May 1994

Posture Interpolation with Collision Avoidance

Norman I. Badler
University of Pennsylvania, badler@seas.upenn.edu

Ramamani Bindiganavale
University of Pennsylvania

John P. Granieri
University of Pennsylvania

Susanna Wei
St. Joseph's University

Xinmin Zhao
University of Pennsylvania

Posture Interpolation with Collision Avoidance

Abstract
While interpolating between successive postures of an articulated figure is not mathematically difficult, it is much more useful to provide postural transactions that are behaviorally reasonable and that avoid collisions with nearby objects. We describe such a posture interpolator which begins with a number of pre-defined static postures. A finite state machine controls the transactions from any posture to a goal posture by finding the shortest path of required motion sequences between the two. If the motion between any two postures is not collision free, a collision avoidance strategy is invoked and the posture is changed to one that satisfies the required goal while respecting object and agent integrity.

Keywords
human figure animation, motion control, posture interpolation, goal-directed behavior, collision avoidance potential fields, self-collision, computer animation

Posture Interpolation with Collision Avoidance

Norman I. Badler, Ramamani Bindiganavale, John P. Granieri, Susanna Wei*, Xinmin Zhao
Center for Human Modeling and Simulation
University of Pennsylvania
Philadelphia, Pennsylvania 19104-6389

Abstract

While interpolating between successive postures of an articulated figure is not mathematically difficult, it is much more useful to provide postural transitions that are behaviorally reasonable and that avoid collisions with nearby objects. We describe such a posture interpolator which begins with a number of pre-defined static postures. A finite state machine controls the transitions from any posture to a goal posture by finding the shortest path of required motion sequences between the two. If the motion between any two postures is not collision free, a collision avoidance strategy is invoked and the posture is changed to one that satisfies the required goal while respecting object and agent integrity.

Keywords

Human figure animation, motion control, posture interpolation, goal-directed behavior, collision avoidance, potential fields, self-collision, computer animation.

1 Introduction

Postures are a very important aspect of human figure simulation. A static and recognizable posture such as stand, sit, crawl, prone, or supine can be defined by the relative positioning of various parts of the body. When simulating human motions, an animation system must create suitable motions between these static postures. Most "classical" animation systems use keyframe or key-pose animation [1] for moving an object from one posture to another. An interactive positioning program is used to define the location and orientation of parts of the object at various key times. The in-between frames are then calculated by interpolating the individual joint angles. Key based techniques, though satisfactory for some kinds of animation, do not directly address other important human motion factors such as balance and collision avoidance. With keys alone, the animator must be willing and able to devote considerable care and attention to the handcrafting of suitable in-between key poses.

Real-time graphical manipulation and display has accelerated research into the control and animation of human-like characters [2, 3]. Among the newer animation techniques are those based on physics [4, 5], dynamics [6, 7], optimization [8], cost functions [9], behavioral networks[10], control theoretic state-spaces [11, 12, 13, 14], and strength [15]. In many of these approaches, postures of a human-like figure are specified in advance – often as key poses or goal configurations – and then animated by applying forces, energy minimization, geometric constraints, or just straightforward joint angle interpolation. The "in-between" postures are thus determined by physics or mathematics rather than psychomotor considerations. Realistic motion (to be distinguished from just "smooth" motion) requires that the movements into or between given postures be biomechanically motivated, cognizant of physical stability, and free from intersections with self as well as nearby objects.

We present here a computational model for collision-free postural transitions that are behaviorally reasonable. The posture interpolator begins with a number of pre-defined static postures. A finite state machine controls the transitions from any posture to a goal posture by finding the shortest path of required (but pre-determined) motion sequences between the two. If the motion between any two postures is not collision free, a collision avoidance strategy is invoked and the posture is changed to one that satisfies the required goal while respecting object and agent integrity. Since postures are treated as goal states based on constraints and other geometric relationships, perturbations caused by collision avoidance are seamlessly managed by the constraint satisfaction process.

