done separately, then combined for the final posture.

A participation vector is derived from the spine's current position, target position, and maximum position. This global participation represents a 3D vector of the ratio of spine movement to the maximum range of movement. Participation is used to calculate the joint weights.

The following formulas are defined in each of three DOFs. Let

$Target$ = spine target position
$Current$ = spine current position
Max = spine sum of joint limits
$Rest$ = spine sum of joint rest positions.

If the spine is bending, then the participation P is

$$P = \frac{Target - Current}{Max - Current}.$$

Otherwise, the spine is unbending and

$$P = \frac{Target - Current}{Rest - Current}.$$

The joint positions of the entire spine must sum up to the target position. To determine how much the joint participates, a set of weights is calculated for each joint. The participation weight is a function of the joint number, the initiator joint, and the global participation derived above. Also, a resistance weight is based on the resistor joint, degree of resistance, and global participation. To calculate the weight for each joint i, let:

j_i = joint position
$limit_i$ = the joint limit
$rest_i$ = the rest position
p_i = participation weight
r_i = resistance weight.

If the spine is bending, then

$$w_i = p_i \cdot r_i \cdot (limit_i - j_i),$$

while if the spine is unbending,

$$w_i = p_i \cdot r_i \cdot (rest_i - j_i).$$

The weights range from 0 to 1. A weight of $k\%$ means that the movement will go $k\%$ of the differential between the current position and either the joint limit (for bending) or the joint rest position (for unbending).

To understand resistance, divide the spine into two regions split at the resistor joint. The region of higher activity contains the initiator. Label these regions *active* and *resistive*. The effect of resistance is that joints in the resistive region will resist participating in the movement specified by the parameter degree of resistance. Also, joints inbetween the initiator and resistor will have less activity depending on the degree of resistance.

Resistance does not freeze any of the joints. Even at 100% resistance, the active region will move until all joints reach their joint limits. Then, if there is no other way to satisfy the target position, the resistive region will begin to participate.

If the desired movement is from the current position to one of two maximally bent positions, then the weights calculated should be 1.0 for each joint participating. The algorithm interpolates correctly to either maximally bent position. It also interpolates correctly to the position of highest comfort. To calculate the position of each joint i after movement succeeds, let:

j_i = joint position
j_i^* = new joint position
$Target$ = spine target position
$Current$ = spine current position
$M = Target - Current$ = incremental movement of the spine.

Then

$$j_i^* = j_i + \frac{M w_i}{\sum w_i},$$

and it is easy to show that $\sum j_i^* = Target$:

$$\begin{aligned}
\sum j_i^* &= \sum (j_i + \tfrac{M w_i}{\sum w_i}) \\
&= \sum j_i + \sum \tfrac{M w_i}{\sum w_i} \\
&= Current + M \tfrac{\sum w_i}{\sum w_i} \\
&= Current + M \\
&= Target.
\end{aligned}$$

The bend torso command positions the torso using forward kinematics, without relying on a dragging mechanism. It consists of potentiometers which control the total bending angle along the three DOFs. The command also

prompts for the flavor of bending. These controls are the same as for the set torso behavior command described above. They include options which specify the range of motion of the spine, defined through a top and bottom joint, along with *initiator* and *resistor* joints which control the weighting between the vertebrae.

Bending the torso tends to cause large movements of the center of mass, so this process has a great effect on the posture of the figure in general, particularly the legs. For example, if the figure bends forward, the hips automatically shift backwards so that the figure remains balanced. This is illustrated in Figure 4.7.

4.2.4 The Pelvis

The rotate pelvis command changes the global orientation of the hips. This can curl the hips forwards or backwards, tilt them laterally, or twist the entire body around the vertical axis. The manipulation of the pelvis also activates the torso behavior in a pleasing way. Because of its central location, manipulations of the pelvis provide a powerful control over the general posture of a figure, especially when combined with the balance and *keep vertical* torso constraints. If the torso is kept vertical while the pelvis curls underneath it, then the torso curls to compensate for the pelvis. This is shown in Figure 4.8.

The rotate pelvis command can also trigger the active stepping behavior if the orientation reaches an extreme angle relative to the feet.

4.2.5 The Head and Eyes

The move head and move eyes commands manipulate the head and eyes, respectively, by allowing the user to interactively move a fixation point. The head and eyes both automatically adjust to aim toward the reference point. The head and eyes rotate as described in Section 4.1.1.

4.2.6 The Arms

The active manipulation of the arm allows the user to drag the arm around in space using the mechanism described in Section 3.2.5. These movements utilize the shoulder complex as described in Section 2.4 so that the coupled joints have a total of three DOFs. Figure 4.10 shows the left hand being moved forwards.

