *roll* refers to two motions occurring concurrently: a rotational motion about the longitudinal axis of the object and a translational motion of the object along an arbitrary path. The rotational motion is repeated an arbitrary number of times. The verb *roll* is thus specified as:

```
roll (l-agent, l-object, path-relation)←
        agent: l-agent
        object:l-object
        kernel-action:
          concurrent {{  {  kinematic:
                            motion-type:   rotational
                            axis:          longitudinal-axis-of (l-object)
                         } repeat-arbitrary-times }
                         { kinematic:
                            motion-type:   translational
                            path:          path-relation } }
        selectional restrictions:   has-circular-contour (l-object,
                                          longitudinal-axis-of (l-object))
```

A verb that removes constraints: *open*

One sense of *open* is *to move (as a door) from closed position*. The meaning is defined with respect to a specific position of a specific object. The closed position of the object can be viewed as a *constraint* on its position or orientation. Thus, this sense of *open* involves an underlying action that undoes an existing *constraint*. The object under consideration is required to have at least two parts: a solid 2D part (the *cover*) and an unfilled 2D part defined by some kind of frame (the *hole*). The meaning must capture two things: (1) that at the start of the action, the object's cover must occupy the total space available in object's hole in the constrained position, and (2) that the result of the action is to remove the constraint that object's cover and its hole are in one coincident plane. This is fulfilled by requiring that the two sub-objects (the hole and the cover) are of the same shape and size.

The definition for *open* is:

```
open (Ag, Obj) ←
  agent: Ag
  object: Obj
  kernel-action:
        geometric-constraint:
```

execution-type: break
spatial-type: positional
geometric-relation: source-constraint-space: Obj • hole
destination-constraint-space: Obj • cover
selectional-restrictions: contains-part (Obj, hole)
contains-part (Obj, cover)
area-of (Obj • cover) = area-of (Obj • hole)
shape-of (Obj • cover) = shape-of (Obj • hole)

A locative preposition: *in*

The sense of *in* captured here is *within the bounds of, contained in or included within*. According to Herskovits [Her86], this use type for *in* is *spatial entity in a container.* This meaning of *in* is specified as

in (X,Y) ⟵
geometric-relation:
spatial-type: positional
source-constraint-space: volume-of (X)
destination-constraint-space: interior-of (Y)
selectional-restrictions:
or (container-p (Y), container-p (any-of (sub-parts-of (Y))))
size-of (X) ≤ size-of (Y)
normally-oriented (Y)

A *container* is an object which can hold one or more objects such that the object is "surrounded by" the volume defined by the boundaries of the container. It is a concept which is difficult to define clearly, although heuristics can be devised to recognize whether an object is a container. For our purposes, if an object or any of its part(s) can work as container(s), it will be so labeled in the *function* slot of its representation. The second selectional restriction is due to Cooper [Coo68]. The third restriction is due to Herskovits, who explains its necessity by stating that the sentence *The bread is in the bowl* is pragmatically unacceptable if the bowl is upside down and covers the bread under it [Her86].

A path preposition: *across*

Path is a part of kinematic specification of a motion or an action. A complete definition of path requires specifying its source, destination and path geometry, which Kalita does, using a structure called a *path-specification*:

path-specification:
source:
destination:
path-geometry:

Across is one of several path prepositions in English. Others include *from, to, around, round* and *along*. *Across* has two types of meanings—dynamic and static (locative) meaning. The dynamic meaning implies a journey across an object, whereas the static meaning implies a location between two lines (edges) perpendicular to them and touching, and (possibly) extending beyond them. The dynamic sense of *across* is seen in:

- Roll/Slide/Move the block/ball across the board.

This dynamic sense of *across* specifies all three components required for path specification.

across (X, Y) ⟵ path-specification:
 source: any-of (exterior-edges-of (Y, parallel-to (longitudinal-axis (Y))))
 destination:any-of (exterior-edges-of (Y, parallel-to (longitudinal-axis (Y))))
 path-geometry: straight-line
 selectional-restrictions:
  destination ≠ source
  has-axis (X, longitudinal)
  angle-between (path-geometry, longitudinal-axis (Y), 90°)
  length (Y) ≥ width (Y)
  length (Y) > dimension-of (X,along-direction (longitudinal-axis (Y)))

The *longitudinal axis* of an object is the axis along which the *length* of an object is measured. There are a number of selectional restrictions imposed on the objects X and Y also. For example, the reason for the fourth selectional restriction can be gauged from the two phrases: *across the road* and *along the road*.

6.2.4 Processing a sentence

The sentence *Put the block on the table* can be used to show how Kalita's system obtains a meaning for a whole sentence from the meanings of its parts, i.e., the lexical entries of its constituent words.

The lexical entry for *put* specifies the achievement of a geometric relationship between an object and a location specified by a prepositional phrase. The meaning of the verb is specified in terms of a yet-unspecified geometric relation between two objects. The preposition *on* along with the objects involved leads to the sense that deals with support.

A bottom-up parser [FW83] returns the logical meaning representation as (*put you block-1* (*on block-1 table-1*)). In this representation, the verb *put* takes three arguments: a subject, an object and the representation for a locative expression. Entities *block-1* and *table-1* are objects in the world determined to be the referents of the noun phrases. The logical representation has *you* as the value of the subject since the sentence is imperative.

Now, to obtain the intermediate meaning representation, the arguments of *put* in the logical representation are matched with the arguments in the following lexical entry for *put*:

```
put (l-agent, l-object, l-locative) ⟵
    agent: l-agent
    object: l-object
    kernel-actions:
        geometric-constraint:
            execution-type:     achieve
            geometric-relation: l-locative
```

This lexical entry has three arguments. After matching, *l-agent* has the value *you*, *l-object* has the value *block-1*, and *l-locative* has the value (*on block-1 table-1*). The value of the *geometric-relation* slot (of the *kernel-actions* slot in the representation) is filled in by the semantic representation for the *l-locative* argument which is created from the meaning of "on the table", using the following definition of "on":

```
on (X,Y) ⟵
    geometric-relation:
        spatial-type:                positional
        source-constraint-space:     any-of (self-supporting-spaces-of (X))
        destination-constraint-space: any-of (supporter-surfaces-of (Y))
    selectional-restrictions:
        horizontal-p (destination-constraint-space)
        equal (direction-of (normal-to
            destination-constraint-space), "global-up")
        free-p (destination-constraint-space)
```

As a result, the intermediate meaning representation of "put the block on the table" is:

```
agent:    you
object:   block-1
kernel-actions:
    geometric-constraint:
        execution-type:     achieve
        geometric-relation:
            spatial-type:                positional
            source-constraint-space:     any-of
              (self-supporting-spaces-of (block-1))
            destination-constraint-space: any-of
              (supporting-surfaces-of (table-1))
        selectional-restrictions:
            horizontal-p (destination-constraint-space)
            equal (direction-of (normal-to
              destination-constraint-space), "global-up")
            free-p (destination-constraint-space)
```

In order to execute the action dictated by this sentence, the program looks at the knowledge stored about the block to find a part of the block on which

it can support itself. It observes that it can be supported on any one of its faces and no face is more salient than any other. A cube (the shape of the block) has six faces and one is chosen randomly as the support area. Next, the program consults the knowledge stored about the table and searches for a part or feature of the desk which can be used to support other objects. It gathers that its function is to support "small" objects on its top. This top surface is also horizontal. As a result, finally, the system concludes that one of the sides of the cube has to be brought in contact with the top of the table.

The final meaning for the sentence obtained is

```
agent:  you
object: block-1
kernel-actions:
    geometric-constraint:
        execution-type:      achieve
        geometric-relation:
            spatial-type:                  positional
            source-constraint-space:       block-1•side-2
            destination-constraint-space:  table-1•top-1
```

block-1•side-2 represents a specific face of a specific block. *table-1•top-1* represents the top surface of a specific table. This final representation is then sent to a planner [JKBC91] which produces a plan for performing the task by an animated agent in a given workspace. The plan is taken up by a simulator [BWKE91] which establishes connection with *Jack* and then produces an animation:

> The block is initially sitting on top of a closed box. The agent reaches for it with his right hand, grasps it, moves it to a point near the top of a table to his left, places it on the table, and moves his hand back.

As with Esakov's work, there were still unfortunate capability gaps in the simulator available to Kalita. In particular, the lack of a flexible torso, unchecked collisions with the environment, and no balance constraints led to some painful-looking postures and object trajectories which passed through obstacles.

6.2.5 Summary

This section has discussed the representation of meanings of some verbs and prepositions, emphasizing the importance of geometric information such as axes of objects, location of objects, distance or angle between objects, path of object motion, physical contact between objects, etc., in the meaning representation of prepositions. Elsewhere it is shown that such geometric considerations are important for not only representing verbs and prepositions, but also adverbs [KB90].

In the work described here, the operational meanings of action verbs and their modifiers have been represented in terms of components pertaining to constraints and kinematic/dynamic characterization. For additional examples of decomposition see [Kal90].

6.3 Task-Level Simulation

[3]The third experiment tested the feasibility of using what might be viewed as low-level task primitives to create task animations [Lev91]. If successful, this would have two advantages:

- Since we viewed some kind of low-level task primitives as being the output specification language of any language processing stages, it would allow us to design and test a set of primitives in parallel with the other system components.
- This kind of lower-level specification language might itself be usable by an engineer to generate animations in terms of task-level actions rather than having to specify particular body movements.

To illustrate the latter contrast, consider a scene with an animated agent, a table, and a cup on a shelf next to the table. The animator–engineer wants to create an animation of the agent moving the cup from the shelf to the table. A task-level specification could enable the animator–engineer to produce the desired behavior, using a set of **task-action** specifications. For example, the sequence

grasp-action (*hand cup*)
position-action (*cup table-top*)

could be used to generate an animation of the agent's hand grasping the cup, followed by a positioning of the cup on the top of the table.

As a test environment we used an expanded version of some written instructions to remove a Fuel Control Valve (FCV) from an imaginary aircraft fuselage (Figure 6.1).

Fuel Control Valve Removal Instructions:

1. With right hand, remove socket wrench from tool belt, move to front of body. With left hand, reach to tool belt pocket, remove 5/8 inch socket, move to wrench, engage. Adjust ratchet for removal.
2. Move wrench to left hand bottom hole, apply pressure to turn in a loosening motion, repeat approximately 7 times to loosen threaded bolt.
3. Move wrench away from bolt, with left hand reach to bolt and remove bolt and washer from assembly, move left hand to belt pouch, place bolt and washer in pouch.

[3]Libby Levison.

Figure 6.1: A Frame from the Fuel Control Valve Removal Task. The FCV is the Cylindrical Object Mounted to the Flat Plate.

4. Move wrench to bottom right hand bolt, apply pressure to turn in a loosening motion, repeat approximately 7 times to loosen threaded bolt.

5. Repeat operation 3.

6. Move wrench to top bolt, apply pressure to turn in a loosening motion, repeat approximately 6 times to loosen threaded bolt. Move left hand to grasp assembly, loosen the bolt the final turn. Move wrench to tool belt, release. With right hand reach to bolt, remove bolt and washer, place in pouch. Return right hand to assembly, with both hands move Fuel Control Valve to movable cart and release.

6.3.1 Programming Environment

The work area, tools and parts for the scene were modeled with *Jack*. Just as the engineer who currently writes the instruction manuals has knowledge of the task and knows, for example, that a Phillips head screwdriver is required, it is assumed that the engineer-animator will have the knowledge required to lay out the scene of the animation. It is also assumed that a skilled engineer is already trained in analyzing tasks and developing instruction sets for the do-

main. This project simply provides a different medium in which the engineer can explain the task.

The task simulation is based on **Yaps**, a symbolic process simulator [EB90, BWKE91]. **Yaps** provides *animation-directives* which access *Jack*'s behaviors. These animation-directives are not only ordered and sequenced via **Yaps'** temporal and conditional relationships [KKB88], but can also be composed to produce parameterized simulation procedures. These procedures, called **task-actions**, are defined for a number of parameters (agent, object, location, etc.). The same task-action can thus be used at various times with different parameters to create distinct animation segments. The possibility of defining and reusing these procedures simplifies the animation programming problem for the engineer. By extending these procedural compositions, high-level procedures could be generated so that the mapping from the instructions to these procedures would be straightforward.

KB [Esa90] is a frame-based, object-oriented knowledge system which establishes symbolic references to *Jack*'s geometric data. While *Jack* maintains and manipulates the geometric model of the world, **KB** maintains the symbolic information. **Yaps** uses **KB**'s symbolic representation to manipulate the geometric model. (These symbolic **KB** representations are passed to the **Yaps** task-actions as parameters.) This frees **Yaps** from "knowing" the specific world coordinates of an object or the object's exact geometric representation. For instance, if *Jack* contains a model of a cup, **KB** would have an entry which identified *cup* as that particular *Jack* entity. **Yaps** has no knowledge of the object's location; **KB**'s mapping from symbolic to geometric representation will resolve any ambiguity. Thus the animator need not talk about *the-cup-on-the-table-at-world-coordinates-(x,y,z)*, but can reference the symbolic entity, *cup*. Should the *cup* move during the course of the action, **KB** resolves the problem of the *cup*'s exact location.

6.3.2 Task-actions

At the time of this research, **Yaps** provided only three low-level animation-directives with which to access *Jack* behaviors. These are *generate-motion*, *create-constraint* and *delete-constraint*. *Generate-motion* causes an object (not necessarily animate) to move from its current location to another. (No path planning was performed in the *Jack* version of the time, and **Yaps** handled frame-to-frame timing directly as described in Section 6.1.) *Create-constraint* establishes a physical link between two (not necessarily adjacent) objects. If two objects are linked together and one of the objects is moved, the second object moves along with it. The physical constraint (relation) between the objects is maintained. *Create-constraint* can be further specified to use *positional* and/or *orientational* alignments. *Delete-constraint* removes the specified constraint between two objects.