Our collision avoidance system is based on a distributed sensor model. On a modern workstation it can avoid collisions during interactive manipulation of a human figure model with 136 degrees of freedom. Unlike traditional motion planning approaches [16], our system's collision avoidance is built as a reactive behavior of the agent [2]. It detects imminent collisions and reacts by establishing instantaneous constraints which are then solved together with other human behavior constraints. As there are often several alternative movements that can be executed to avoid collisions, the algorithm can consider the agent's comfort level (required joint torques, balance, etc.) at different postures and choose a motion sequence that minimizes discomfort.

In the following sections we discuss our motion system, the posture representation and interpolation pro-

*Susanna Wei is an assistant professor at the Dept. of Mathematics and Computer Science, St. Joseph's University, Philadelphia, Pennsylvania 19131-1395

0-8186-6240-9/94 $3.00 © 1994 IEEE

cess, and real-time collision avoidance. We conclude with methods to avoid local minima (getting "stuck") and improving the human-like qualities of the motion.

2 Motion System

The human figure used in our system, *Jack*®, can be controlled through a set of kinematic constraints that make the figure *behave* in a certain way [2]. In the motion system, we animate goals and behavioral parameters, and let the behavior functions and constraints create the motion on the joint angles. *Jack*'s motion system is built of several layers of functionality, which all work together to produce desired motion: a database, geometric constraints, behaviors, and motion control.

Database: There is a language for defining the 3D articulated human figure's joint structure and segment geometry, along with proper joint limits, segment mass, moment of inertia, and joint strength.

Constraints: A *constraint* defines a desired geometric relationship (such as `point-to-point`) between two objects (the *end effector* and the *goal*) in the environment. It also specifies which variables (a set of joints) may be changed to achieve the relationship. We use a variety of methods to solve the constraints, but mostly depend on iterative inverse kinematics [17].

Behaviors: Layered on top of the constraints, are *behaviors* [18]. Behaviors manage and coordinate the underlying constraints which, in turn, control the position and posture of the figure. The behaviors are grouped around the major body parts: the torso, arms and hands, legs and feet, and head. The behaviors can also interact. For example, the balance behavior (which controls the location of the center of mass) must work with the feet behaviors, in case the feet need to be moved to maintain balance.

Each behavior has a set of control parameters. For example, the arm and hand behavior has these parameters: (1) the type of control (`"hold global location"`, `"hold relative location"`, `"release"`) (2) the end-effector location (`"palm"`, `"lower arm"`, `"attached object"`) (3) the joints involved, `"to shoulder"`, `"to waist"`, and (4) a reference object (or just a homogeneous transform) in the environment.

Each behavioral control has a corresponding interactive manipulation primitive which usually takes the same parameters as the behavioral control, but allows the user to interactively adjust the goal of the behavior. For example, the user can manipulate an arm by dragging a 3D reach site in space; the arm will move according to the parameters in effect for that arm's behavior (e.g. include spine in the reach if using the `"to waist"` parameter).

Motion control: The motion system extends the interactive manipulation and behavioral control primitives by expressing movement of a part of the body from one place to another over a specific time interval. The movement is specified in terms of the final position, or *goal*, and the parameters of how to get there (similar to the parameters used for the behavioral control, with additional parameters for velocity). The *initial* position of the motion, however, is **implicit** – wherever the body part is when the motion starts. This fact is exploited in the collision avoidance functions described later.

Each motion control has 3 major functions: (1) a *preaction* function which is executed at the starting time of the motion and usually picks up the current position of the body part, (2) an *apply* function, executed on every frame of the motion, which controls the interpolation of the goal as well as the other behavioral parameters of the motion, and finally (3) a *postaction* function, executed after the last *apply*, which sets the final behavior of the body part in question. Since the behavioral parameters are all ultimately implemented via constraints, the constraint solving process has the ability to blend overlapping motions to create a final solution for the posture of the body part. (There are several blending parameters used in each motion type.) Of course, we also have the ability to perform forward kinematic joint motions which just interpolate a joint angle over time.