Although it seems natural to drag this limb around from the palm or fingertips, in practice this tends to yield too much movement in the wrist and the wrist frequently gets kinked. The twisting scheme helps, but the movements to get the wrist straightened out can interfere with an acceptable position for the arm. It is much more effective to do the positioning in two steps, the first positioning the arm with the wrist fixed, and the second rotating the hand into place. Therefore, our active manipulation command for the arms can control the arm either from a reference point in the palm or from the lower

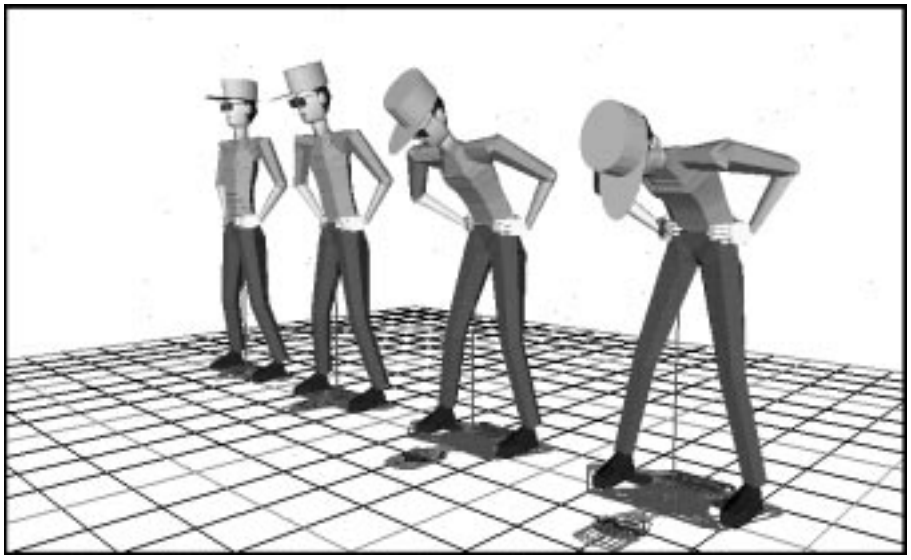

Figure 4.7: Bending the Torso while Maintaining Balance.

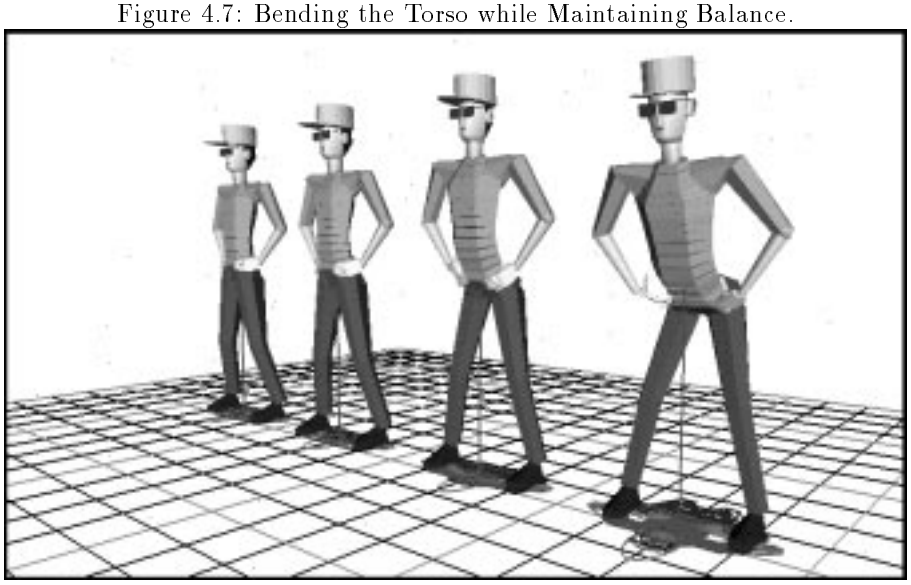

Figure 4.8: Rotating the Pelvis while Keeping the Torso Vertical.

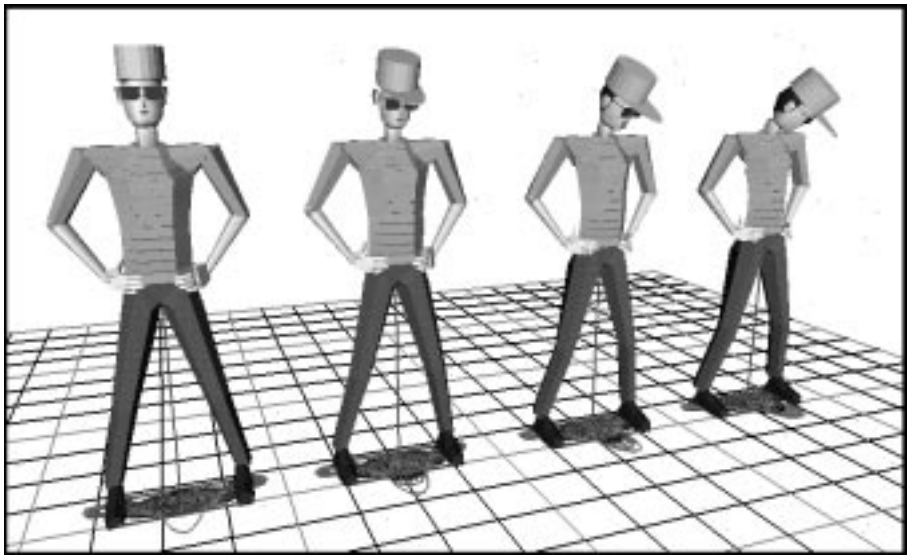

Figure 4.9: Moving the Head.