Yaps provides a mechanism for building animation templates by combining or composing the above animation-directives. Using different combinations of *generate-motion*, *create-constraint*, and *delete-constraint*, and vary-

ing the agents and the objects of these *animation-directives* as well as their temporal and causal relations, it is possible to build a set of task-actions. Task-actions can themselves be composed into more complex task-actions. As the procedures acquire more specification, the task-actions approach task-level descriptions. It is important to note, however, that task-actions simply define templates; an animation is realized by instantiating the task-actions, supplying parameters as well as timing constraints and other conditions. The composability of the task-actions allows for the definition of some abstract and high-level concepts. It is these high-level animation descriptions which will allow the engineer to program an animation at the task-level.

6.3.3 Motivating Some Task-Actions

The first templates to be defined were simply encapsulations of the *Jack* animation-directives: **reach-action**(*agent object*), **hold-action**(*agent object*) and **free-object-action** (*object*) – were just *generate-motion*, *create-constraint* and *delete-constraint*, respectively. (Although the names chosen for the task-actions do make some attempt to elicit their definition, there was no attempt to come up with definitive definitions of these actions in this segment of the research project.) In the following, the use of *agent* and *object* is simply for readability; for example, a **hold-action** can be applied between two objects (*e.g.*, **hold-action** (*wrench-head 5-8th-socket*)).

Consider trying to describe the actions inherent in the example:

Move the cup to the table

assuming that the agent is not currently holding the cup. The agent must first hold the cup before he can move it. How is this animation specified? Explicitly stating the sequence of actions:

reach-action (*agent cup*)
hold-action (*agent cup*)

to cause the agent to reach his hand to the location of the cup and to constrain his hand to the cup seems awkward. Composing two task-actions allows a new task-action **grasp-action** to be defined:

(deftemplate **grasp-action** (*agent object*)
reach-action (*agent object*)
hold-action (*agent object*)).

(This is the actual **Yaps** definition. `Deftemplate` is the **Yaps** command to define a new task-action template.) **Grasp-action** is a sequence of instantiations of two primitive task-actions.

Now that the agent can grasp the cup, how can he move the cup? A second action, **position-action**, is defined to relocate the cup to a new location and constrain it there:

(deftemplate **position-action** (*object1 location*)
reach-action (*object1 location*)
hold-action (*object1 location*)).

If a previous action had left an object (the cup) in the agent's hand, this task-action could be used to move the object to a new location (**position-action** *cup table*). (In this instruction set, the only use of the instruction "move something that is already being held" required that the object be constrained to the new location. This is the justification of the **hold-action** in this definition.) Note here that *location* could be the location of *object2*.

Thus, to animate the instruction:

Move the cup to the table

the *animation-script* could be:

grasp-action (*agent-right-hand cup*)
position-action (*cup table-top*).

It is still necessary to specify a list of commands, since no high-level task-action has been defined for *move*, and therefore the action must be described in increments. However **move-action** could be defined as:

(deftemplate **move-action** (*agent object1 location*)
grasp-action (*agent object1*)
position-action (*object1 location*)).

In other words, grasp (reach to and hold) *object1*, and position (move to and constrain) *object1* at *location* (where *location* might be the location of some *object2*). In the *Move the cup* example, the instantiation required to achieve the desired animation would be:

move-action (*agent-right-hand cup table-top*).

This conciseness is one benefit of task-action composition.

Once the *cup* is actually on the *table*, it can be "un-grasped" by using:

free-object-action (*cup*)

which breaks the constraint between the *hand* and the *cup*. If the *hand* is later moved, the *cup* will no longer move with it.

The final *animation script* for *Move the cup to the table* becomes:

move-action (*agent-right-hand cup table-top*)
free-object-action (*cup*).

6.3.4 Domain-specific task-actions

The *Move the cup to the table* example motivated a few fundamental task-action definitions. Some of these are actions common to many instructional

tasks and milieus; this set of task-actions is also usable in the instruction set describing the FCV removal. However, it was also necessary to return to the instruction set and develop **Yaps** definitions for actions specific to the domain in question. These task-actions can be either primitive (see **turn-action** below) or compositional (see **ratchet-action**). The first new task-action, **attach-action**, is defined as:

(deftemplate **attach-action** (*agent object1 object2*)
move-action (*agent object1 object2*)
hold-action (*object1 object2*)).

This allows the agent to grasp *object1*, move it to the location of *object2*, and establish a constraint between *object1* and *object2*. The expansion of this task-action is the command string:

reach-action, hold-action, reach-action, hold-action.

Attach-action could have been equivalently defined as:

(deftemplate **attach-action** (*agent object1 object2*)
grasp-action (*agent object1*)
position-action (*object1 object2*))

which would expand to exactly the same *Jack* animation-directive command string as above. The task-action definitions are associative; this provides flexibility and power to the system, and increases the feasibility of defining a minimal set of task-actions to be used throughout the domain.

The FCV removal instructions also require: **turn-action** (*object degrees*). **Turn-action** causes the *object* to rotate by the specified number of *degrees*. The geometric definition of the object includes information on its DOFs; for example, around which axis a bolt will be allowed to rotate. At the time that this research was done, the system did not have a feedback tool to monitor *Jack* entities; instead of testing for an ending condition on an action (a bolt being free of its hole), actions had to be specified iteratively (the number of times to turn a bolt). **Turn-action** is actually a support routine, used in the final task-action needed to animate the FCV instructions: **ratchet-action**. This is defined as:

(deftemplate **ratchet-action** (*object degrees iterations*)
turn-action (*object degrees*)
turn-action (*object* $-$*degrees*)
ratchet-action (*object degrees iterations*-1)).

Ratchet-action is used to animate of a socket wrench ratcheting back and forth.[4]

[4]Having to explicitly state a number of degrees is not an elegant programming solution; it would have been preferable to take advantage of *Jack*'s collision detection algorithms to determine the range of the ratchet movement. Processing considerations at the time the work was done required this rather rough implementation.

The complete set of task-actions is listed below. With this set of only nine task-actions, it was possible to program the entire animation script from the natural language instructions (see Table 6.2 for an excerpt of the final animation script).

- **reach-action** (*agent object*)
- **hold-action** (*agent object*)
- **free-object-action** (*object*)
- **grasp-action** (*agent object*)
- **move-action** (*agent object location*)
- **attach-action** (*agent object1 object2*)
- **position-action** (*object1 object2*)
- **turn-action** (*object degrees*)
- **ratchet-action** (*object degrees iterations*)

6.3.5 Issues

Where Does Task-Action Decomposition Stop?

There is an interesting question as to whether, in defining task-actions, one needs to be concerned with variations that arise from differences in agents and their abilities.

Because our work is embedded in *Jack*, variations in agent ability at the animation specification level is not a concern. As long as the animation is within the agent's capabilities (and thus the animation is "solvable"), substituting different agents gives different valuations of the tasks. By testing different agents with varying abilities, one can analyze the task requirements and gather information on human factors issues. Similarly, it is possible to vary workplace geometry, tools, and agent placement.

Note the comparison here between innate and planned action. In reaching to grab a cup, we do not think about how to control the muscles in the forearm; we do, however, consider the goal of getting our hand to the same location as the cup. This distinction between cognizant motion and action is internal in this animation; *Jack* manages the motor skills. The same distinction is found in the level of detail of the instructions. One does not tell someone:

> *Extend your hand to the cup by rotating your shoulder joint* 40° *while straightening your elbow joint* 82° *degrees. Constrain your hand to the cup by contracting fingers . . . .*

Rather, we give them the goal to achieve and allow that goal to lend information as to how to accomplish the instruction. The hierarchy of the task-actions captures some of this knowledge.

The task-actions have been defined in such a way that they are not concerned with the abilities of a specific agent, but rather allow for interpretation

Table 6.2: Animation Script Excerpt.

```
;;;  No. 1
;;;   With right hand, remove socket wrench from tool belt,
;;;   move to front of body.  With left hand, reach to tool belt
;;;   pocket, remove 5/8" socket, move to wrench, engage.
;;;   Adjust ratchet for removal.
;
; with the right hand, grasp the wrench from the tool belt,
; and move it to site-front-body
;
(instantiate move-action
      (fred-rh wrench-handle fred-front-body-site planar)

   :instancename "r0-wrench-to-front"

   :time-constraints '((start now)
                       (duration
                          (eval(+(fitts fred-rh wrench-handle)
                                 (fitts wrench-handle
                                 fred-front-body-site)))))

; with the left hand, attach socket to wrench handle.
;  an attach entails, reaching for the socket, grasping
;  it and moving it to the wrench head.
;  if successful, free the left hand from the socket.
;
(instantiate attach-action
      (fred-lh 5-8th-socket wrench-head
                attach-socket-time planar oriented)

   :instancename "r5-attach-socket"

   :time-constraints '((start (end "r0-wrench-to-front"))
                       (duration (eval
                           (+ (fitts fred-lh 5-8th-socket)
                              (fitts fred-left-pocket
                                     fred-front-body-site)
                              attach-socket-time))))

   :on-success '(progn
                   (free-object-action fred-lh)
                   (free-object-action 5-8th-socket)
                   (hold-action wrench-head 5-8th-socket
                         :orientation-type '("orientation")))
```

based on each agent's capabilities. Not only does this allow the same animation script to be used for different agents, generating different analyses, but it also means that the definitions of the task-actions decomposition stops at the level of innate action. There is no need to have multiple task-action definitions for various physical attributes; *Jack* handles this issue for us.

Instruction Translation

We noted earlier that one advantage of a **task-action** level of specification was that it might allow an engineer/animator to animate tasks directly. In terms of the above task-actions, moving the cup to the table (noted in the introduction) could be animated by issuing either of two command sequences:

grasp-action (*agent-right-hand cup*)
position-action (*cup table-top*)
free-object-action (*cup*)

or:

move-action (*agent-right-hand cup table-top*)
free-object-action (*cup*).

In both cases, the engineer has decided to release the constraint between the *agent* and the *cup* as soon as the *cup* is on the *table-top*. The engineer has also described the required animation at the task-level.

Sequencing Sub-tasks

Yaps is a simultaneous language; that is, all task-action instantiations are resolved concurrently. To sequence the actions and force them to occur in a specific order, the engineer/animator must use the *timing-constraints* option provided by **Yaps**. This construct allows the user to specify starting, ending and duration conditions for the instantiation of each action. It is possible to achieve the ordering needed to create a sequential animation by predicating the starting condition of instruction-2 on the ending condition of instruction-1; but a task-action template, which is defined as a series of other task-actions, has the sequencing automatically built in via the instantiation process. If this were not the case, defining **grasp-action**, for example, would be impossible because achieving and completing the **reach-action** before starting the **hold-action** could not be guaranteed.

The actions do not need to be performed discretely. Other **Yaps** timing constructs allow the actions to be overlapped and delayed by specifying *(start (after 5 min))* or *(start now)*, for example [KKB88]. Nor is defining a discrete linear order on the sub-tasks the only possibility. The simultaneous nature of **Yaps** is used to animate actions (such as moving an object with both hands) by simultaneously animating:

move-action (*agent-left-hand box*)
move-action (*agent-right-hand box*).

The **Yaps** timing constraints provide a powerful mechanism for specifying the relationships among the task-actions in the animation. Timing is one of the most critical issues involved in generating realistic animations; the power that **Yaps** provides in resolving timing issues greatly enhances the potential of the *Jack* animation system.

Task Duration

The **Yaps** timing constraints provide a powerful mechanism for specifying the inter-relationships among the task-actions in the animation script. Timing is one of the most critical issues involved in generating realistic animations. We have already noted that it is not sufficient to simply list all the actions; they must be times, sequenced and connected temporally. As in Esakov's work, adaptations of Fitts' Law were used to determine mimimum action times. Fitts' Law was used to calculate the duration of all **reach-action** instantiations. Thus, time requirements were cumulative (i.e., the sum of the sub-task-action times). **Create-constraint** uses a small default constant time to estimate sub-task duration. Although Fitts' Law only approximates the action times in this domain and must be further scaled by a motivation factor, it does give reasonable estimates. Relative to one another, the sub-task times make sense. Although the length of each **task-action** might not be correct, the animation does appear to be temporally coherent.

6.3.6 Summary

Recent work in defining *animation behaviors* reviewed earlier in this book greatly expands the set of animation directives available in *Jack*. In our current work, we will investigate using the new animation behaviors to script animations. Since animation directives form the semantical basis for our action definitions, a more powerful set of animation directives provides us with a richer language with which to work. As it becomes easier to define new task-actions, the animator will spend less time coordinating sub-actions.

Finally, this new vocabulary will allow us to express tasks (or define task-actions) which differ from the earlier work in their semantic content. Our first attempt at rescripting the instruction set resulted in a more realistic animation, in that the new behaviors allowed us to include such low-level actions as `take step to maintain balance` when the animated agent was reaching beyond his comfort range. We need to compare the expressive powers of the previous animation directives with the enhanced set of animation behaviors.

6.4 A Model for Instruction Understanding

[5]The three experiments described in the previous sections were all concerned with the operational semantics of *single-clause commands*. But the range of

[5]Barbara Di Eugenio, Michael White, Breck Baldwin, Chris Geib, Libby Levison, Michael Moore.

tasks that can be communicated to an agent with such commands is very limited – the less expertise and experience on an agent's part, the more he needs to be told. A telling example of this is given in [Pri81]. Here, Prince compares a recipe for stuffed roast pig given in a nineteenth century French cookbook with that given in Rombauer's contemporary *The Joy of Cooking*. The former says, essentially, "Roast pig. Stuff with farce anglaise." Rombauer's instructions go on for two pages: she assumes very little culinary experience with pigs on the part of today's men and women.