3 Posture Transition

As humans, we automatically move from one posture to another without giving any thought to the exact sequence of motions involved. But when these posture transitions have to be simulated, explicit knowledge about these motions must be supplied. For example, when moving from a standing posture to a squatting posture, we need to lower our upper body and bend our knees a little. What we are effectively doing is just lowering our center of mass. If we instruct *Jack* to lower his center of mass, he automatically tries to maintain balance by bending the knees a little thus resulting in the desired posture. When we want to change postures from squatting to kneeling on both knees, we first lower our upper body and bend the knees slightly. Then, keeping one foot planted firmly on the ground for support, we lower our body till the other knee rests completely on the ground. Lastly, we move the first leg until that knee rests on the ground, too. This sequence is more complex than just squatting, so to make *Jack* do it we need to provide these intermediate postural transitions. Each posture transition involves a number of motions (in the motion control sense described above) which may overlap and must be synchronized in time.

Postural transitions are effected by using a Finite State Machine (FSM) in which each state is a key posture. First, a set of goal postures – standing, sitting, squatting, kneeling, prone, etc. – have to be identified, These form the static postures (Fig. 1). Next, a system has to be built which enables posture transitions between any two of these postures. The simplest method is by considering each posture transition to consist of simpler posture transitions. As a FSM, the static postures form the nodes and the motion sequences form the arcs (Fig. 2). For every posture, all its adjacent static postures are identified, where adjacent postures can be defined to be two nodes in the transition graph connected together by a single arc. For example, in Fig. 2, the only adjacent static posture for `Stand` is `Squat`. But the adjacent postures

Figure 1: Static Postures

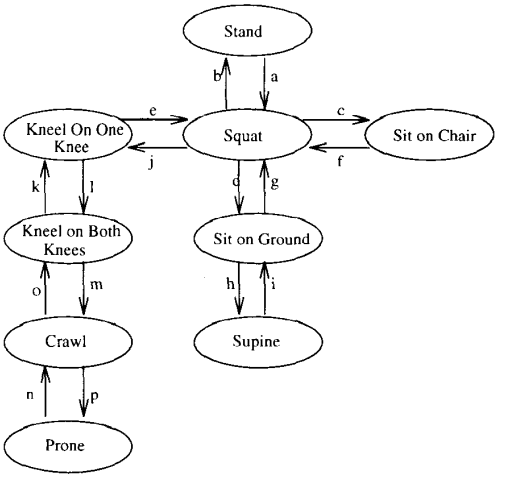

Figure 2: Posture Transition Graph

of **Squat** are **Stand**, **Sit on Chair**, **Sit on Ground**, and **Kneel on One Knee**.

The number of motions involved in the posture transition between any two adjacent postures is usually very small. These simple transitions are easy to define and can be combined together to form a complex posture transition. For example, the posture transition from **Stand** to **Supine** (lying down on back) would consist of three smaller posture transitions, namely **Stand** to **Squat**, **Squat** to **Sit on Ground**, and **Sit on Ground** to **Supine**. This same set of posture transitions from **Stand** to **Supine** can be obtained from the transition graph by tracing the shortest path between the **Stand** and **Supine** nodes. Thus the problem of complex posture transitions reduces to finding (and executing) the shortest path in the transition graph between the start and goal postures. The FSM makes this easy. The complete set of posture transitions thus derived can be compactly represented by a matrix as shown in Fig. 3. A straightforward algorithm (Fig. 4) can then be used to generate the motions along this shortest path.

Final / Initial	0	1	2	3	4	5	6	7	8
0	-1	a	1	1	1	1	1	1	1
1	b	-1	c	d	3	e	5	5	5
2	1	f	-1	1	1	1	1	1	1
3	1	g	1	-1	h	1	1	1	1
4	3	3	3	i	-1	3	3	3	3
5	1	j	1	1	1	-1	k	6	6
6	5	5	5	5	5	1	-1	m	7
7	6	6	6	6	6	6	n	-1	o
8	7	7	7	7	7	7	7	p	-1

0 Stand
1 Squat
2 Sit on Chair
3 Sit on Ground
4 Supine
5 Kneel on One Knee
6 Kneel on Both Knees
7 Crawl
8 Prone

Figure 3: Posture Transition Matrix

4 Collision Avoidance for Human Motions

One possible limitation to direct implementation of posture transitions arises from impossible or inappropriate movements caused by blind achievement of the posture goals. In real life, if there is any obstacle in our path either when we are walking or moving from one position to another, we automatically avoid it and try to reach our goal position by circumventing the obstacle. The posture transitions in the FSM cannot explicitly represent all conceivable environmental obstacles. We have therefore augmented the posture transition interpreter to include collision avoidance during any posture transition. If any obstacle is present in

Transit (x:posture,y:posture,M:Matrix)

1. If M(x,y) = -1 then stop.
2. If M(x,y) contains a motion sequence (denoted by lower-case characters), then execute the sequence and stop.
3. If M(x,y) contains posture x, then execute M(x,z) followed by M(z,y).