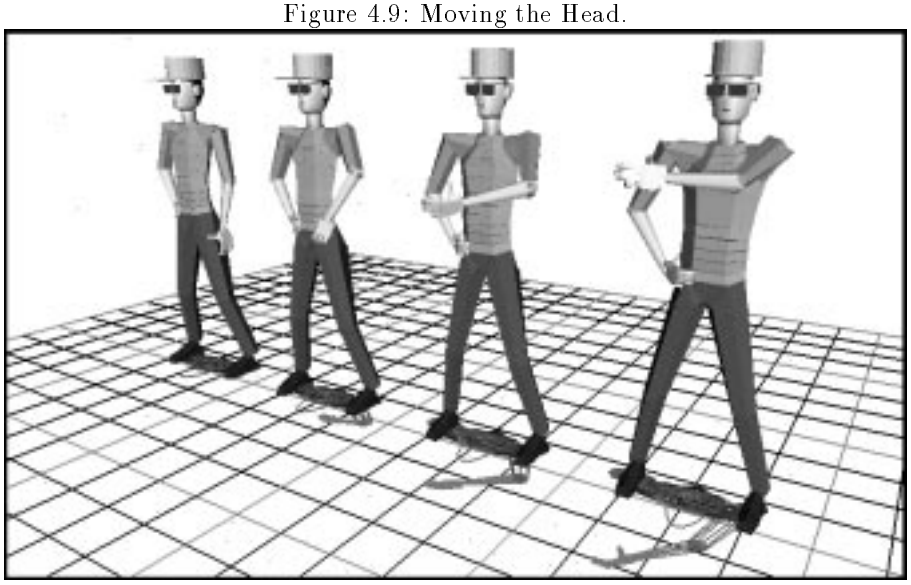

Figure 4.10: Moving the Hand.

end of the lower arm, just above the wrist. This process may loosely simulate how humans reach for objects, for there is evidence that reaching involves two overlapping phases, the first a ballistic movement of the arm towards the required position, and the second a correcting stage in which the orientation of the hand is fine-tuned [Ros91]. If the target for the hand is an actual grasp, then a specialized *Jack* behavior for grasping may be invoked which effectively combines these two steps.

4.2.7 The Hands and Grasping

Jack contains a fully articulated hand. A hand grasp capability makes some reaching tasks easier [RG91]. The grasp action requires a target object and a grasp type. The *Jack* grasp is purely kinematic. It is a considerable convenience for the user, however, since it virtually obviates the need to individually control the 20 DOFs in each hand.

For a grasp, the user specifies the target object and a grip type. The user chooses between a predefined grasp site on the target or a calculated transform to determine the grasp location. A distance offset is added to the site to correctly position the palm center for the selected grip type. The hand is preshaped to the correct starting pose for the grip type selected, then the palm moves to the target site.

The five grip types implemented are the power, precision, disc, small disc, and tripod [Ibe87]. The grips differ in how the hand is readied and where it is placed on or near the object. Once these actions are performed, the fingers and thumb are just closed around the object, using collision detection on the bounding box volume of each digit segment to determine when to cease motion.

4.3 The Animation Interface

[2]The *Jack* animation system is built around the concept of a *motion*, which is a change in a part of a figure over a specific interval of time. A motion is a rather primitive notion. Typically, a complex animation consists of many distinct motions, and several will overlap at each point in time. Motions are created interactively through the commands on the *motion menu* and the *human motion menu*. There are commands for creating motions which control the placement of the feet, center of mass, hands, torso, arms, and head.

Jack displays motions in an animation window. This window shows time on a horizontal axis, with a description of the parts of each figure which are moving arranged vertically. The time interval over which each motion is active is shown as a segment of the time line. Each part of the body gets a different track. The description shows both the name of the figure and the name of the body part which is moving. The time line itself displays motion attributes graphically, such as velocity control and relative motion weights.

[2]Paul Diefenbach.