Multi-clause commands are very common in maintenance and assembly instructions, such as the following examples from Air Force maintenance manual T.O. 1F-16C-2-94JG-50-2:

> "With door opened, adjust switch until roller contacts cam and continuity is indicated at pins A and B. Verify positive switch contact by tightening bottom nut one additional turn." (p. 5-24)
>
> "Hold drum timing pin depressed and position entrance unit on drum. Install three washers and three bolts, release drum timing pin, and torque bolts to 60-80 inch-pounds." (p. 6-14)

Now just as multi-clause texts are commonly organized into paragraphs, multi-clause instructions are commonly organized into *steps*. In fact, the above multi-clause commands are actually single steps from longer, multi-step instructions. While there are no firm guidelines as to what a single instruction step should encompass, there is a strong tendency at least for steps to be organized around small coherent sub-tasks (such as adjusting a switch or installing a component, as in the above examples). A typical step may specify several actions that need to be performed together to accomplish a single subtask, or several aspects of a single complex action (e.g. its purpose, manner, things to watch out for, appropriate termination conditions, etc.). The agent must develop some degree of understanding of the whole step before starting to act.

In our current work on instruction understanding, we add to this sub-task sense of step, the sense that a step specifies behavior that the agent must attend to continuously: while carrying out a step, the agent's attention is fixed on the task at hand. Communication with the instructor is not allowed until completion (or failure) of the current step. Because of this, a step defines the extent of the instructions that must be processed *before* the agent begins to act on them. (With some reflection on one's own confrontations with new instructions, it is easy to recall situations where one has tried to understand too much or to act on too little understanding. It is not always obvious when one should begin to act.)

While our focus is on multi-clause instructions, it turns out that many of their important features can be demonstrated simply with two-clause instructions. (As in many things, the biggest leap is from one to two.) The two-clause example we will use here to describe our framework for instruction understanding and animation is:

"Go into the kitchen to get me the coffee urn."

This example will be used to illustrate, among other things:

- expectations raised by instructions;
- the need for *incremental* generation of sub-goals (plan expansion) in order to act in accordance with instructions;
- the need to accommodate the agent's behavior in carrying out actions, to the objects being acted upon; and
- the need to develop plans at more than one level.

Figure 6.2 shows a schematic diagram of the AnimNL (ANIMation from Natural Language) architecture. Before going through the example, we want to call attention to the system's overall structure – in particular, to the fact that it consists of two relatively independent sets of processes: one set of which produces commitments to *act* for a particular purpose, what we call *animated task actions* – e.g.

- goto(door1, open(door1)) – "go to door1 for the purpose of opening it"
- grasp(urn1, carry(urn1)) – "grasp urn1 for the purpose of carrying it"

and the other set of which figures out how the agent should *move* in order to act for that purpose. In this framework, instructions lead to initial commitments to act, and actions once embarked upon allow further commitments to be made and acted upon. (While our discussion here will be in terms of single-agent procedures, it can be extended to multi-agent procedures by adding communicative and coordinating actions. As shown in earlier chapters, both *Jack* and its behavioral simulator can support the activity of multiple agents. However, extending the upper set of processes to delineate the communication and coordination required of multiple agents cooperating on a task requires solution of many problems currently under investigation by members of the AI planning community.

We now begin by giving AnimNL the instruction step:

"Go into the kitchen to get me the coffee urn."

A picture of the agent in its starting situation, when it is given the instruction, is shown in Plate 6.

Steps are first processed by a parser that uses a combinatory categorial grammar (CCG) [Ste90] to produce an action representation based on Jackendoff's *Conceptual Structures* [Jac90]. We are using CCG because of its facility with conjoined constituents, which are common in instructions – for example

"Clear and rope off an area around the aircraft and post warning signs." [Air Force Maintenance manual T.O. 1F-16C-2-94JG-50-2]

Figure 6.2: AnimNL System Architecture.

We are using Jackendoff's Conceptual Structures for two reasons: first, the primitives of his decompositional theory capture important generalizations about action descriptions and their relationships to one another, and second, they reveal where information may be missing from an utterance and have to be provided by inference. For the instruction step "Go into the kitchen to get me the coffee urn", the parser produces the following structure:

$$\left[\begin{array}{l} \mathrm{GO}_{\mathrm{Sp}}([\textsc{agent}]_i, [\mathrm{TO}([\mathrm{IN}([\textsc{kitchen}])])]) \\ \mathrm{FOR}(\beta) \end{array}\right]_\alpha$$

$$[\mathrm{CAUSE}(i, [\mathrm{GO}_{\mathrm{Sp}}([\textsc{coffee-urn}]_j, k)])]_\beta$$

$$\left[\begin{array}{l} \mathrm{FROM}([\mathrm{AT}(j)]) \\ \mathrm{TO}(l) \end{array}\right]_k$$

This representation makes explicit the fact that getting the coffee urn involves its moving from its current location to a new one (which should be the location of the instructor). The FOR-function (derived from the *to*-phrase) encodes the purpose relation holding between the *go*-action α and the *get*-action β. Indices indicate different instances of a single conceptual type [ZV92].

From these indexed conceptual structures, an initial *plan graph* is constructed to represent the agent's intentions, beliefs and expectations about the task it is to perform. To do this, the system consults the agent's knowledge of actions and plans (the *Action KB* and *Plan Library* in Figure 6.2), to develop hypotheses about the instructor-intended relationships between the specified actions (e.g., temporal relations, enablement relations, generation relations, etc.). The initial plan graph for our running example is shown in Figure 6.3.

This initial plan graph is further elaborated through processes of *reference resolution*, *plan inference*, *reference grounding*, *plan expansion* and *performance* (through *simulation*). To show the interaction between these processes and how they are used to elaborate the plan graph, we will contrast our example

"Go into the kitchen to get me the coffee urn."

with a somewhat different but related example

"Go into the kitchen and wash out the coffee urn."

In the first case, recall from the conceptual structure produced by the parser, that "get me" is interpreted as an instance of a "cause something to go somewhere" action. One recipe that the system has in its *Plan Library* for accomplishing this is shown in Figure 6.4. With respect to this recipe, "go" can be seen as a substep of "get" – which is one way it can serve the purpose of "get". (This action representation and the plan graph are described in greater detail in [EW92].)

Getting an object from one place to another requires first going to its location. This leads to the assumption, noted in Figure 6.3, that the coffee

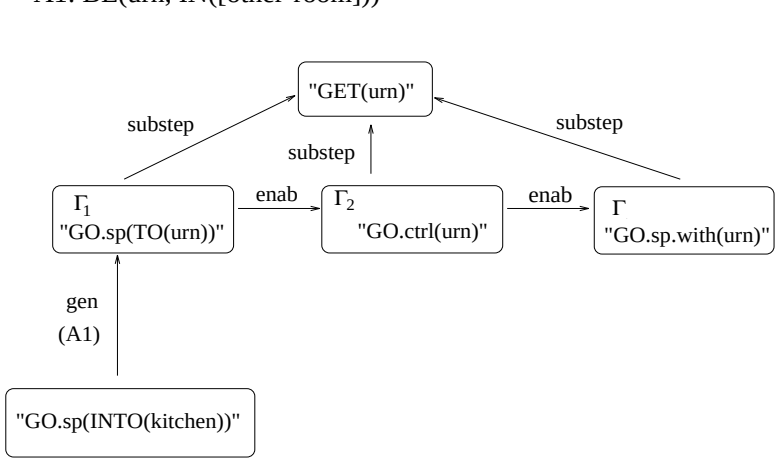

Figure 6.3: Initial Plan Graph: "Go into the kitchen and get me the coffee urn."

urn is in the kitchen. (The role of plan inference in instruction understanding is discussed in more detail in [Di 92, DW92].) Reference resolution cannot contribute any further constraints to the description "the coffee urn", since (1) there is no urn in the discourse context (nor anything that has a unique coffee urn associated with it), and (2) the assumption that the urn is in the kitchen is incompatible with its being unique in the current spatio-temporal context (which is the room next to the kitchen). Reference grounding does not attempt to associate this description with an object in the current spatio-temporal context, for the same reason. In fact, the agent will not attempt to ground this referring expression until it has entered the kitchen. (Whether the agent then succeeds immediately in grounding the expression will depend on whether the urn is perceivable – i.e., out in full view. We will discuss this shortly. In any case, the agent *expects* to be able to get access to the urn when it gets to the kitchen. This is what will drive it to *seek* the urn, if it is not in view when it gets to the kitchen.)

In the contrasting example "Go into the kitchen and wash out the coffee urn", the system again hypothesizes that the purpose relation between *go* and *wash-out* is a substep relation – but in this case, it is because washing out an object requires being at a washing site (e.g., a sink or tub). That kitchens usually have sinks gives further weight to this hypothesis.

Reference resolution may now contribute something to the agent's understanding of the definite expression "the coffee urn". While the discourse context does not provide evidence of a unique coffee urn, either directly or by association, there is also no evidence *against* the hypothesis that the urn is in the current spatio-temporal context. An initial hypothesis added by reference resolution that the urn is in the current space, if confirmed by reference

Header

$[\text{CAUSE}([\text{AGENT}]_i, [\text{GO}_{\text{Sp}}(j,k)])]$

$\left[\begin{array}{l}\text{FROM}([\text{AT}(j)]) \\ \text{TO}(l)\end{array}\right]_k$

Body

- $[\text{GO}_{\text{Sp}}([i, [\text{TO}([\text{AT}(j)])])]_{\gamma_1}$

- $[\text{CAUSE}(i, [\text{GO}_{\text{Ctrl}}(j, [\text{TO}([\text{AT}(i)])])])]_{\gamma_2}$

- $\left[\begin{array}{l}\text{GO}_{\text{Sp}}(i,k) \\ [\text{WITH}(j)]\end{array}\right]_{\gamma_3}$

- Annotations -

- γ_1 *enables* γ_2 *enables* γ_3

Qualifiers

- $[\text{NOT BE}_{\text{Sp}}(j, l)]$

Effects

- $[\text{BE}_{\text{Sp}}(j, l)]$

Figure 6.4: A *Move Something Somewhere* Action.

grounding, would lead plan expansion (through sub-goal generation) to get the agent over to its location. Failure of that hypothesis would lead to the alternative hypothesis that the coffee urn is in the kitchen. This is the same hypothesis as in the original example – it has just arisen in a different way.

The next thing to discuss is how the plan graph is expanded, and why it is expanded incrementally, as actions are performed in accordance with earlier elements of the plan graph. *How* it is expanded is through subgoal generation down to what we have called *annotated task actions*. This process makes use of a new kind of planner that (1) eschews pre-conditions in favor of decisions based on the agent's positive and negative intentions, and (2) takes upcoming intentions into account when deciding how to expand current goals, so as to put the agent in the best position with respect to satisfying those intentions. This planner, called *ItPlanS*, is described in more detail in [Gei92]. It is also

Figure 6.5: The Urn is Not Visible, so Cabinets will be Opened.

the source of the annotations of purpose in annotated task actions.

The main reason *why* the plan graph is expanded incrementally is that the agent does not have sufficient knowledge, before beginning to act, of what it will need to do later. In particular, AnimNL assumes that an agent cannot have up-to-date *knowledge* of any part of its environment that is outside its direct perception. (An AnimNL agent may know what non-visible parts of its environment *were* like, when it saw them earlier, and have expectations about what they *will be* like, when it sees them next, but its *knowledge* is limited to general truths about the world and to its direct perceptions.) As for the extent of the agent's perception, it is assumed that an agent cannot see into any space that has no *portal* open into the space the agent occupies. Thus AnimNL agents have to open doors, closets, boxes, etc., if they want to know what is inside, or go into other rooms to find out what is there.

What this means in our example is that only the plan graph node corresponding to "go into the kitchen" can be expanded – in this case, to "go over to the door", "open door", and "enter kitchen" – before the agent begins to act. The node corresponding to "go to the location of the coffee urn" cannot be expanded until the door has been opened and the agent can see whether or not the urn is visible. If it is visible, the agent can go to its location (Plate 6). If it is not visible, this same node must be expanded with actions corresponding to finding the urn – going through the kitchen cabinets one at a time looking for the urn, until it is found or all cabinets have been searched (Figure 6.5).

When an annotated task action becomes sufficiently specified for the agent

to be ready to commit to it and temporal dependencies permit such commitment, it is gated, triggering other, low-level planning processes (see Figure 6.2 below the "action gate"). An annotated task action is sufficiently specified if

- the action is "executable" (i.e., a *task action*, as described in Section 6.3).
- all actions temporally prior to it have been committed to. (Note that previous actions need not be completed before a new action is committed to: an agent can be (and usually is) doing more than one thing at a time.)
- its purpose has been determined.

It is worthwhile saying a bit more here about these purpose annotations, since we have come to believe they play a large part in low-level decisions about how to act. The kind of observations that motivates them are the following:

- when told to pick up a book and hand it to someone, an agent will grasp it one way;
- when told to pick up the same book and turn it over, an agent will commonly grasp it in quite a different way;
- when told to pick up the book and open to page 70, the agent will grasp it yet a third way.
- when just told to pick up the book, and nothing further, agents commonly grasp it, lift it up and wait expectantly for the next command.

These variations in grasp extend to such low-level features as grasp site and wrist position.

What we have tentatively concluded from such observations is that when agents don't know the purpose of some action they are told to perform, they put themselves into a position that easily supports subsequent action. Of course, always going into a position in which an agent is poised for subsequent action is very inefficient, especially when the agent knows what that subsequent action will be. In that case, he or she acts in such a way to smoothly and efficiently transition from one to the other. In AnimNL, purpose annotations (including "PFA" or *poised for action*) are there to allow the simulator, upon action commitment, to come up with the most effective ways of moving the agent's body for the given purpose. It is also why the system is designed to delay commitment until it knows the purpose of any task action or knows that the only thing it *can* know is PFA.