Figure 4: Posture Transition Algorithm

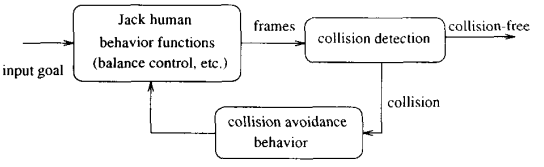

Figure 5: The architecture of the system

the path, the motion is changed to avoid the collision while still reaching the goal posture (if possible) in the best possible way.

Collision avoidance has been an active research topic in Robotics. In general there have been two directions in this area: motion planning and reactive behavior. In motion planning, a collision-free path is planned and the agent is instructed to follow the path [16, 19]. In the latter approach, collision avoidance is built as a reactive behavior of the agent. When the agent is close to an obstacle, it produces a repulsive force which will push the agent away and thus avoid collision [20].

Both approaches have advantages and disadvantages. The reactive behavior approach has a clear advantage in performance. In most cases it can be done in real time or near real time, even when the number of DOFs involved is large (e.g., over 5). Reactive methods can even avoid moving obstacles. These methods suffer, however, from the local minima problem: the agent may get trapped in a local minima posture and fail to reach the goal posture, even if the goal posture is reachable.

While the motion planning approach does not suffer the local minima problem in general, it is usually too slow to be used in an interactive (changing) environment, especially when the number of DOFs is large (e.g., 5 or more).

Collision avoidance for human motions has additional stringent requirements. First of all, the human body has many DOFs (*Jack* has over 130), compared to the few DOFs in a typical robot arm. So the method used must be able to handle a large number of DOFs[1]. Second, balance is not usually a problem in robot path planning[2]. In human motion this and other motion constraints are very important considerations. Third, in human motion, we usually require the motion to be natural, where in robotics this is not an issue.

4.1 The Collision Avoidance Subsystem

[1]The significance of this fact is that naive joint space or "configuration space" planning methods are exponential in the number of DOFs!

[2]Since the robot is either bolted to a stable floor where it can provide a suitably large reaction force to counterbalance any manipulable load, or else is mobile on a stable multi-legged or wheeled base.

Fig. 5 depicts the functionality of our reactive behavior approach. First, *Jack* produces incremental motions using the posture transitions to invoke human behavior functions interpreted by the motion system. (To distinguish intermediate figure configurations from the postures in the transition matrix, we will call them *poses*.) Each pose is sent to the collision avoidance module which checks for self-collisions and collisions with environment obstacles. If there is no collision, the pose is sent on for display. If there is a collision, the collision avoidance system will resolve it and produce a new pose. This new pose, even though collision-free, may not satisfy human motion constraints (keeping balance, etc.), so it will be sent back to the behavior system for processing. This procedure iterates until the final pose is collision-free and satisfies human motion constraints. Then the pose can be sent on for display.

As *Jack*'s posture transitions are goal-directed rather than path-directed, any changes in pose still allow *Jack* to find its way to the goal posture. For this reason, it is very natural to implement collision avoidance as one of *Jack*'s reactive behaviors. This makes the integration of collision avoidance and other human behaviors extremely simple, an impossibility in a key-frame based system.

4.2 Determining Collisions using Potential Fields

Each obstacle is modeled as an artificial potential field. The potential energy will be maximum at the obstacle center and will decrease to zero on the surface or outside the obstacle. When there is no collision, the human body will have zero potential energy and no repulsive force acting on it. When collisions occur, some parts of the body are inside the obstacle (and thus inside its potential field), which in turn will produce a repulsive force to push them away. In order to have good performance, we use a kinematics approach instead of dynamics. Instead of simulating the effects of the repulsive force on the human body, we use an optimization procedure to minimize potential energy on the human body. Given a pose, the optimization routine finds a pose with minimal potential energy and which is similar to the original pose. This minimized potential energy pose is collision-free providing that the total energy is zero.