The numbers along the bottom of the animation grid are the time line. By default, the units of time are in seconds. When the animation window first appears, it has a width of 3 seconds. This can be changed with the arrows below the time line. The horizontal arrows scroll through time keeping the width of the window constant. The vertical arrows expand or shrink the width of the window, in time units. The current animation time can be set either by pressing the middle mouse button in the animation window at the desired time and scrolling the time by moving the mouse or by entering the current time directly through the goto time.

Motions actually consist of three distinct phases, although this is hidden from the user. The first stage of a motion is the pre-action step. This step occurs at the starting time of the motion and prepares the figure for the impending motion. The next stage is the actual motion function itself, which occurs at every time interval after the initial time up to the ending time, inclusive. At the ending time after the last incremental motion step, the post-action is activated disassociating the figure from the motion. Because of the concurrent nature of the motions and the possibility of several motions affecting the behavior of one moving part, these three stages must occur at each time interval in the following order: motion, post-action, pre-action. This allows all ending motions to finish before initializing any new motions affecting the same moving part.

While the above description implies that body part motions are controlled directly, this is not the true behavior of the system. The animation system describes postures through constraints, and the motions actually control the existence and parameters of the constraints and behaviors which define the postures. Each motion has a set of parameters associated with it which control the behavior of the motion. These parameters are set upon creation of the motion and can be modified by pressing the right mouse button in the animation window while being positioned over the desired motion. This changes or deletes the motion, or turns the motion on or off.

Each motion is active over a specific interval in time, delimited by a *starting time* and an *ending time*. Each motion creation command prompts for values for each of these parameters. They may be entered numerically from the keyboard or by direct selection in the animation window. Existing time intervals can be changed analogously. Delimiting times appear as vertical "ticks" in the animation window connected by a velocity line. Selecting the duration line enables time shifting of the entire motion.

The yellow line drawn with each motion in the animation window illustrates the motion's *weight function*. Each motion describes movement of a part of the body through a kinematic constraint. The constraint is only active when the current time is between the motion's starting time and ending time. It is entirely possible to have two motions which affect the same part of the body be active at the same time. The posture which the figure assumes is a weighted average of the postures described by the individual motions. The weights of each constraint are described through the weight functions, which can be of several types:

`constant` The weight does not change over the life of the constraint.

`increase` The weight starts out at 0 and increases to is maximum at the end time.

`decrease` The weight starts out at its maximum and decreases to 0 at the end time.

`ease in/ease out` The weight starts at 0, increases to its maximum halfway through the life of the motion, and then decreases to 0 again at the end time.

The shape of the yellow line in the animation window illustrates the weight function. The units of the weight are not important. The line may be thought of as an icon describing the weight function.

The green line drawn with each motion in the animation window represents the velocity of the movement. The starting point for the motion comes from the current posture of the figure when the motion begins. The ending position of the motion is defined as a parameter of the motion and is specified when the motion is created. The speed of the end effector along the path between the starting and ending positions is controlled through the velocity function:

`constant` Constant velocity over the life of the motion.

`increase` The velocity starts out slow and increases over the life of the motion.

`decrease` The velocity starts out fast and decreases over the life of the motion.

`ease in/ease out` The velocity starts slow, increases to its maximum halfway through the life of the motion, and then decreases to 0 again at the end time.

The shape of the green line in the animation window illustrates the velocity function. The scale of the velocity is not important. This line can be thought of as an icon describing the velocity.

4.4 Human Figure Motions

The commands on the *human motion menu* create timed body motions. These motions may be combined to generate complex animation sequences. Taken individually, each motion is rather uninteresting. The interplay between the motions must be considered when describing a complex movement. These motions are also mostly subject to the behavioral constraints previously described.

Each one of these commands operates on a human figure. If there is only one human figure present, these commands automatically know to use that figure. If there is more than one human figure, each command will begin

by requiring the selection of the figure. Each of these commands needs the starting and ending time of the motion. Default or explicitly entered values may be used. The motion may be repositioned in the animation window using the mouse.

A motion is a movement of a part of the body from one place to another. The movement is specified in terms of the final position and the parameters of how to get there. The *initial* position of the motion, however, is defined implicitly in terms of where the part of the body is when the motion starts. For example, a sequence of movements for the feet are defined with one motion for each foot fall. Each motion serves to move the foot from its current position, wherever that may be, when the motion starts, to the final position for that motion.

4.4.1 Controlling Behaviors Over Time

We have already seen how the posture behavior commands control the effect of the human movement commands. Their effect is permanent, in the sense that behavior commands and constraints hold continuously over the course of an animation. The "timed" behavior commands on the *human behavior menu* allow specifying controls over specific intervals of time. These commands, create timed figure support, create timed balance control, create timed torso control, create time hand control, and create time head control each allow a specific interval of time as described in Section 4.3 just like the other motion commands. The behavior takes effect at the starting time and ends with the ending time. At the ending time, the behavior parameter reverts to the value it had before the motion started.