When an action is committed to, there is still further work to be done in order to determine the agent's behavior. In particular, one result of the experiment described in the previous section (Section 6.3) was our recognition of the need for tailoring an agent's behavior in carrying out an action to the

type of object given as an argument to that action. This follows from the fact that the same Natural Language verb is commonly used with different objects to denote very different behavior on an agent's part, and for a task animation to be correct, these differences must be depicted.

Consider, for example, the following definition (from [JCMM73]) of the word "remove" and sentences illustrating its use:

> **Remove**: to perform operations necessary to take an equipment unit out of the next larger assembly or system;
> to take off or eliminate; to take or move away.
> 1a. Remove bleed air shutoff valves.
> 1b. Remove bolts from nuts.
> 2. Remove paint.
> 3. Remove covers.

For each different object, the behavior needed to effect a "remove" is quite different. The question is whether to define a single *remove-action*, to use in animating both *Remove the paint* and *Remove the bolt*? The alternative – defining a multitude of animation procedures (e.g. *remove-paint, remove-bolt, remove-nut, remove-nail, remove-boxtop*, etc.) – appears expensive in terms of time and effort, and prone to error.

The solution we are adopting is to build a hybrid system. Instead of specifying complete definitions for each verb, we can identify the *core* or *kernel* action for a verb like *remove* in a fashion similar to that described in Section 6.2. We will use this core meaning, central to many different instantiations of the verb, in building the task-action. The missing information can be supplied by the verb's object: The knowledge base is object-oriented and so can store relevant information about individual objects. For example, one slot of information might be the DOFs an object has – a bolt "knows" (i.e., its geometric constraints specify) around which axis it turns. Joint and rotation information is already available in *Jack*.

The hybrid system would process an instruction by combining the information in the two representations – the underspecified definitions of the task-actions, in conjunction with the object-oriented knowledge base. By identifying which information is lacking in the task-actions, the system can try to supply that information from the knowledge base.

The advantages of a hybrid system is economy of both action definitions and of the object feature information to be stored. We no longer need to worry about developing separate definitions for each animation movement based on distinct verb/object pairs. Instead we take advantage of the compositional nature of the task-actions, and the object-oriented, hierarchical knowledge-base. Using these utilities, we can define a single animation definition for *remove* which will allow us to animate both *Remove the bolt* and *Remove nuts from bolts* while still distinguishing the instruction *Remove covers*.

Our work on using complex Natural Language instructions to motivate the behavior of animated agents is still in its infancy. There is much more to be done before it is a useful tool in the hands of task designers and human factors

engineers. On the other hand, we have begun to demonstrate its potential flexibility in accommodating the task behavior of an agent to the environment in which the task is being carried out and the agent's own capabilities.

Chapter 7

Epilogue

[1]To define a future for the work described in this book, it is essential to keep in mind the broad goals which motivated the efforts in the first place. Useful and usable software is desired, to be sure, but the vision of manipulating and especially *instructing* a realistically behaved animated agent is the greater ambition. Some of our visions for the near future are presented, not just for the sake of prognostication, but for its exciting prospects and possibilities.

Any discussion of the future of software must take into account the extraordinary pace of developments in the hardware arena. Even conservative predictions of hardware capabilities such as speed and capacity over the five year term lead one perilously close to science fiction. Accordingly, predictions of "better, faster, cheaper, more reliable, more fault tolerant, more highly parallel computers" are easy to make but do little to inform us of the applications these fantastic machines will facilitate. Rather, as general purpose computers improve in all these ways, specialized hardware solutions will decrease in importance and robust, usable software and symbiotic human-computer interfaces will remain the crucial link between a task and a solution.

Transforming research into practice is a lengthy process, consisting of a flow of concepts from ideas through algorithms to implementations, from testing and analysis through iterated design, and finally transfer of demonstratably workable concepts to external users and actual applications. This entire process may span years, from the initial description of the concept to a fielded system. The publication of initial results often breeds over-optimism and has been known to lead researchers to allow false expectations to arise in the minds of potential users, with unfortunate results. (Automatic machine translation of text, speech understanding, and early promises of Artificial Intelligence problem solving are good examples of premature speculations.) At the other end of the spectrum, however, are concepts which take a long time to work their way into mainstream technological consciousness. (3D computer graphics is a good example where concepts and even working systems pre-dated

[1]With the help of Mark Steedman.

widespread commercial availability by more than a decade.) So we will attempt to strike a balance in making speculations: while looking toward a long term research plan we will generally consider technology transfer to occur when serious but sympathetic users can experiment and accomplish real work with it. Our experience with software in the past is both our model and our promise for expecting new concepts to eventually reach potential users for evaluation and feedback.

7.1 A Roadmap Toward the Future

We seek to study the conceptual structure and limits of "virtual agents" in "simulated tasks" (VAST): the software and interface systems necessary to permit a user to describe, control, animate, analyze, interact with, and cooperate with multiple virtual computer-synthesized human models. We will remain cognizant of anticipated developments in underlying computer capabilities, but our principal intention will be to probe the intelligent software and user-interface issues. VAST focuses on the simulated human figure not just as a graphical entity but as an active, behaviorally complex *agent* who can follow instructions and autonomously negotiate its own way in the world.

Recall that our introduction emphasized certain simulation goals:

- Create an interactive computer graphics human model;
- Endow it with reasonable biomechanical properties;
- Provide it with "human-like" behaviors;
- Use this simulated human as an agent to effect changes in its world;
- Describe and guide its tasks through natural language instructions;

VAST augments this list to further improve the symbiosis between user and virtual agents:

- Control the agent through natural manual interfaces;
- Automatically generate explications or commentary on its behavior as sub-titles (text), voice-over or its own speech;
- Coordinate the activity of multiple agents engaged in a task.

We have probed the state-of-the-art in the first set of goals, but many problems and prospects remain for study. Indeed, there are other efforts that are advancing human modeling and animation. But our emphasis on interactivity and usability, especially by non-animators, outweighs mere visual beauty.

7.1.1 Interactive Human Models

While we are currently partway toward this goal with *Jack*, there are enhancements that are necessary before a virtual human looks and behaves realistically. Increases in hardware capability will certainly aid in the presentation of highly detailed models with smooth real-time response. The realistic and beautiful human models created by the Thalmanns [MTT90, MTT91a] are illustrative of the surface veracity possible under non-real-time conditions.

7.1.2 Reasonable Biomechanical Properties

Joint limits prevent unnatural adjacent body segment postures, but do nothing to prevent non-adjacent collisions. Collision-avoidance should be an implicit part of interactive manipulation for body integrity and natural appearance. In addition, clothing or equipment worn by the virtual human should demonstrate similar collision-free behavior.

Realistic strength models for the whole body and especially the torso should be incorporated into the model not only as a source of data but as an active resource for body motions. Preliminary work on strength-guided motion has shown feasibility, but more study of joint torque distribution strategies under comfort and loading constraints is needed. Ongoing "performance" models of the virtual human should be maintained throughout an interactive session or animation so that realistic assessments of workload and fatigue may be monitored.

7.1.3 Human-like Behaviors

We must continue to build a "primitive" behavior library so that the "innate" motion vocabulary of the virtual figure is as broad as possible. Ideally the behaviors can be "taught" to the figure rather than procedurally coded. Each behavior should enumerate the consequences of its execution so that higher level planning activities may take its potential effects into account.

Posture planning should take into account the spatial organization of the world and the virtual agent's desire to maximize effective behavior while minimizing useless movements (work). This can be done, in part, by having effective collision-avoidance schemes for articulated figures and, in part, by using symbolical spatial information to prune the high-dimensional numerical search space and move the figure into predictably useful postures. We already realize that classical AI planning paradigms are too weak for posture planning, and more reactive, incremental planners with "mental motion simulation" are needed.

Moving a virtual figure around an environment requires more than simple locomotion behavior: the behavioral repertoire and planner should understand crawling, climbing, jumping, sliding, etc. With this large repertoire of possible behaviors, a planner will be busy coordinating them all and sorting out priorities, even for simple activities.

7.1.4 Simulated Humans as Virtual Agents

The notion of skill level should be quantified with notions of context-sensitive execution time and [optimal] motion smoothness. Synthesized animations should be customized to the user's own body size and capabilities (or limitations) for training situations.

The environment in which the virtual humans work must be imported from any number of external CAD systems, preferably through standardized interfaces. During importation, perceptually and behaviorally significant features such as articulations, handles, removable parts, and open spaces (holes) should be recognized from the geometric model. We can expect some CAD systems to offer some of this data, but in general we should expect to build enhanced semantics into the models interactively or semi-automatically ourselves. For example, handles are needed to determine likely grasp points, and articulations, parts, and holes are needed for automatic generation of disassembly behaviors given only the object descriptions.

7.1.5 Task Guidance through Instructions

For a designer to use Natural Language instructions to describe and guide a virtual agent through a task, the overall system must know how to *understand* instructions and to *use* them appropriately in a given environment. Instructions must be understood in terms of intention – what is meant to be *achieved* in the world – and in terms of positive and negative constraints on the behavior used to achieve it. Instructions must be used to interpret features of the environment and to coordinate the agent's task-level knowledge and skills in acting to achieve its given goals. Advances in AI planning and execution are coming at a rapid rate, independent of our own work, and we will be incorporating those advances into VAST, to make the bridge to actual behavior.

7.1.6 Natural Manual Interfaces and Virtual Reality

The virtual figure should exist in a virtual 3D world that is accessible to a user with a minimum of training and little, if any, computer expertise. While mouse and keyboard input to *Jack* addresses some of these goals, it is still too "low level." By taking advantage of novel 3D, 6D, multiple 6D, and hand posture sensor input devices, the user's movements can be translated directly into virtual figure behaviors. The trick is *not* to make the mapping one-to-one, so the user exhausts herself flailing arms and twisting her body as the current Virtual Reality paradigms would have one do. Rather the mapping should have considerable intelligent "multipliers" so that the suggestion of action is enough to precipitate complete behaviors in the virtual human. We can already control the center of mass of the figure to effect a significant multiplier of input effort, this needs to be extended to arm gestures, view focus, and locomotion generation. We envision a strong corroborating role

from our animation from instructions work, such as speech-based commands. Minimal device encumbrances on the user are deemed essential.

7.1.7 Generating Text, Voice-over, and Spoken Explication for Animation

Animation for purposes of instruction in the performance of a task frequently requires spoken or written text, as well as graphic presentation, if it is to be understood. Negative instructions such as "warnings" or "cautions" provide an obvious example. While written captions can be adequate for some purposes, presenting few problems for animation, spoken language is more efficient and more engaging of the viewers' attention. For certain purposes even, a human face speaking in accompaniment to action is the most efficient and attention-holding device of all. Among the problems involved in including linguistic information in an on-going animation are: appropriately allocating information to the different modalities; integration of the two modalities over time; limitations of existing speech synthesizers with respect to intonation; integration of a facial animation with speech. All of these are tasks which the animator can in principle take over, but all of them, especially the last, are laborious. It would be highly desirable to automate all of them, especially if they are to be used in highly interactive animations, in which model-based synthesis-by-rule is required, rather than image based techniques.

One of the extensions to *Jack* is an animated facial model with a programming language for integrating its movements with a segmental representation of speech [PBS91, Pel91]. This effort is now focused on the following extensions:

- Provide an improved programming language for facial animation of speech.
- Provide a discourse semantics for spoken intonation in terms of appropriate knowledge representations.
- Perform automatic generation from such semantic representations of phonological representations of spoken explications, including appropriate intonational markers of contrast and background, for input to a speech synthesizer, with or without the facial animation program.

7.1.8 Coordinating Multiple Agents

Our principle goal has been getting a single agent to *behave plausibly* in response to multi-clause instruction steps. One goal for the longer term involves producing *sensible behavior* on the part of a single virtual agent and plausible behavior from a group of virtual agents engaged in a multi-agent task.

Coordinated multi-person simulations will be the next [large] step after an individual agent's actions can be effectively determined. Such simulations require physical, task and cognitive coordination among the agents. For

physical coordination, our particular efforts will focus on determining timing coordination, strength and workload distribution, and mutual achievement of spatial goals. In these multi-agent tasks, the interactive user may or may not be one of the participating agents.

Task coordination requires augmenting our task knowledge base with information on multi-agent tasks. This is essential both for understanding the text of multi-agent task instructions and for interpolating between explicit instructions, since instructions cannot (by virtue of not knowing the precise circumstances under which they will be carried out) specify everything.

For cognitive coordination among the agents, communication may be necessary to determine or alter the leadership role, initiate activity, keep it moving along, interrupt or abort it, or rest on imminent fatigue. Research on communication for coordinating multi-agent tasks is being carried on at other institutions [GS89, CL91, Loc91]. In the longer term, we look to importing the results of this research and incorporating it into the VAST framework. Until then, we will focus on single virtual agents or centrally-controlled multiple agents engaged in tasks in which communication is not required for coordination.

7.2 Conclusion

There are a multitude of other directions for virtual agent work. Some of these are automatic view control, a perceptual "sense," spatial reasoning for improved posture planning, recognizing and accommodating task failures, skill acquisition, flexible object interactions, animation presentation techniques, behaving with common sense, enriched instruction understanding, and speech-based agent animation. But all that's for a sequel.

Bibliography

[AAW74] M. A. Ayoub, M. M. Ayoub, and A. Walvekar. A biomechanical model for the upper extremity using optimization techniques. *Human Factors*, 16(6):585–594, 1974.

[Abb53] Edwin A. Abbott. *Flatland; a romance of many dimensions.* Dover, New York, NY, 1953.

[ABS90] T. Alameldin, N. Badler, and T. Sobh. An adaptive and efficient system for computing the 3-D reachable workspace. In *Proceedings of IEEE International Conference on Systems Engineering*, pages 503–506, 1990.