Fig. 6 gives an example potential function for a cylindrical obstacle. The potential field of the obstacle generates a repulsive potential force to push any end-effector out of itself. For any given point $\mathbf{r}$ in space, the potential energy generated by the cylindrical obstacle at $\mathbf{r}$ can be defined as follows:

$$P(\mathbf{r}) = \begin{cases} 0 & \text{if } \|\mathbf{r}-\mathbf{r0}\| >= \mathbf{D} \\ (\mathbf{D})^2 - (\mathbf{r}-\mathbf{r0})\cdot(\mathbf{r}-\mathbf{r0}) & \text{otherwise;} \end{cases} \quad (1)$$

where $\mathbf{r0}$ is the projection of $\mathbf{r}$ on the center line of the cylinder, and $\mathbf{D}$ is the length of the vector from $\mathbf{r0}$ to the boundary of the cylinder which goes through $\mathbf{r}$. The gradient of this field is needed by the optimization

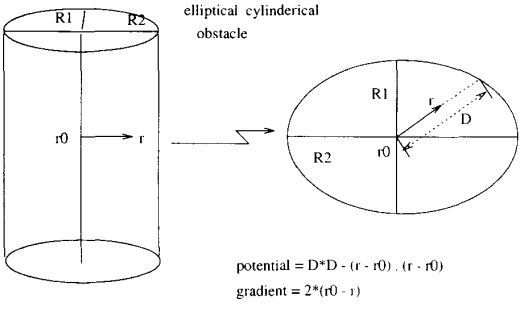

Figure 6: An example potential function for cylindrical obstacles

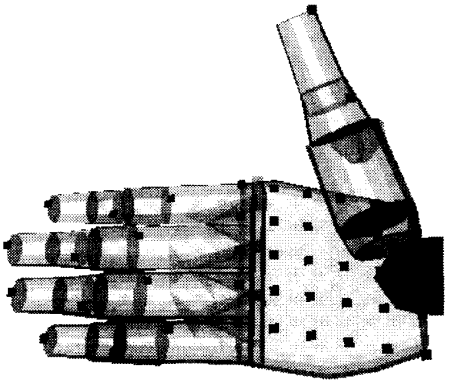

Figure 7: Sensors on the hand

procedure [2]:

$$\nabla_{\mathbf{r}} P(\mathbf{r}) = \begin{cases} 0 & \text{if } \|\mathbf{r} - \mathbf{r0}\| >= \mathbf{D} \\ -2(\mathbf{r} - \mathbf{r0}) & \text{otherwise.} \end{cases} \quad (2)$$

Note that if **r0** lies beyond the center line segment of the cylinder (i.e., past the ends), both the potential and gradient are 0.

4.3 Using Sensors to Simplify the Evaluation of Potential Functions

To prevent the hand from colliding with an obstacle, all points on the hand must stay outside of the obstacle. Thus the function which evaluates the *handpotential* should have the following properties:

$$handpotential = \begin{cases} 0; & \text{if outside the obstacle} \\ > 0; & \text{if inside the obstacle.} \end{cases}$$

Potential functions, however, are defined on points in space. Since hands are complex objects (in one version of *Jack*'s human model each hand has 16 polyhedra, 3 for each finger and 1 for the palm), and the hand-shape can change, there would seem to be no easy way to compute *handpotential* from obstacle potential functions.

Our solution is to place "sensors" on the hand (Fig. 7). The potential from an obstacle detected by a sensor represents the potential felt by the area on the hand surrounding the sensor.

Let p_i $(1 \leq i \leq n)$ be the position vector of the i^{th}-sensor on the hand, and the potential detected by the i^{th}-sensor be $potential(p_i)$. Then

$$handpotential = max\ (potential(p_i)).$$

In the case of hand-hand collisions, the potential is calculated using the minimum distance between any two sensors on different hands.