4.4.2 The Center of Mass

A movement of the center of mass can be created with the create center of mass motion command. This controls the balance point of the figure. There are two ways to position the center of mass. The first option positions the balance point relative to the feet by requiring a floating point number between 0.0 and 1.0 which describes the balance point as an interpolation between the left (0.0) and right (1.0) foot; thus 0.3 means a point $\frac{3}{10}$ of the way from the left foot to the right. Alternatively, one can specify that the figure is standing with 30% of its weight on the right foot and 70% on the left.

The global location option causes the center of mass to move to a specific point in space. Here *Jack* will allow the user to move the center of mass to its desired location using the same technique as with the move center of mass command on the *human manipulation menu*.

After choosing the positioning type and entering the appropriate parameters, several other parameters may be provided, including the weight function and velocity. The weight of the motion is the maximum weight of the constraint which controls the motion, subject to the weight function.

The behavior of the create center of mass motion command depends on the setting of the figure support. It is best to support the figure through the foot which is closest to the center of mass, which is the foot bearing most of the weight. This ensures that the supporting foot moves very little while the weight is on it.

The effect of the center of mass motion depends upon both the setting of the figure support at the time the motion occurs and when the motion is created. For predictable behavior, the two should be the same. For example, if a motion of the center of mass is to take place with the figure seated, then the figure should be seated when the motion is created.

The support of the figure can be changed at a specific moment with the create timed figure support command. This command requires starting and ending times and the figure support, just like the set figure support command. When the motion's ending time is reached, the support reverts to its previous value.

4.4.3 The Pelvis

The lower torso region of the body is controlled in two ways: through the center of mass and through the pelvis. The center of mass describes the location of the body. The pelvis constraint describes the orientation of the hips. The hips can rotate over time with the command create pelvis motion.

The create pelvis motion command allows the user to rotate the pelvis into the final position, using the same technique as the rotate pelvis command. It also requires the velocity, and weight functions, and the overall weight.

4.4.4 The Torso

The movement of the torso of a figure may be specified with the create torso motion. This command permits bending the torso into the desired posture, using the same technique as the move torso command. Like the move torso command, it also prompts for the torso parameters.

The create torso motion command requires a velocity function, but not a weight or a weight function because this command does not use a constraint to do the positioning. Because of this, it is not allowable to have overlapping torso motions.

After the termination of a torso motion, the vertical torso behavior is turned off. The behavior of the torso can be changed at a specific moment with the create timed torso control command. This command requires starting time and ending times and the type of control, just like the set torso control command. When the motion's ending time is reached, the behavior reverts to its previous value.

4.4.5 The Feet

The figure's feet are controlled through the pair of commands create foot motion and create heel motion. These two commands can be used in conjunction to

cause the figure to take steps. The feet are controlled through constraints on the heels and on the toes. The toe constraints control the position and orientation of the toes. The heel constraint controls only the height of the heel from the floor. The position of the heel, and the entire foot, comes from the toes. The commands allow the selection of the right or left foot.

The create foot motion command gets the ending position for the foot by the technique of the move foot command. In addition, a height may be specified. The motion causes the foot to move from its initial position to its final position through an arc of a certain elevation. A height of 0 implies that the foot moves in straight-line path. If both the initial and final positions are on the floor, then this means the foot will slide along the floor. A height of 10cm means the toes will reach a maximum height from the floor of 10cm halfway through the motion.

The effect of the create foot motion command depends upon how the figure is supported. Interactively, the move foot command automatically sets the support of the figure to the moving foot, and the create foot motion command does the same. However, this does *not* happen during the generation of the movement sequence. The behavior of the feet depends very much on the support of the figure, although the effect is quite subtle and difficult to define. A foot motion can move either the supported or non-supported foot, but it is much better at moving the non-supported one.

The general rule of thumb for figure support during a movement sequence is the opposite of that for interactive manipulation: during a movement sequence, it is best to have the support through the foot on which the figure has most of its weight. This will ensure that this foot remains firmly planted.

The behavior of the feet can be changed at a specific moment with the create timed foot control command. This command needs starting and ending times and the type of control, just like the set foot control command. When the motion's ending time is reached, the behavior reverts to its previous value.

4.4.6 Moving the Heels

The movement of the foot originates through the toes, but usually a stepping motion begins with the heel coming off the floor. This may be specified with the create heel motion command. This command does not ask for a location; it only asks for a height. A height of 0 means on the floor.

Usually a stepping sequence involves several overlapping motions. It begins with a heel motion to bring the heel off the floor, and at the same time a center of mass motion to shift the weight to the other foot. Then a foot motion causes the foot to move to a new location. When the foot is close to its new location, a second heel motion causes the heel to be planted on the floor and a second center of mass motion shifts some of the weight back to this foot.