[AC87] Phillip Agre and David Chapman. Pengi: An implementation of a theory of activity. *Proceedings of the AAAI-87 Conference*, pages 268–272, June 1987.

[AD90] T.L. Anderson and M. Donath. Animal behavior as a paradigm for developing robot autonomy. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 145–168. MIT Press, 1990.

[AG85] W. W. Armstrong and Mark Green. The dynamics of articulated rigid bodies for purposes of animation. *The Visual Computer*, 1(4):231–240, 1985.

[AGL87] William Armstrong, Mark Green, and R. Lake. Near-real-time control of human figure models. *IEEE Computer Graphics and Applications*, 7(6):52–61, June 1987.

[AGR+81] M. M. Ayoub, C. F. Gidcumb, M. J. Reeder, M. Y. Beshir, H. A. Hafez, F. Aghazadeh, and N. J. Bethea. Development of an atlas of strengths and establishment of an appropriate model structure. Technical Report (Final Report), Institute for Ergonomics Research, Texas Tech Univ., Lubbock, TX, Nov. 1981.

[AGR+82] M. M. Ayoub, C. F. Gidcumb, M. J. Reeder, H. A. Hafez, M. Y. Beshir, F. Aghazadeh, and N. J. Bethea. Development of a female atlas of strengths. Technical Report (Final Report), Institute for Ergonomics Research, Texas Tech Univ., Lubbock, TX, Feb. 1982.

[AHN62] E. Asmussen and K. Heeboll-Nielsen. Isometric muscle strength in relation to age in men and women. *Ergonomics*, 5(1):167–169, 1962.

[Ala91] Tarek Alameldin. *Three Dimensional Workspace Visualization for Redundant Articulated Chains.* PhD thesis, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1991.

[Alb81] James S. Albus. *Brains, Behavior, and Robotics.* BYTE Books, McGraw-Hill, 1981.

[Ali90] Alias Research, Inc. *ALIAS V3.0 Reference Manual*, 1990.

[AM71] B. Anson and C. McVay. *Surgical Anatomy.* Saunders, Philadelphia,

PA, 1971.

[And60] E. Anderson. A semigraphical method for the analysis of complex problems. *Technometrics*, 2:381–391, 1960.

[And72] D. F. Andrews. Plots of high-dimensional data. *Biometrics*, 28(125), 1972.

[Ayo91] M. Ayoub. From biomechanical modeling to biomechanical simulation. In Edward Boyle, John Ianni, Jill Easterly, Susan Harper, and Medhat Korna, editors, *Human-Centered Technology for Maintainability: Workshop Proceedings*. Wright-Patterson Air Force Base, Armstrong Laboratory, June 1991.

[Bad75] Norman I. Badler. *Temporal scene analysis: Conceptual descriptions of object movements*. PhD thesis, Computer Science, Univ. of Toronto, Toronto, Canada, 1975. (Univ. of Pennsylvania, Computer and Information Science, Tech. Report MS-CIS-76-4).

[Bad76] Norman I. Badler. Conceptual descriptions of physical activities. *American Journal of Computational Linguistics*, Microfiche 35:70–83, 1976.

[Bad89] Norman I. Badler. A representation for natural human movement. In J. Gray, editor, *Dance Technology I*, pages 23–44. AAHPERD Publications, Reston, VA, 1989.

[Bar89] David Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Computer Graphics*, 23(3):223–232, 1989.

[BB78] Norman I. Badler and Ruzena Bajcsy. Three-dimensional representations for computer graphics and computer vision. *Computer Graphics*, 12(3):153–160, Aug. 1978.

[BB88] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constraints. *Computer Graphics*, 22(4):179–188, 1988.

[BBA88] P. G. Bullough and O. Boachie-Adjei. *Atlas of Spinal Diseases*. Lippincott, Philadelphia, PA, 1988.

[BBB87] Richard H. Bartels, John C. Beatty, and Brian A. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, Los Altos, CA, 1987.

[BBH+90] C. Blanchard, S. Burgess, Y. Harvill, J. Lanier, A. Lasko, M. Oberman, and M. Teitel. Reality built for two: A virtual reality tool. *Computer Graphics*, 24(2):35–36, 1990.

[BC68] R. Beckett and K. Chang. An evaluation of the kinematics of gait by minimum energy. *Journal of Biomechanics*, 1:147–159, 1968.

[BC89] Armin Bruderlin and Tom W. Calvert. Goal-directed, dynamic animation of human walking. *Computer Graphics*, 23(3):233–242, 1989.

[BCS90] R. D. Beer, H. J. Chiel, and L. S. Sterling. A biological perspective on autonomous agent design. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 169–186. MIT Press, 1990.

[Bec92] Welton M. Becket. Simulating adaptive autonomous behavior with recurrent neural networks. Technical report, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1992. To appear.

[BEK+81] P. Bapu, S. Evans, P. Kitka, M. Korna, and J. McDaniel. User's guide for COMBIMAN programs. Technical Report AFAMRL-TR-80-91, Univ. of Dayton Research Institute, Jan 1981. U.S.A.F. Report.

[Ber83] J. Bertin. *Semiology of Graphics, translated by W. J. Berg*. The Univ. of Wisconsin Press, 1983.

[BG86] Norman I. Badler and Jeffrey S. Gangel. Natural language input for hu-

man task description. In *Proc. ROBEXS '86: The Second International Workshop on Robotics and Expert Systems*, pages 137–148. Instrument Society of America, June 1986.

[BHJ+83] Michael Brady, John M. Hollerbach, Timothy L. Johnson, Tomas Lozano-Pérez, and Matthew T. Mason, editors. *Robot Motion: Planning and Control*. MIT Press, Cambridge, MA, 1983.

[Bie86] Eric Allan Bier. Snap-dragging. *Computer Graphics*, 20(3):233–240, 1986.

[Bie87] Eric Allan Bier. Skitters and jacks: Interactive positioning tools. In *Proceedings of 1986 ACM Workshop on Interactive 3D Graphics*, Chapel Hill, NC, Oct. 1987.

[Bie90] Eric Allan Bier. Snap-dragging in three dimensions. *Computer Graphics*, 24(2):193–204, March 1990.

[BKK+85] Norman I. Badler, Jonathan Korein, James U. Korein, Gerald Radack, and Lynne Brotman. Positioning and animating human figures in a task-oriented environment. *The Visual Computer*, 1(4):212–220, 1985.

[BKT86] K. Boff, L Kaufmann, and J Thomas, editors. *The Handbook of Perception and Human Performance*. John Wiley and Sons, New York, NY, 1986.

[BL88] Kenneth R. Boff and Janet E. Lincoln, editors. *Engineering Data Compendium*. Harry G. Armstrong Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, OH, 1988.

[BL89] J. Barraquand and J. Latombe. Robot motion planning: A distributed representation approach. Technical Report STAN-CS-89-1257, Computer Science, Stanford Univ., Stanford, CA, May 1989.

[Bli82] James F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, July 1982.

[BLL89a] J. Barraquand, B. Langlois, and J. Latombe. Numerical potential field techniques for robot path planning. Technical Report STAN-CS-89-1285, Computer Science, Stanford Univ., Stanford, CA, 1989.

[BLL89b] J. Barraquand, B. Langlois, and J. Latombe. Robot motion planning with many degrees of freedom and dynamic constraints. In *Fifth Intl. Sym. on Robotics Research (ISRR), Tokyo*, pages 1–10, 1989.

[BLP78] Edward G. Britton, James S. Lipscomb, and Michael E. Pique. Making nested rotations convenient for the user. *Computer Graphics*, 12(3):222–227, August 1978.

[BLP83] Rodney A. Brooks and Tomas Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. In *Proc. 8th Int. Joint Conf. Artificial Intelligence*, pages 799–806, 1983.

[BMB86] Norman I. Badler, Kamran H. Manoochehri, and David Baraff. Multi-dimensional input techniques and articulated figure positioning by multiple constraints. In *Proc. Workshop on Interactive 3D Graphics*, New York, NY, Oct. 1986. ACM.

[BMTT90] R. Boulic, Nadia Magnenat-Thalmann, and Daniel Thalmann. A global human walking model with real-time kinematic personification. *The Visual Computer*, 6:344–358, 1990.

[BMW87] Norman I. Badler, Kamran Manoochehri, and G. Walters. Articulated figure positioning by multiple constraints. *IEEE Computer Graphics and Applications*, 7(6):28–38, 1987.

[BN88] L. S. Brotman and A. N. Netravali. Motion interpolation by optimal

control. *Computer Graphics*, 22(4):309–315, 1988.

[Bob88] J. E. Bobrow. Optimal robot path planning using the minimum-time criteria. *IEEE Journal of Robotics and Automation*, 4(4):443–450, August 1988.

[Bod77] Margaret Boden. *Artificial Intelligence and Natural Man*. Basic Books, New York, NY, 1977.

[BOK80] Norman I. Badler, Joseph O'Rourke, and Bruce Kaufman. Special problems in human movement simulation. *Computer Graphics*, 14(3):189–197, July 1980.

[BOT79] Norman I. Badler, Joseph O'Rourke, and Hasida Toltzis. A spherical representation of a human body for visualizing movement. *IEEE Proceedings*, 67(10):1397–1403, Oct. 1979.

[BP88] Alain Berthoz and Thierry Pozzo. Intermittent head stabilization during postural and locomotory tasks in humans. In B. Amblard, A. Berthoz, and F. Clarac, editors, *Posture and Gait: Development, Adaptation, and Modulation*. Excerpta Medica, 1988.

[Bra84] Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. The MIT Press, 1984.

[Bre89] David E. Breen. Choreographing goal-oriented motion using cost functions. In N. Magnenat-Thalmann and D. Thalmann, editors, *State-of-the-Art in Computer Animation*, pages 141–151. Springer-Verlag, New York, NY, 1989.

[Bro83a] Rodney A. Brooks. Planning collision-free motions for pick-and-place operations. *Int. Journal of Robotics Research*, 2(4):19–44, Winter 1983.

[Bro83b] Rodney A. Brooks. Solving the find-path problem by good representation of free space. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(3):190–197, Mar 1983.

[Bro86] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, pages 14–23, April 1986.

[Bro90] Rodney A. Brooks. Elephants don't play chess. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 3–18. MIT Press, 1990.

[Bru88] Armin Bruderlin. Goal-directed, dynamic animation of bipedal locomotion. Master's thesis, Simon Fraser Univ., Vancouver, Canada, 1988.

[BS76] Maxine Brown and Stephen W. Smoliar. A graphics editor for Labanotation. *Computer Graphics*, 10(2):60–65, 1976.

[BS79] Norman I. Badler and Stephen W. Smoliar. Digital representations of human movement. *ACM Computing Surveys*, 11(1):19–38, March 1979.

[BS91] Jules Bloomenthal and Ken Shoemake. Convolution surfaces. *Computer Graphics*, 25(4):251–256, 1991.

[BSOW78] Norman I. Badler, Stephen W. Smoliar, Joseph O'Rourke, and Lynne Webber. The simulation of human movement. Technical Report MS-CIS-78-36, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1978.

[BW76] N. Burtnyk and M. Wein. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *Communications of the ACM*, 19(10):564–569, Oct. 1976.

[BW90] Aijaz A. Baloch and Allen M. Waxman. A neural system for behavioral conditioning of mobile robots. *IEEE International Joint Conference on Neural Networks*, 2:723–728, 1990.

[BwDL80] Irmgard Bartenieff and (with Dori Lewis). *Body Movement: Coping*

with the Environment. Gordon and Breach, New York, NY, 1980.

[BWKE91] Norman I. Badler, Bonnie L. Webber, Jugal K. Kalita, and Jeffrey Esakov. Animation from instructions. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 51–93. Morgan-Kaufmann, San Mateo, CA, 1991.

[CA84] D. B. Chaffin and G. B. J. Andersson. *Occupational Biomechanics*. John Wiley & Sons, 1984.

[Cal91] Tom Calvert. Composition of realistic animation sequences for multiple human figures. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 35–50. Morgan-Kaufmann, San Mateo, CA, 1991.

[Car72] Sven Carlsoo. *How Man Moves*. William Heinemann Ltd, 1972.

[Cat72] Edwin Catmull. A system for computer generated movies. In *Proceedings of ACM Annual Conference*, pages 422–431, August 1972.

[Cat78] E Catmull. The problems of computer-assisted animation. *Computer Graphics*, 12(3):348–353, August 1978.

[CB92] Wallace Ching and Norman I Badler. Fast motion planning for anthropometric figures with many degrees of freedom. In *IEEE Intl. Conf. on Robotics and Automation*, May 1992.

[CBR] K. Corker, A. Bejczy, and B. Rappaport. *Force/Torque Display For Space Teleoperation Control Experiments and Evaluation*. Cambridge, MA.

[CCP80] Tom Calvert, J. Chapman, and A. Patla. The integration of subjective and objective data in the animation of human movement. *Computer Graphics*, 14(3):198–203, July 1980.

[CCP82] Tom Calvert, J. Chapman, and A. Patla. Aspects of the kinematic simulation of human movement. *IEEE Computer Graphics and Applications*, 2(9):41–50, Nov. 1982.

[Ceb87] David Cebula. The semantic data model and large information requirements. Technical Report MS-CIS-87-72, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1987.

[Cen81] NASA Johnson Space Center. Space Shuttle Flight Data File Preparation Standards. Flight Operations Directorate, Operations Division, 1981.

[Che73] H. Chernoff. The use of faces to represent points in k-dimensional space graphically. *J. of the American Statistical Assoc.*, 68(342), 1973.

[CJ71] E. Y. Chao and D. H. Jacobson. Studies of human locomotion via optimal programming. *Mathematical Biosciences*, 6:239–306, 1971.

[CL91] P. Cohen and H. Levesque. Teamwork. *Nôus*, 25, 1991.