Similarly, we also put sensors on the arms and other parts of the body and evaluate the potential in the same way. Since we model body part potential fields as elliptical cylinders, it is possible to calculate arm potential directly without using sensors, i.e. we can use the minimum distance between two elliptical cylinders to compute the potential. Although analytic computation of the distance between two bounding volumes can extend the set of geometric shapes considered for collision avoidance, arbitrarily shaped objects may not offer such a convenient route. The sensor-based method, however, is quite suitable for general objects.

4.4 Escaping Local Minima

Even though it is much more efficient compared to other methods, the potential field approach suffers from the local minima problem: the agent may be trapped in a local minimum pose, although there is a pose that has even lower potential energy.

We provide two ways to handle the local minima problem. A simple but powerful way is the random-walk approach [19] which perturbs the figure slightly when it is trapped in a local minima pose. If the sizes and numbers of local minimum wells are not too great, chances are good that the agent can escape from them efficiently.

If the space is less well-behaved, a systematic search may be necessary. We use an incremental search method that builds the configuration-space (C-space) incrementally during searching instead of in advance. Pre-computing a C-space is impractical for the number of DOFs in our model. For example, for a C-space of only 4 DOFs with each dimension discretized into 20 regions, the size of the space is $20^4 = 16 \times 10^4$ and so requires 16×10^4 collision detections to create. To detect so many collisions in a complex human environment is too costly (if each detection takes 10ms, the total collision detection time would be over 26 minutes). Besides, many of the detections may not be used in the motion plan at all. So incremental search will work best under these circumstances.

When the agent is trapped in a local minima position and a random-walk has failed to free him, a search begins in the vicinity of the local minima pose

of the discretized C-space. Each time we start with the most promising neighbors in the C-space (by evaluating each posture's *promising-index*, see below). Instead of building the C-space in advance, each time we search for the next posture, collision detection is executed to determine if it is collision-free. In most cases, the incremental method only searches a small fraction of the C-space. In the example given below, only 35 out of 1000 C-space cells are searched to find a collision-free path.

This method gives good performance when the number of DOFs is small (e.g., 5 or less) and the resolution is rather coarse. But this is possible with posture transitions because each posture transition involves a few DOFs. The incremental search method can help the agent escape from local minima postures easily.

This incremental search method can also produce more realistic human motions by controlling the search direction. The *promising-index* of a pose represents the likelihood that this pose will lead the agent to the goal posture collision-free along a natural motion path. Factors which can be taken into consideration to evaluate the promising-index are: the pose comfort level, the distance to the goal posture, the freedom of movement in its neighborhood, etc. All these factors are added using a weighted sum (where the weight given to each factor depends on the application):

$$\begin{aligned} \textit{promising-index} = {} & f1 * \textit{comfort-evel} + \\ & f2 * \textit{distance-to-goal} + \\ & f3 * \textit{free-space-factor}, \end{aligned}$$

$$\textit{where} f1 + f2 + f3 = 1.0.$$

When exploring from the current pose, all of its neighbor pose's promising-indexes are evaluated and the pose with the highest value is chosen as the next candidate posture. Note that if a given pose involves collision, its promising-index will be 0 and it will never be chosen.

The comfort level of each posture is computed by its balance factor (well balanced or not), stress factor at all joints, maximum torque required if the motion is executed (from current posture to the evaluated neighbor posture), preference to some groups of joints during a particular motion, etc. The free space factor is the ratio of number of free (non-obstacle) neighbors to the total number of neighbors of the evaluated pose.

5 Examples

The following example (Fig. 8) shows the incremental search results involving 3 DOFs – two at the shoulder and one at the elbow. The task is to compute a collision-free reach motion of the hand from above the table to below the table. In this example, we always prefer to move the shoulder joint by setting its preference factor higher than that of the elbow joint (in this example, only preference factor and distance to goal posture are used to evaluate the promising-index). Fig. 8 shows the resulting motion, which was computed in less than 1 second on a graphics workstation.