4.4.7 The Arms

The arms may be controlled through the command create arm motion. This command moves the arms to a point in space or to a reference point such as a site. The arm motion may involve only the joints of the arm or it may involve bending from the waist as well. The command requires the selection of the right or left arm and whether the arm movement is to be confined to the arm or include a bending of the torso. Arm movements involving the torso should *not* be combined with a torso movement generated with the create torso motion command. Both of these control the torso in conflicting ways.

The hand is then moved to the new position in space, using the same technique as the move arm command. The user can specify if this position is relative to a segment; that is, to a global coordinate location or to a location relative to another object. If the location is relative, the hand will move to that object even if the object is moving as the hand moves during the movement generation.

4.4.8 The Hands

Hand behavior may also be specified over time with the create timed hand control command. The hand can be temporarily attached to certain objects over certain intervals of time. This command requires starting and ending times and the type of control, just like the set torso control command.

Objects can be attached *to the hands* over an interval of time with the create timed attachment command. The timing of the grasp action can be set accordingly. During animation, one can specify the hand grasp site, the approach direction, the starting hand pose, and the sequencing of finger motions culminating in the proper grasp. If one is willing to wait a bit, the hand pose will even be compliant, via collision detection, to changes in the geometry of the grasped object as it or the hand is moved.

4.5 Virtual Human Control

[3]We can track, in real-time, the position and posture of a human body, using a minimal number of 6 DOF sensors to capture full body standing postures. We use four sensors to create a good approximation of a human operator's position and posture, and map it on to the articulated figure model. Such real motion inputs can be used for a variety of purposes.

- If motion data can be input fast enough, live performances can be animated. Several other virtual human figures in an environment can react and move in real-time to the motions of the operator-controlled human figure.

[3]Michael Hollick, John Granieri

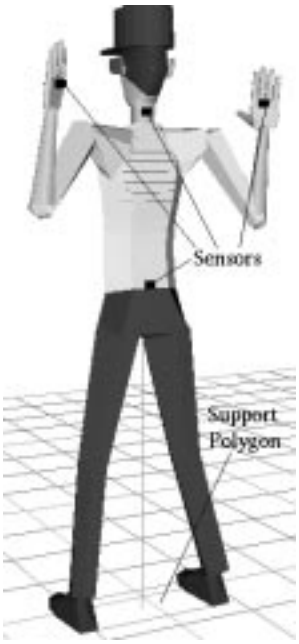

Figure 4.11: Sensor Placement and Support Polygon.

- Motion can be recorded and played back for analysis in different environments. The spatial locations and motions of various body parts can be mapped onto different-sized human figures; for example, a 5^{th} percentile operator's motion can be mapped onto a 95^{th} percentile figure.

- Virtual inputs can be used for direct manipulation in an environment, using the human figure's own body segments; for example, the hands can grasp and push objects.

We use constraints and behavior functions to map operator body locations from external sensor values into human postures.

We are using the **Flock of Birds** from Ascension Technology, Inc. to track four points of interest on the operator. Sensors are affixed to the operator's palms, waist, and base of neck by elastic straps fastened with velcro (Fig. 4.11). Each sensor outputs its 3D location and orientation in space. With an Extended Range Transmitter the operator can move about in an 8-10 foot hemisphere. Each bird sensor is connected to a Silicon Graphics 310VGX via a direct RS232 connection running at 38,400 baud.

One of the initial problems with this system was slowdown of the simulation due to the sensors. The Silicon Graphics operating system introduces a substantial delay between when data arrives at a port and when it can be accessed. This problem was solved by delegating control of the Flock to a separate server process. This server will configure the Flock to suit a client's

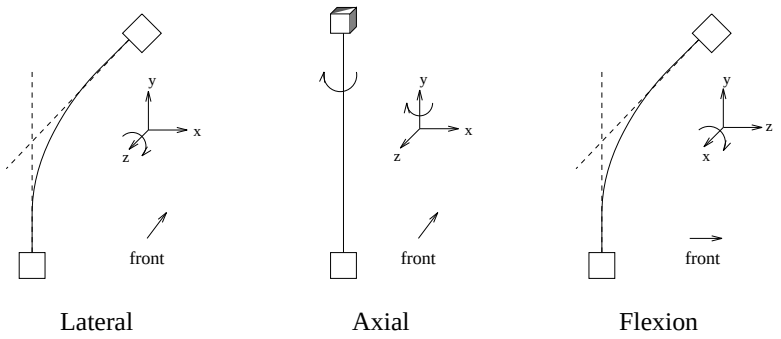

Figure 4.12: Extracting the Spine Target Vector

needs, then provide the client with updates when requested. The server takes updates from the Birds at the maximum possible rate, and responds to client requests by sending the most recent update from the appropriate Bird. This implementation allows access to the Flock from any machine on the local network and allows the client to run with minimal performance degradation due to the overhead of managing the sensors. The sensors produce about 50 updates per second, of which only about 8 to 10 are currently used due to the effective frame rate with a shaded environment of about 2000 polygons. The bulk of the computation lies in the inverse kinematics routines.