[CMS88] Michael Chen, S. Joy Mountford, and Abigail Sellen. A study in interactive 3-D rotation using 2-D control devices. *Computer Graphics*, 22(4):121–129, August 1988.

[Coo68] G.S. Cooper. A semantic analysis of English locative prepositions. Technical Report Report No. 1587, BBN: Clearinghouse for Federal Scientific and Technical Information, Springfield, VA, 1968.

[Del70] Cecily Dell. *A Primer for Movement Description*. Dance Notation Bureau, Inc., New York, NY, 1970.

[DH55] Jacques Denavit and Richard Hartenberg. A kinematic notation for

lower pair mechanisms based on matrices. *Journal of Applied Mechanics*, 23, 1955.

[Di 92] Barbara Di Eugenio. Goals and actions in Natural Language instructions. Technical Report MS-CIS-92-07, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1992.

[DLRG91] Bruce R. Donald, Jed Lengyel, Mark Reichert, and Donald Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. *Computer Graphics*, 25(4):327–336, July 1991.

[Don84] Bruce Donald. Motion planning with six degrees of freedom. Technical Report 791, MIT AI Lab, 1984.

[Don87] B. Donald. A search algorithm for motion planning with six degrees of freedom. *Artificial Intelligence*, 31:295–353, 1987.

[Doo82] Marianne Dooley. Anthropometric modeling programs – A survey. *IEEE Computer Graphics and Applications*, 2(9):17–25, Nov. 1982.

[Dru75] C. Drury. Application of Fitts' Law to foot-pedal design. *Human Factors*, 17, 1975.

[DW92] B. Di Eugenio and B. Webber. Plan recognition in understanding instructions. In *Proc. First Int'l Conference on Artificial Intelligence Planning Systems, College Park MD*, pages 52–61, June 1992.

[DX89] Bruce Donald and Patrick Xavier. A provably good approximation algorithm for optimal-time trajectory planning. In *IEEE Intl. Conf. on Robotics and Automation*, pages 958–963, 1989.

[EB90] Jeffrey Esakov and Norman I. Badler. An architecture for high-level human task animation control. In P. A. Fishwick and R. S. Modjeski, editors, *Knowledge-Based Simulation: Methodology and Application*, pages 162–199. Springer-Verlag, New York, NY, 1990.

[EB91] Jeffrey Esakov and Norman I. Badler. Animation from instructions – video tape. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*. Morgan-Kaufmann, San Mateo, CA, 1991. Videotape.

[EBJ89] Jeffery Esakov, Norman I. Badler, and M. Jung. An investigation of language input and performance timing for task animation. In *Graphics Interface '89*, pages 86–93, San Mateo, CA, June 1989. Morgan-Kaufmann.

[EC86] S. M. Evans and D. B. Chaffin. Using interactive visual displays to present ergonomic information in workspace design. In W. Karwowski, editor, *Trends in Ergonomics/Human Factors III*. Elsevier Science Publishers B.V. (North-Holland), 1986.

[EI91] Jill Easterly and John D. Ianni. Crew Chief: Present and future. In Edward Boyle, John Ianni, Jill Easterly, Susan Harper, and Medhat Korna, editors, *Human-Centered Technology for Maintainability: Workshop Proceedings*. Wright-Patterson Air Force Base, Armstrong Laboratory, June 1991.

[Emm85] Arielle Emmett. Digital portfolio: Tony de Peltrie. *Computer Graphics World*, 8(10):72–77, Oct. 1985.

[EP87] Ali Erkan Engin and Richard D. Peindl. On the biomechanics of human shoulder complex – I: Kinematics for determination of the shoulder complex sinus. *Journal of Biomechanics*, 20(2):103–117, 1987.

[EPE88] S. M. Evans, S. L. Palmiter, and J. Elkerton. The edge system: Er-

gonomic design using graphic evaluation. The Annual Meeting of the Human Factors Society, Los Angeles, CA, Oct. 1988.

[Esa90] Jeffrey Esakov. KB. Technical Report MS-CIS-90-03, Univ. of Pennsylvania, Philadelphia, PA, 1990.

[ET89] Ali Erkan Engin and S. T. Tumer. Three-dimensional kinematic modelling of the human shoulder complex – I: Physical model and determination of joint sinus cones. *Journal of Biomechanical Engineering*, 111:107–112, May 1989.

[ETW81] Kenneth B. Evans, Peter Tanner, and Marceli Wein. Tablet based valuators that provide one, two or three degrees of freedom. *Computer Graphics*, 15(3):91–97, 1981.

[Eva85] Susan M. R. Evans. *Ergonomics in manual workspace design: Current practices and an alternative computer-assisted approach.* PhD thesis, Center for Ergonomics, Univ. of Michigan, Ann Arbor, MI, 1985.

[Eva88] S. M. Evans. Use of biomechanical static strength models in workspace design. In *Proceedings for NATO Workshop on Human Performance Models in System Design*, Orlando, FL, May 1988.

[EW92] Barbara Di Eugenio and Michael White. On the interpretation of Natural Language instructions. In *Proceedings of 1992 International Conference on Computational Linguistics (COLING-92), Nantes, France*, 1992.

[Far88] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press, San Diego, CA, 1988.

[Fav84] Bernard Faverjon. Obstacle avoidance using an octree in the configuration space of a manipulator. In *IEEE Intl. Conf. on Robotics and Automation*, pages 504–512, 1984.

[FB85] K. Fishkin and B. Barsky. An analysis and algorithm for filling propagation. In *Proceedings Graphics Interface*, pages 203–212, 1985.

[Fet82] William Fetter. A progression of human figures simulated by computer graphics. *IEEE Computer Graphics and Applications*, 2(9):9–13, Nov. 1982.

[Fey86] Carl R. Feynman. Modeling the appearance of cloth. Master's thesis, Massachusetts Institute of Technology, 1986.

[Fis86] Paul A. Fishwick. *Hierarchical Reasoning: Simulating Complex Processes over Multiple Levels of Abstraction.* PhD thesis, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1986.

[Fis88] Paul A. Fishwick. The role of process abstraction in simulation. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):18–39, Jan/Feb. 1988.

[Fis90] K. Fishkin. Filling a region in a frame buffer. In A. Glassner, editor, *Graphics Gems*, pages 278–284. Academic Press, Cambridge, MA, 1990.

[Fit54] P. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47:381–391, 1954.

[FKU77] H. Fuchs, Z. Kedem, and S. Uselton. Optimal surface reconstruction from planar contours. *Communications of the ACM*, 20(10):693–702, Oct. 1977.

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

[FLP89] H. Fuchs, M. Levoy, and M. Pizer. Interactive visualization of 3D medi-

cal data. *IEEE Transactions on Computers*, pages 46–57, August 1989.

[FMHR87] S.S. Fisher, M. McGreevy, J. Humphries, and W. Robinett. Virtual environment display system. In *Proceedings of 1986 ACM Workshop on Interactive 3D Graphics*, Chapel Hill, NC, Oct. 1987.

[FP64] P. Fitts and J. Peterson. Information capacity of discrete motor responses. *Journal of Experimental Psychology*, 67(2), 1964.

[FS91] James A. Freeman and David M. Skapura. *Neural Networks: Algorithms, Applications, and Programming Techniques.* Addison Wesley, 1991.

[FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, Reading, MA, 1990. Second Edition.

[FW83] T. W. Finin and B. L. Webber. BUP – A Bottom Up Parser. Technical Report MS-CIS-83-16, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1983.

[FW88] David R. Forsey and Jane Wilhelms. Techniques for interactive manipulation of articulated bodies using dynamic analysis. In *Proceedings of Graphics Interface '88*, 1988.

[Gal80] C. R. Gallistel. *The Orginization of Action: A New Synthesis.* Lawrence Elerbaum Associates, Publishers, Hillsdale, NJ, 1980. Distributed by the Halsted Press division of John Wiley & Sons.

[Gan85] Jeffrey S. Gangel. A motion verb interface to a task animation system. Master's thesis, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, August 1985.

[Gei92] Christopher Geib. Intentions in means-end planning. Technical Report MS-CIS-92-73, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1992.

[GFS71] R. M. Goldwyn, H. P. Friedman, and T. H. Siegel. Iteration and interaction in computer data bank analysis. *Computer in Biomedical Research*, 4:607–622, 1971.

[Gir87] Michael Girard. Interactive design of 3D computer-animated legged animal motion. *IEEE Computer Graphics and Applications*, 7(6):39–51, 1987.

[Gir91] Michael Girard. Constrained optimization of articulated animal movement in computer animation. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 209–232. Morgan-Kaufmann, San Mateo, CA, 1991.

[GL90] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 729–734. Morgon Kaufmann Publishers, Inc., 1990.

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

[GM86] Carol M. Ginsberg and Delle Maxwell. Graphical marionette. In N. I. Badler and J. K. Tsotsos, editors, *Motion: Representation and Perception*, pages 303–310. Elsevier, North Holland, New York, NY, 1986.

[GMTT89] Jean-Paul Gourret, Nadia Magnenat-Thalmann, and Daniel Thalmann. Simulation of object and human skin deformations in a grasping task.

Computer Graphics, 23(3):21–30, 1989.

[Gol69] D. Goldfarb. Extension of Davidon's variable metric method to maximization under linear inequality and equality constraints. *SIAM Journal of Appl. Math.*, 17:739–764, 1969.

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

[Gom84] Julian E. Gomez. Twixt: A 3D animation system. In *Proc. Eurographics '84*, pages 121–133, New York, NY, July 1984. Elsevier Science Publishers B.V.

[Gou84] Laurent Gouzenes. Strategies for solving collision-free trajectories problems for mobile and manipulator robots. *Int. Journal of Robotics Research*, 3(4):51–65, Winter 1984.

[GP88] Ralph Guggenheim and PIXAR. Tin Toy (excerpt). *SIGGRAPH Video Review*, 38, 1988.

[GQB89] Marc Grosso, Richard Quach, and Norman I. Badler. Anthropometry for computer animated human figures. In N. Magnenat-Thalmann and D. Thalmann, editors, *State-of-the Art in Computer Animation*, pages 83–96. Springer-Verlag, New York, NY, 1989.

[GQO+89] Marc Grosso, Richard Quach, Ernest Otani, Jianmin Zhao, Susanna Wei, Pei-Hwa Ho, Jiahe Lu, and Norman I. Badler. Anthropometry for computer graphics human figures. Technical Report MS-CIS-89-71, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1989.

[GR82] K. Gupta and B. Roth. Design considerations for manipulator workspace. *ASME Journal of Mechanical Design*, 104:704–711, Oct. 1982.

[GRB+85] S. M. Goldwasser, R. A. Reynolds, T. Bapty, D. Baraff, J. Summers, D. A. Talton, and E. Walsh. Physician's workstation with real-time performance. *IEEE Computer Graphics and Applications*, 5(12):44–57, Dec. 1985.

[GS89] Barbara Grosz and Candice Sidner. Plans for discourse. In J. Morgan, P. Cohen, and M. Pollack, editors, *Intentions in Communication*. MIT Press, 1989.

[Gup86] K. Gupta. On the nature of robot workspace. *Int. Journal of Robotics Research*, 5:112–122, 1986.

[Gup90] Kamal Kant Gupta. Fast collision avoidance for manipulator arms: A sequential search strategy. *IEEE Transactions on Robotics and Automation*, 6(5):522–532, Oct 1990.

[Hac77] R. J. Hackathorn. ANIMA II: A 3-D color animation system. *Computer Graphics*, 11(2):54–64, July 1977.

[Hah88] James K. Hahn. Realistic animation of rigid bodies. *Computer Graphics*, 22(4):299–308, August 1988.

[Har75] J. A. Hartigan. Printer graphics for clustering. *Journal of Statistical Computation and Simulation*, 4:187–213, 1975.

[Hau89] Edward J. Haug, editor. *Concurrent Engineering of Mechanical Systems: Volume I.* The Univ. of Iowa, Iowa City, IA, 1989.

[HBD80] R. Harris, J. Bennet, and L. Dow. CAR-II – A revised model for crew assesment of reach. Technical Report 1400.06B, Analytics, Willow Grove, PA, 1980.

[HC90] Adele E. Howe and Paul R. Cohen. Responding to environmental

change. *Proceedings of the ARPA Workshop on Planning, Scheduling, and Control*, pages 85–92, Nov. 1990.

[HE78] Don Herbison-Evans. NUDES2: A numeric utility displaying ellipsoid solids. *Computer Graphics*, 12(3):354–356, Aug. 1978.

[HE82] Don Herbison-Evans. Real-time animation of human figure drawings with hidden lines omitted. *IEEE Computer Graphics and Applications*, 2(9):27–34, 1982.

[Her86] Annette Herskovits. Language and spatial cognition. In Aravind Joshi, editor, *Studies in Natural Language Processing*. Cambridge Univ. Press, Cambridge, England, 1986.

[HH87] C. Hoffmann and R. Hopcroft. Simulation of physical systems from geometric models. *IEEE Journal of Robotics and Automation*, RA-3(3):194–206, 1987.

[Hir77] Vicki Hirsch. Floorplans in Labanotation. Master's thesis, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1977.

[HJER86] V. H. Heyward, S. M. Johannes-Ellis, and J. F. Romer. Gender differences in strength. *Research Quarterly for Exercise and Sport*, 57(2):154–159, 1986.

[HKP91] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the theory of neural computation*. Addison Wesley, 1991.

[Hol81] W. H. Hollinshead. *Functional Anatomy of the Limbs and Back*. Saunders, Philadelphia, PA, 1981.

[Hol82] W. H. Hollinshead. *Anatomy for Surgeons*. Harper & Row, Philadelphia, PA, 1982.

[HP88] David R. Haumann and Richard E. Parent. The behavioral testbed: Obtaining complex behavior from simple rules. *The Visual Computer*, 4(6), 1988.

[HS85a] Pat Hanrahan and David Sturman. Interactive animation of parametric models. *The Visual Computer*, 1(4):260–266, 1985.