Figures 10 and 11 show an example of posture transition with collision avoidance. Initially the agent is sitting on a chair and is instructed to stand up. This involves two posture transitions, **`Sit to Squat`** and **`Squat to Stand`**. If there were no obstacles, he would have been able to stand up straight as shown in Fig. 9. However, since there is an obstacle over his head, he tries to stand up as straight as possible while avoiding collisions at the same time. Fig. 10 shows the posture transition without collision avoidance, and Fig. 11 shows the same transition with collision avoidance. Notice that the collision avoidance behavior and other human behaviors work together to provide a transition which is not only collision-free, but is also a natural-looking motion.

6 Conclusions

At the present time the only major weakness in the posture interpolation system is the control of timing. Manual adjustment of timing on postural transitions is under animator control through in interactive interface in *Jack*. If a collision is imminent, however, there is no effective strategy to predict the new timing of the transitional motion. We are investigating some possible strategies based on torque availability [15, 21] or Fitts' Law timing predictions [22]

We have demonstrated an integrated animation system for the postural control of a human-like agent. The system uses constraints and behaviors to guide postural changes toward goals. The intermediate postures comprising a complex postural change may be explicitly stored in a transition network. The individual postures in the network may be characterized by any constraints or behaviors available in the system. While transitioning from one posture to another, a real-time collision avoidance mechanism is invoked. This component creates additional constraints or behaviors that augment the existing ones, and so seamlessly integrates with the postural interpolation. The collision avoidance mechanism avoids self as well as environmental collisions, and is able to accommodate a number of measures to tune the naturalness of the resulting motions.

Acknowledgments

This research is partially supported by ARO DAAL03-89-C-0031 including U.S. Army Research Laboratory (Aberdeen); U.S. Air Force DEPTH through Hughes Missile Systems F33615-91-C-0001; Naval Training Systems Center N61339-93-M-0843; Sandia Labs AG-6076; DMSO through the University of Iowa; NASA KSC NAG10-0122; MOCO, Inc.; NSF CISE CDA88-22719, and Instrumentation and Laboratory Improvement Program #USE-9152503.

References

[1] N. Magnenat-Thalmann and D. Thalmann, *Computer Animation: Theory and Practice.* New York, NY: Springer-Verlag, 1985.