The system must first be calibrated to account for the operator's size. This can be done in two ways – the sensor data can be offset to match the model's size, or the model can be scaled to match the operator. Either approach may be taken, depending on the requirements of the particular situation being simulated.

Each frame of the simulation requires the following steps:

1. The pelvis segment is moved as the first step of the simulation. The absolute position/orientation of this segment is given by the waist sensor after adding the appropriate offsets. The figure is rooted through the pelvis, so this sensor determines the overall location of the figure.

2. The spine is now adjusted, using the location of the waist sensor and pelvis as its base. The spine initiator joint, resistor joint, and resistance parameters are fixed, and the spine target position is extracted from the relationship between the waist and neck sensors. The waist sensor gives the absolute position of the pelvis and base of the spine, while the rest of the upper torso is placed algorithmically by the model.

   The spine target position is a 3 vector that can be thought of as the sum of the three types of bending the spine undergoes – flexion, axial, and lateral. Since the sensors approximate the position/orientation of the base and top of the spine, we can extract this information directly. Lateral bending is found from the difference in orientation along the z axis, axial twisting is found from the difference in y orientation, and

flexion is determined from the difference in x orientation (Fig. 4.12). Note that the "front" vectors in this figure indicate the front of the human. This information is composed into the spine target vector and sent directly to the model to simulate the approximate bending of the operator's spine.

3. Now that the torso has been positioned, the arms can be set. Each arm of the figure is controlled by a sensor placed on the operator's palm. This sensor is used directly as the goal of a position and orientation constraint. The end effector of this constraint is a site on the palm that matches the placement of the sensor, and the joint chain involved is the wrist, elbow, and shoulder joint.

4. The figure's upper body is now completely postured (except for the head), so the center of mass can be computed. The active stepping behaviors are used to compute new foot locations that will balance the figure. Leg motions are then executed to place the feet in these new locations.

One unique aspect of this system is the absolute measurement of 3D cartesian space coordinates and orientations of body points of interest, rather than joint angles. Thus, while the model's posture may not precisely match the operator's, the end effectors of the constraints are always correct. This is very important in situations where the operator is controlling a human model of different size in a simulated environment.

With a fifth sensor placed on the forehead, gaze direction can be approximated. Hand gestures could be sensed with readily available hand pose sensing gloves. These inputs would directly control nearly the full range of *Jack* behaviors. The result is a virtual human controlled by a minimally encumbered operator.

Chapter 5

Simulation with Societies of Behaviors

[1]Recent research in autonomous robot construction and in computer graphics animation has found that a control architecture with networks of functional behaviors is far more successful for accomplishing real-world tasks than traditional methods. The high-level control and often the behaviors themselves are motivated by the animal sciences, where the individual behaviors have the following properties:

- they are grounded in perception.
- they normally participate in directing an agent's effectors.
- they may attempt to activate or deactivate one-another.
- each behavior by itself performs some task useful to the agent.

In both robotics and animation there is a desire to control agents in environments, though in graphics both are simulated, and in both cases the move to the animal sciences is out of discontent with traditional methods. Computer animation researchers are discontent with direct kinematic control and are increasingly willing to sacrifice complete control for realism. Robotics researchers are reacting against the traditional symbolic reasoning approaches to control such as automatic planning or expert systems. Symbolic reasoning approaches are brittle and incapable of adapting to unexpected situations (both advantageous and disastrous). The approach taken is, more or less, to tightly couple sensors and effectors and to rely on what Brooks [Bro90] calls *emergent behavior*, where independent behaviors interact to achieve a more complicated behavior. From autonomous robot research this approach has been proposed under a variety of names including: *subsumption architecture* by [Bro86], *reactive planning* by [GL90, Kae90], *situated activity* by [AC87],

[1]Welton Becket.

and others. Of particular interest to us, however, are those motivated explicitly by animal behavior: *new AI* by Brooks [Bro90], *emergent reflexive behavior* by Anderson and Donath [AD90], and *computational neuro-ethology* by Beer, Chiel, and Sterling [BCS90]. The motivating observation behind all of these is that even very simple animals with far less computational power than a calculator can solve real world problems in path planning, motion control, and survivalist goal attainment, whereas a mobile robot equipped with sonar sensors, laser-range finders, and a radio-Ethernet connection to a Prolog-based hierarchical planner on a supercomputer is helpless when faced with the unexpected. The excitement surrounding the success of incorporating animal-based control systems is almost revolutionary in tone and has led some proponents such as Brooks [Bro90] to claim that symbolic methods are fundamentally flawed and should be dismantled.