[HS85b] J. M. Hollerbach and K. C. Suh. Redundancy resolution of manipulators through torque optimization. In *IEEE Intl. Conf. on Robotics and Automation*, pages 1016–1021, St. Louis, MO, 1985.

[Hut70] Ann Hutchinson. *Labanotation*. Theatre Arts Books, New York, NY, 1970.

[Hut84] Ann Hutchinson. *Dance Notation*. Dance Horizons, New York, NY, 1984.

[Ibe87] T. Iberall. The nature of human prehension: Three dextrous hands in one. In *IEEE Intl. Conf. on Robotics and Automation*, pages 396–401, 1987.

[IC87] Paul M. Isaacs and Michael F. Cohen. Controlling dynamic simulation with kinematic constraints. *Computer Graphics*, 21(4):215–224, 1987.

[Imr83] S. N. Imrhan. *Modelling Isokinetic Strength of the Upper Extremity*. PhD thesis, Texas Tech Univ., 1983.

[IRT81] Verne T. Inman, Henry J. Ralston, and Frank Todd. *Human Walking*. Williams and Wilkins, Baltimore, MD, 1981.

[Jac90] Ray Jackendoff. *Semantic Structures*. MIT Press, Cambridge, MA, 1990.

[JCMM73] Reid Joyce, Andrew Chenzoff, Joseph Mulligan, and William Mallory. Fully proceduralized job performance aids. Technical Report AFHRL-

Tr-73-43(I), Air Force Human Resources Laboratory, Wright-Patterson AFB, 1973.

[JKBC91] Moon Jung, Jugal Kalita, Norman I. Badler, and Wallace Ching. Simulating human tasks using simple natural language instructions. In *Proc. Winter Simulation Conf.*, Phoenix, AZ, 1991.

[JM85] R. J. Jagacinski and D. L. Monk. Fitts' Law in two dimensions with hand and head movements. *Journal of Motor Behavior*, 17, 1985.

[Joh76] G. Johansson. Spatial-temporal differentiation and integration in visual motion perception. *Psychology Research*, 38:379–383, 1976.

[Jun92] Moon Jung. *Human-Like Agents with Posture Planning Ability.* PhD thesis, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1992.

[Kae90] Leslie P. Kaelbling. An architecture for intelligent reactive systems. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 713–728. Morgon Kaufmann Publishers, Inc., 1990.

[Kal90] Jugal Kumar Kalita. *Natural Language Control of Animation of Task Performance in a Physical Domain.* PhD thesis, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1990.

[Kar87] Robin Karlin. SEAFACT: A semantic analysis system for task animation of cooking operations. Master's thesis, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, Dec. 1987.

[Kar88] Robin Karlin. Defining the semantics of verbal modifiers in the domain of cooking tasks. In *Proc. of the 26st Annual Meeting of ACL*, pages 61–67, 1988.

[KB82] James U. Korein and Norman I. Badler. Techniques for goal directed motion. *IEEE Computer Graphics and Applications*, 2(9):71–81, Nov. 1982.

[KB90] Jugal Kalita and Norman I. Badler. Semantic analysis of a class of action verbs based on physical primitives. In *Proc. 12th Annual Conference of the Cognitive Science Society*, pages 412–419, Boston, MA, July 1990.

[KB91] Jugal Kalita and Norman I. Badler. Interpreting prepositions physically. In *Proc. AAAI-91*, pages 105–110, Anaheim, CA, 1991.

[Kee82] Steve W. Keele. Learning and control of coordinated motor patterns: The programming perspective. In J.A. Scott Kelso, editor, *Human Motor Behavior.* Lawrence Erlbaum Associates, 1982.

[KH81] B. Kleiner and J. A. Hartigan. Representating points in many dimensions by trees and castles. *Journal of American Statistical Association*, 76(374):260–269, 1981.

[KH83] C.A. Klein and C.H. Huang. Review of pseudoinverse control for use with kinematically redundant manipulators. *IEEE Transactions on Systems, Man and Cybernetics*, 13(2), 1983.

[Kha86] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. Journal of Robotics Research*, 5(1):90–98, 1986.

[Kha87] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal of Robotics and Automation*, RA-3(1):43–53, 1987.

[KKB88] Scott Kushnier, Jugal Kalita, and Norman I. Badler. Constraint-based temporal planning. Technical report, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1988.

[KN87] K. Kazerounian and A. Nedungadi. An alternative method for minimization of driving forces in redundant manipulators. In *IEEE Intl. Conf. on Robotics and Automation*, pages 1701–1706, Raleigh, NC, 1987.

[Kor85] James U. Korein. *A Geometric Investigation of Reach.* MIT Press, Cambridge, MA, 1985.

[KR79] M. E. Kahn and B. Roth. The near-minimum time control of open loop articulated kinematic chains. *Transactions of the ASME: Journal of Dynamic Systems, Measurement, and Control*, 93(3):164–172, 1979.

[KSC81] E. Kingsley, N. Schofield, and K. Case. SAMMIE – A computer aid for man-machine modeling. *Computer Graphics*, 15(3):163–169, Aug. 1981.

[KTV+90] James P. Karlen, Jack M Thompson, Havard I. Vold, James D. Farrell, and Paul H. Eismann. A dual-arm dexterous manipulator system with anthropomorphic kinematics. In *IEEE Intl. Conf. on Robotics and Automation*, 1990.

[Kum80] A. Kumar. *Characterization of Manipulator Geometry.* PhD thesis, Univ. of Houston, 1980.

[KW81] A. Kumar and K. Waldron. The workspace of a mechanical manipulator. *ASME Journal of Mechanical Design*, 103:665–672, July 1981.

[KZ86] Kamal Kant and Steven W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *Int. Journal of Robotics Research*, 5(3):72–89, Fall 1986.

[Lau76] L. L. Laubach. Comparative muscular strength of men and women: A review of the literature. *Aviation, Space, and Environmental Medicine*, 47(5):534–542, 1976.

[LCF76] G. D. Langolf, D. B. Chaffin, and J. A. Foulke. An investigation of Fitts' Law using a wide range of movement amplitudes. *Journal of Motor Behavior*, 8, 1976.

[Lee92] Philip L. Y. Lee. *Modeling Articulated Figure Motion with Physically- and Physiologically-Based Constraints.* PhD thesis, Mechanical Engineering and Applied Mechanics, Univ. of Pennsylvania, Philadelphia, PA, 1992.

[Lev77] Marc Levoy. A color animation system based on the multi-plane technique. *Computer Graphics*, 11(2):64–71, July 1977.

[Lev91] Libby Levison. Action composition for the animation of Natural Language instructions. Technical Report MS-CIS-91-28, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1991.

[Lif91] Kinetic Effects, Inc., Seattle, WA. *Life Forms User Manual*, 1991.

[Loc91] K. Lochbaum. An algorithm for plan recognition in collaborative discourse. In *Proc. 29^{th} Annual Meeting of the Assoc. for Computational Linguistics*, pages 33–38, Berkeley, CA, June 1991.

[Lou83] R. Louis. *Surgery of the Spine.* Springer-Verlag, New York, NY, 1983.

[LP81] Tomas Lozano-Pérez. Automatic planning of manipulator transfer movements. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(10):681–698, Oct 1981.

[LP83] Tomas Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, c-32(2):26–37, Feb 1983.

[LP87] Tomas Lozano-Pérez. A simple motion planning algorithm for general robot manipulators. *IEEE Journal of Robotics and Automation*, RA-

3(3):224–238, June 1987.

[LPW79] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, Oct. 1979.

[LRM88] Timothy Lohman, Alex Roche, and Reynaldo Martorell. *Anthropometric Standardization Reference Manual.* Human Kinetic Books, Champaign, IL, 1988.

[LWZB90] Philip Lee, Susanna Wei, Jianmin Zhao, and Norman I. Badler. Strength guided motion. *Computer Graphics*, 24(4):253–262, 1990.

[LY83] T. Lee and D. Yang. On the evaluation of manipulator workspace. *Journal of Mechanisms, Transmissions, and Automation in Design*, 105:70–77, March 1983.

[Mae90] Pattie Maes. Situated agents can have goals. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 49–70. MIT Press, 1990.

[Mau91] Ruth A. Maulucci. Personal communication, 1991.

[MB77] M. A. MacConaill and J. V. Basmajian. *Muscles and Movements, a Basic for Human Kinesiology.* R. E. Krieger, Huntington, NY, 1977.

[MB91] G. Monheit and N. Badler. A kinematic model of the human spine and torso. *IEEE Computer Graphics and Applications*, 11(2):29–38, 1991.

[McD89] J. W. McDaniel. Modeling strength data for CREW CHIEF. In *Proceedings of the SOAR 89 (Space Operations, Automation, and Robotics)*, Johnson Space Center, Houston, TX, July 1989.

[Mil88] Gavin S. P. Miller. The motion dynamics of snakes and worms. *Computer Graphics*, 22(4):169–178, 1988.

[Mil91] Gavin Miller. Goal-directed animation of tubular articulated figures or how snakes play golf. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 209–233. Morgan-Kaufmann, San Mateo, CA, 1991.

[Min86] Marvin Minsky. *The Society of Mind.* Simon and Schuster, 1986.

[MK85] A. A. Maciejewski and C. A. Klein. Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *Int. Journal of Robotics Research*, 4(3):109–117, 1985.

[MKK+88] J. McDaniel, M. Korna, P. Krauskopf, D. Haddox, S. Hardyal, M. Jones, and J. Polzinetti. User's Guide for CREW CHIEF: A computer graphics simulation of an aircraft maintenance technician. Technical report, Armstrong Aerospace Medical Research Laboratory, Human Systems Division, Air Force System Command, Wright-Patterson Air Force Base, OH, May 1988.

[MPZ90] Michael McKenna, Steve Pieper, and David Zeltzer. Control of a virtual actor: The roach. *Computer Graphics*, 24(2):165–174, 1990.

[MS86] A. Mital and N. Sanghavi. Comparison of maximum volitional torque exertion capabilities of males and females using common hand tools. *Human Factors*, 28(3):283–294, 1986.

[MTT85] Nadia Magnenat-Thalmann and Daniel Thalmann. *Computer Animation: Theory and Practice.* Springer-Verlag, New York, NY, 1985.

[MTT90] Nadia Magnenat-Thalmann and Daniel Thalmann. *Synthetic Actors in 3-D Computer-Generated Films.* Springer-Verlag, New York, NY, 1990.

[MTT91a] Nadia Magnenat-Thalmann and Daniel Thalmann. Complex models for animating synthetic actors. *IEEE Computer Graphics and Applications*,

11(5):32–44, Sept. 1991.

[MTT91b] Nadia Magnenat-Thalmann and Daniel Thalmann. Human body deformations using joint-dependent local operators and finite-element theory. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 243–262. Morgan-Kaufmann, San Mateo, CA, 1991.

[Muj87] C. Mujabbir. Workspaces of serial manipulators. Master's thesis, Mechanical Engineering and Applied Mechanics, Univ. of Pennsylvania, 1987.

[NAS78] NASA. *The Anthropometry Source Book*. NASA Reference Publication 1024, Johnson Space Center, Houston, TX, 1978. (Two volumes).

[NAS87] NASA. Man-System Integration Standards. NASA-STD-3000, March 1987.

[Nel85] Greg Nelson. Juno, a constraint-based graphics system. *Computer Graphics*, 19(3):235–243, 1985.

[NHK+85] H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, and K. Omura. Object modeling by distribution function and a method of image generation. In *Proc. Electronics Communication Conf.*, volume J68-D(4), 1985. (in Japanese).

[NHK86] NHK. Caron's world. *SIGGRAPH Video Review*, 24, 1986.

[NO87] Gregory Nielson and Dan Olsen Jr. Direct manipulation techniques for 3D objects using 2D locator devices. In *Proceedings of 1986 ACM Workshop on Interactive 3D Graphics*, Chapel Hill, NC, Oct. 1987.

[OB80] Joseph O'Rourke and Norman I. Badler. Model-based image analysis of human motion using constraint propagation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2(6):522–536, Nov. 1980.

[OO81] T. J. O'Donnell and Arthur J. Olson. GRAMPS – A graphics language interpreter for real-time, interactive, three-dimensional picture editing and animation. *Computer Graphics*, 15(3):133–142, 1981.

[Ota89] Ernest Otani. Software tools for dynamic and kinematic modeling of human motion. Technical Report MS-CIS-89-43, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1989. (MSE Thesis, Mechanical Engineering and Applied Mechanics, Univ. of Pennsylvania).

[Pau81] Richard Paul. *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, Cambridge, MA, 1981.

[PB88] Cary Phillips and Norman I. Badler. Jack: A toolkit for manipulating articulated figures. In *Proceedings of ACM SIGGRAPH Symposium on User Interface Software*, pages 221–229, Banff, Canada, Oct. 1988.

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

[PBS91] Catherine Pelachaud, Norman I. Badler, and Mark Steedman. Issues in facial animation. In *Computer Animation '91*, Geneva, Switzerland, 1991.

[Pel91] Catherine Pelachaud. *Communication and coarticulation in facial animation*. PhD thesis, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1991. Tech. Report MS-CIS-91-82.

[Pen86] Alex Pentland. Perceptual organization and the representation of natural form. *AI Journal*, 28(2):1–38, 1986.

[PMA+91] A. Pandya, J. Maida, A. Aldridge, S. Hasson, and B. Woolford. Devel-

opment of an empirically based dynamic biomechanical strength model. In *Space Operations Applications and Research Conf. Proc.*, pages 438–444, 1991. Vol. 2.

[Pot91] Caren Potter. The human factor. *Computer Graphics World*, pages 61–68, March 1991.

[Pow70] M. J. D. Powell. A hybrid method for nonlinear equations. In P. Rabinowitz, editor, *Numerical Methods for Nonlinear Algebraic Equations*. Gordon and Breach Science, 1970.