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

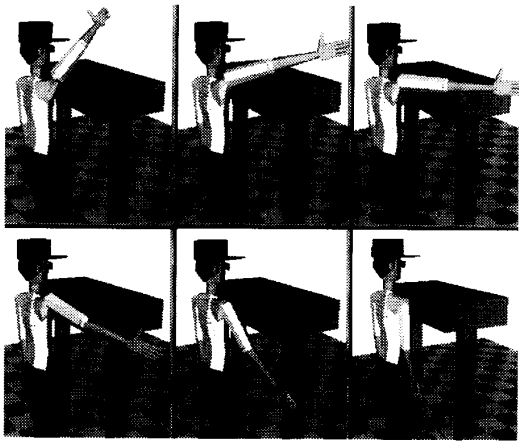

Figure 8: Arm reach motion with collision avoidance

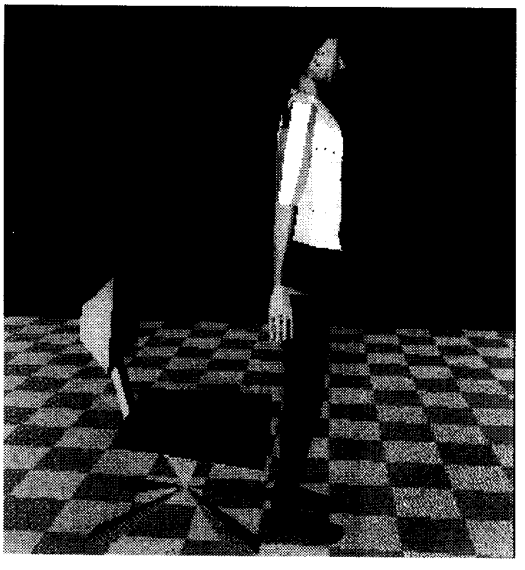

Figure 9: Stand with No Obstacle

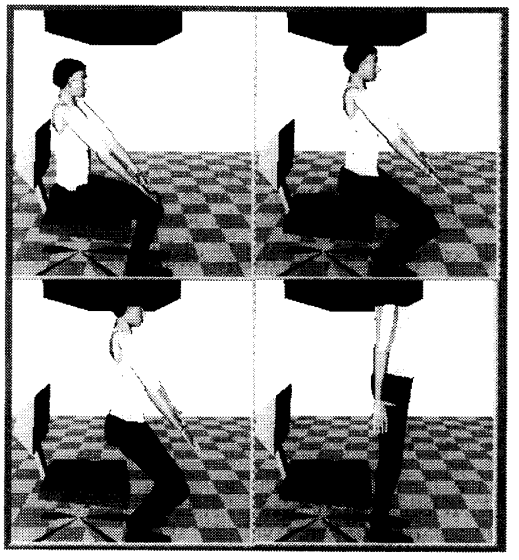

Figure 10: Posture Transition with No Collision Avoidance

Figure 11: Posture Transition with Collision Avoidance

[3] N. Magnenat-Thalmann and D. Thalmann, "Complex models for animating synthetic actors," *IEEE Computer Graphics and Applications*, vol. 11, pp. 32–44, Sept. 1991.

[4] R. Barzel and A. H. Barr, "A modeling system based on dynamic constraints," *Computer Graphics*, vol. 22, no. 4, pp. 179–188, 1988.

[5] D. Metaxas and D. Terzopoulos, "Dynamic deformation of solid primitives with constraints," *Computer Graphics*, vol. 26, pp. 309–312, July 1992.

[6] M. W. Lee and T. L. Kunii, "Animation design: A database-oriented animation design method with a video image analysis capability," in *State-of-the Art in Computer Animation* (N. Magnenat-Thalmann and D. Thalmann, eds.), pp. 97–112, New York, NY: Springer-Verlag, 1989.

[7] J. K. Hahn, "Realistic animation of rigid bodies," *Computer Graphics*, vol. 22, pp. 299–308, August 1988.

[8] A. Witkin and M. Kass, "Spacetime constraints," *Computer Graphics*, vol. 22, no. 4, pp. 159–168, 1988.

[9] D. E. Breen, "Choreographing goal-oriented motion using cost functions," in *State-of-the-art in Computer Animation* (D. T. Nadia Magnenat-Thalmann, ed.), pp. 141–151, Springer-Verlag Tokyo, 1989. Proceedings of Computer Animation '89.

[10] W. Becket and N. I. Badler, "Integrated behavioral agent architecture," March 1993.

[11] L. S. Brotman and A. N. Netravali, "Motion interpolation by optimal control," *Computer Graphics*, vol. 22, no. 4, pp. 309–315, 1988.

[12] M. H. Raibert and J. K. Hodgins, "Animation of dynamic legged locomotion," *Computer Graphics*, vol. 25, pp. 349–358, July 1991.

[13] M. van de Panne, E. Fiume, and Z. Vranesic, "Reusable motion synthesis using state-space controllers," *Computer Graphics*, vol. 24, pp. 225–234, August 1990.

[14] M. van de Panne, E. Fiume, and Z. Vranesic, "Physically based modeling and control of turning," *Graphical Models and Image Processing*, vol. 55, pp. 507–521, November 1993.

[15] P. Lee, S. Wei, J. Zhao, and N. I. Badler, "Strength guided motion," *Computer Graphics*, vol. 24, no. 4, pp. 253–262, 1990.

[16] Y. K. Hwang and N. Ahuja, "Gross motion planning - a survey," *ACM Computing Surveys*, vol. 24, pp. 219–291, September 1992.

[17] J. Zhao and N. I. Badler, "Real-time inverse kinematics with joint limits and spatial constraints," Tech. Rep. MS-CIS-89-09, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104, 1989.

[18] C. B. Phillips and N. I. Badler, "Interactive behaviors for bipedal articulated figures," *Computer Graphics*, vol. 25, no. 4, pp. 359–362, 1991.

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

[20] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[21] W. Ching and N. I. Badler, "Fast motion planning for anthropometric figures with many degrees of freedom," in *IEEE Intl. Conf. on Robotics and Automation*, May 1992.

[22] J. Esakov, N. I. Badler, and M. Jung, "An investigation of language input and performance timing for task animation," in *Graphics Interface '89*, (San Mateo, CA), pp. 86–93, Morgan-Kaufmann, June 1989.