[PR90] Martha E. Pollack and Marc Ringuette. Introducing Tileworld: Experimentally evaluating an agent architecture. *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 183–189, 1990.

[Pra84] P. Prasad. An overview of major occupant simulation models. In *Proceedings of Society of Automotive Engineers*, 1984. Paper No. 840855.

[Pri81] Ellen Prince. Toward a taxonomy of given/new information. In P. Cole, editor, *Radical Pragmatics*, pages 223–255. Academic Press, New York, NY, 1981.

[PW89] A. Pentland and J. Williams. Good vibrations: Modal dynamics for graphics and animation. *Computer Graphics*, 23(3):215–222, 1989.

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

[RA90] David F. Rogers and J. Alan Adams. *Mathematical Elements for Computer Graphics*. McGraw-Hill, New York, NY, 1990. Second Ed.

[Ree83] William T. Reeves. Particle systems – A technique for modelling a class of fuzzy objects. *Computer Graphics*, 17(3):359–376, July 1983.

[Rey82] Craig W. Reynolds. Computer animation with scripts and actors. *Computer Graphics*, 16(3):289–296, July 1982.

[Rey87] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[Rey88] Craig W. Reynolds. Not bumping into things. SIGGRAPH course 27 notes: Developements in Physically-Based Modeling, 1988. G1–G13.

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

[RMTT90] Olivier Renault, Nadia Magnenat-Thalmann, and Daniel Thalmann. A vision-based approach to behavioral animation. *The Journal of Visualization and Computer Animation*, 1(1):18–21, 1990.

[Ros60] J. B. Rosen. The gradient projection method for nonlinear programming – I: Linear constraints. *SIAM Journal of Appl. Math.*, 8:181–217, 1960.

[Ros91] David A. Rosenbaum. *Human Motor Control*. Academic Press, 1991.

[Rot75] B. Roth. Performance evaluation of manipulators from a kinematics viewpoint. *NBS Special Publication*, pages 39–61, 1975.

[SB85] Scott Steketee and Norman I. Badler. Parametric keyframe interpolation incorporating kinetic adjustment and phrasing control. *Computer Graphics*, 19(3):255–262, 1985.

[Sch72] F. T. Schanne. *Three Dimensional Hand Force Capability Model for a Seated Person*. PhD thesis, Univ. of Michigan, Ann Arbor, MI, 1972.

[Sch82a] Richard Schmidt. More on motor programs. In J.A. Scott Kelso, editor, *Human Motor Behavior*. Lawrence Erlbaum Associates, 1982.

[Sch82b] Richard Schmidt. The schema concept. In J.A. Scott Kelso, editor, *Human Motor Behavior*. Lawrence Erlbaum Associates, 1982.

[Sch83] Christopher Schmandt. Spatial input/display correspondence in a stereoscopic computer graphics workstation. *Computer Graphics*, 17(3):253–261, July 1983.

[Sch90] J. H. Schmidhuber. Making the world differentiable: On using supervised learning fully recurrent networks for dynamic reinforcement learning and planning in non-stationary environments. Technical Report FKI-126-90, Technische Universität München, Febuary 1990.

[SEL84] SELF. The first 3-D computer exercises, Sept. 1984.

[SH86] G. Sahar and J. M. Hollerbach. Planning of minimum-time trajactories for robot arms. *Int. Journal of Robotics Research*, 5(3):90–100, 1986.

[Sha70] D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Math. Computation*, 24:647–664, 1970.

[Sha80] U. Shani. Filling regions in binary raster images: A graph-theoretic approach. *Computer Graphics*, 14(3):321–327, 1980.

[Sha88] Lokendra Shastri. A connectionist approach to knowledge representation and limited inference. *Cognitive Science*, 12(3):331–392, 1988.

[Sho92] Ken Shoemake. ARCBALL: A user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of SIGCHI '92*, 1992.

[Sim81] Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, 2 edition, 1981.

[SL87] S. Singh and M. C. Leu. Optimal trajectory generation for robotic manipulators using dynamic programming. *Transactions of the ASME: Journal of Dynamic Systems, Measurement, and Control*, 109:88–96, 1987.

[SP86] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *Computer Graphics*, 20(4):151–160, August 1986.

[SS83a] J. T. Schwartz and M. Sharir. On the piano movers' problem – I: The case of a two dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[SS83b] J. T. Schwartz and M. Sharir. On the piano movers' problem – II: General techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4:298–351, 1983.

[SSSN85] D. Schmitt, A. H. Soni, V. Srinivasan, and G. Naganthan. Optimal motion programming of robot manipulators. *Transactions of the ASME: Journal of Mechanisms, Transmissions, and Automation in Design*, 107:239–244, 1985.

[Ste83] G. Stern. BBOP – A program for 3-Dimensional animation. In *Nicograph '83*, Tokyo, Japan, 1983.

[Ste90] Mark Steedman. Gapping as constituent coordination. *Linguistics and Philosophy*, 13:207–263, 1990.

[Stu84] David Sturman. Interactive key frame animation of 3-D articulated models. In *Proc. Graphics Interface '84*, pages 35–40, Ottawa, Canada, 1984.

[Sug81] D. Sugimoto. Determination of extreme distances of a robot hand. *ASME Journal of Mechanical Design*, 103:631–636, July 1981.

[Sun91] Ron Sun. Neural network models for rule-based reasoning. In *IEEE International Joint Conference on Neural Networks*, pages 503–508, Sin-

gapore, 1991.

[TA75] G. J. Torotra and N. P. Anagnostakos. *Principles of Anatomy and Physiology*. Canfield Press, New York, NY, 1975.

[TJ81] Frank Thomas and Ollie Johnson. *Disney Animation: The Illusion of Life*. Abbeville Press, New York, NY, 1981.

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *Computer Graphics*, 21:205–214, 1987.

[TS81] Y. Tsai and A. Soni. Accessible region and synthesis of robot arms. *ASME Journal of Mechanical Design*, 103:803–811, Oct. 1981.

[TS83] Y. Tsai and A. Soni. An algorithm for the workspace of a general n-R robot. *ASME Journal of Mechanical Design*, 105:52–57, July 1983.

[Tsa86] M. Tsai. *Workspace Geometric Characterization and Manipulability of Industrial Robots*. PhD thesis, Ohio State Univ., 1986.

[TST87] Yosuke Takashima, Hideo Shimazu, and Masahiro Tomono. Story driven animation. In *CHI + GI '87 Proceedings*, pages 149–153. ACM SIGCHI, 1987.

[Tur63] A. M. Turing. Computing machinery and intelligence. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 11–35. McGraw-Hill, New York, NY, 1963.

[VB90] S. A. Vere and T. W. Bickmore. A basic agent. *Computational Intelligence*, 6:41–60, 1990.

[Vij85] R. Vijaykumar. Robot manipulators – Workspaces and geometrical dexterity. Master's thesis, Mechanical Engineering, Ohio State Univ., Columbus, OH, 1985.

[Wav89] Wavefront Technologies. *MODEL User's Manual Version 6.0*, 1989.

[WB85] Jane Wilhelms and Brian A. Barsky. Using dynamics for the animation of articulated bodies such as humans and robots. In *Proc. Graphics Interface '85*, pages 97–104, Montreal, Canada, 1985.

[WB92] Susanna Wei and Norman I. Badler. Graphical displays of human strength data. *Visualization and Computer Animation*, 3(1):13–22, 1992.

[Wei86] Jerry Weil. The synthesis of cloth objects. *Computer Graphics*, 20(4):49–54, 1986.

[Wei90] Susanna Wei. *Human Strength Database and Multidimensional Data Display*. PhD thesis, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1990.

[Wel71] K. Wells. *Kinesiology, the Scientific Basis of Human Action*. Saunders, Philadelphia, PA, 1971.

[Wes73] Barry D. Wessler. *Computer-assisted visual communication*. PhD thesis, Univ. of Utah, Salt Lake City, UT, 1973.

[WFB87] Andrew Witkin, Kurt Fleisher, and Alan Barr. Energy constraints on parameterized models. *Computer Graphics*, 21(3):225–232, 1987.

[Whi72] D. E. Whitney. The mathematics of coordinated control of prostheses and manipulators. *J. Dynamic Systems, Measurement, and Control, Transaction ASME*, 94:303–309, 1972. Series G.

[Wil75] F. Wilson, editor. *The Musculoskeletal System*. Lippincott, Philadelphia, PA, 1975.

[Wil82] K. D. Willmert. Visualizing human body motion simulations. *IEEE Computer Graphics and Applications*, 2(9):35–38, Nov. 1982.

[Wil86] Jane Wilhelms. Virya – A motion editor for kinematic and dynamic

animation. In *Proc. Graphics Interface '86*, pages 141–146, Vancouver, Canada, 1986.

[Wil87] Jane Wilhelms. Using dynamic analysis for realistic animation of articulated bodies. *IEEE Computer Graphics and Applications*, 7(6):12–27, 1987.

[Wil91] Jane Wilhelms. Dynamic experiences. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 265–279. Morgan-Kaufmann, San Mateo, CA, 1991.

[Win90] David A. Winter. *Biomechanics and Motor Control of Human Movement*. Wiley Interscience, New York, NY, 1990. Second Edition.

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

[WMW86] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The Visual Computer*, 2:227–234, 1986.

[WS90] Jane Wilhelms and Robert Skinner. A 'notion' for interactive behavioral animation control. *IEEE Computer Graphics and Applications*, 10(3):14–22, May 1990.

[WSB78] Lynne Weber, Stephen W. Smoliar, and Norman I. Badler. An architecture for the simulation of human movement. In *Proc. ACM Annual Conf.*, pages 737–745, Washington, DC, 1978.

[WW90] Andrew Witkin and William Welch. Fast animation and control of nonrigid structures. *Computer Graphics*, 24(4):243–252, 1990.

[Yeo76] B. P. Yeo. Investigations concerning the principle of minimal total muscular force. *Journal of Biomechanics*, 9:413–416, 1976.

[YL83] D. Yang and T. Lee. On the workspace of mechanical manipulators. *Journal of Mechanisms, Transmissions, and Automation in Design*, 105:62–69, March 1983.

[YN87] V. Yen and M. L. Nagurka. Suboptimal trajectory planning of a five-link human locomotion model. In *Biomechanics Proceedings*, 1987.

[ZB89] Jianmin Zhao and Norman I. Badler. Real time inverse kinematics with joint limits and spatial constraints. Technical Report MS-CIS-89-09, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1989.

[Zel82] David Zeltzer. Motor control techniques for figure animation. *IEEE Computer Graphics and Applications*, 2(9):53–59, Nov. 1982.

[Zel84] David Zeltzer. *Representation and Control of Three Dimensional Computer Animated Figures*. PhD thesis, The Ohio State Univ., 1984.

[Zel91] David Zeltzer. Task-level graphical simulation: Abstraction, representation, and control. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 3–33. Morgan-Kaufmann, San Mateo, CA, 1991.

[ZV92] J. Zwarts and H. Verkuyl. An algebra of conceptual structure: An investigation into Jackendoff's conceptual semantics. *Linguistics and Philosophy*, 1992. To appear.

Index

Norman I. Badler is the Cecilia Fitler Moore Professor and Chair of Computer and Information Science at the University of Pennsylvania and has been on that faculty since 1974. Active in computer graphics since 1968, his research focuses on human figure modeling, manipulation, and animation. Badler received the BA degree in Creative Studies Mathematics from the University of California at Santa Barbara in 1970, the MSc in Mathematics in 1971, and the Ph.D. in Computer Science in 1975, both from the University of Toronto. He is Co-Editor of the Journal *Graphical Models and Image Processing*. He also directs the Computer Graphics Research Laboratory with two full time staff members and about 40 students.

Cary B. Phillips received his PhD in Computer and Information Science from the University of Pennsylvania in 1991, where he was a member of the Technical Staff in the Computer Graphics Research Lab from 1988-91. He is the principal designer and implementor of Jack. He received his BES and MSE degrees in Electrical Engineering and Computer Science from The Johns Hopkins University in 1985. Phillips is currently on the R&D staff at Pacific Data Images, a computer animation production company in Sunnyvale, California.

Bonnie Lynn Webber is a Professor of Computer and Information Science at the University of Pennsylvania. She is known for her work in Computational Linguistics – in particular, discourse processing and human-computer interaction. She did her undergraduate work at MIT and her graduate work at Harvard, receiving a PhD in 1978. While at Harvard, she was also a Senior Scientist at Bolt Beranek and Newman Inc, Cambridge MA, where she contributed to the first DARPA Speech Understanding Initiative. She has co-editted several books, including *Readings in Artificial Intelligence* and *Readings in Natural Language Processing*.

PLATES

1. The *Jack* figure inside a Deere and Company off-road vehicle. Vehicle courtesy Jerry Duncan. Image by Cary Phillips and John Granieri.

2. Apache helicopter cockpit with simulated operator. The helicopter cockpit and helmet models were supplied by Barry Smith of the A^3I project at NASA Ames Research Center. Image by Pei-Hwa Ho, Eunyoung Koh, Jiahe Lu, Welton Becket, Catherine Pelachaud, Soetjianto, and Cary Phillips.

3. Graduation Day. Clothes modeling by Lauren Bello, Welton Becket, and Eunyoung Koh.

4. Translucent workspace for left hand reach. Image by Tarek Alameldin. Apache helicopter model courtesy Barry Smith at NASA Ames Research Center.

5. What you see is what you can reach with the left hand. Image by Tarek Alameldin. Apache helicopter model courtesy Barry Smith at NASA Ames Research Center.

6. "Go into the kitchen and get me the coffee urn." Scene from the "animation from instructions" (AnimNL) project. Image by Leanne Hwang, David Haynes, Brian Stokes, Moon Jung, and Aaron Fuegi.