Our feeling, supported by Maes [Mae90], is that neural-level coupling of sensors to effectors partitioned into functional groupings is essential for the lowest *levels of competence* (to use Brooks' term), though by itself this purely reflexive behavior will not be able to capture the long-term planning and prediction behavior exhibited by humans and mammals in general. Association learning through classical conditioning can be implemented, perhaps through a connectionist approach [BW90], though this leads only to passive statistics gathering and no explicit prediction of future events.

Our feeling is that symbolic reasoning is not flawed, it is just not efficient for controlling real-valued, imprecise tasks directly. The problem with traditional planning is its insistence on constructing complete, detailed plans before executing. Recent research in this area has focused directly on relaxing this constraint by interleaving planning and executing, reusing pieces of plans, delaying planning until absolutely necessary, and dealing directly with uncertainty. The distinction between the symbol manipulation paradigm and the emergent computation paradigm is even blurring—Maes has shown how a traditional means-ends-analysis planner can be embedded in an emergent computation framework, and Shastri [Sha88] has shown how simple symbol representation and manipulation can be accomplished in neural networks (which can be seen as the most fine-grained form of neuro-physiologically consistent emergent computation).

Our strategy for agent construction will be to recognize that some form of symbolic reasoning is at the top motivational level and biologically-based feedback mechanisms are at the bottom effector level. By putting them in the same programming environment we hope to gain insight into how these extremes connect. Hopefully, the result will be more robust than the harsh, rigid, feedback-devoid distinction between the planner and its directly implemented plan primitives. As will be discussed in Section 5.1.7, however, an important technique for understanding what is missing will be to make premature leaps from high-level plans to low-level behaviors appropriate for simple creatures. This approach is bidirectional and opportunistic. Blind top-down development may never reach the real world and pure bottom-up development faces the horror of an infinite search space with no search heuristic and no

clear goals.

In this Chapter we first pursue this notion of societies of behaviors that create a forward (reactive) simulation of human activity. The remaining Sections present some of the particular behaviors that appear to be crucial for natural tasks, including locomotion along arbitrary planar paths, strength guided motion, collision-free path planning, and qualitative posture planning.

5.1 Forward Simulation with Behaviors

Figure 5.1 is a diagram of the control flow of a possible agent architecture. The cognitive model that will manage high-level reasoning is shown only as a closed box. It will not be discussed in this section other than its input/output relation — it is the topic of Chapter 6. The importance of encapsulating the cognitive model is that it does not matter for the purposes of this section how it is implemented. Inevitably, there are direct links between the subcomponents of the cognitive model and the rest of the system. However, we believe the level of detail of the current system allows ignoring these links without harm. The components of an agent are:

1. **Simulated Perception**: this will be discussed in Section 5.1.1, but note that raw perceptual data from the perception module is much higher level than raw data in a machine perception sense — our raw data includes relative positions of objects and their abstract physical properties such as object type and color. In a simulation we have perfect environmental information, so it is the job of the sensors to also simulate realistically limited values.

2. **Perceptual (Afferent) Behavior Network**: perceptual behaviors that attempt to to find high-level information from raw sensory data. Typically they respond to *focusing* signals which change field of view, thresholds, distance sensitivity, restrictions on type of object sensed, and the like.

3. **Cognitive Model**: the source of long-range planning and internal motivation (activity not triggered directly by perception).

4. **Efferent Behavior Network**: behaviors that derive activation or deactivation signals. (Note that the afferent and efferent behavior networks are separated only for organizational convenience — they could actually be one network.)

5. **Simulated Effectors**: attempt to modify objects embedded in the kinematics or dynamics simulation.

Although there may be a general feed-forward nature through the above components in order, the connectivity must be a completely connected graph with the following exceptions:

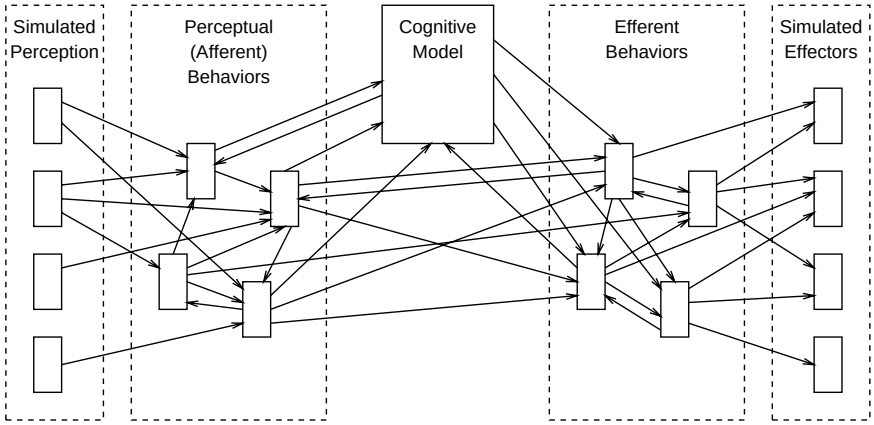

Figure 5.1: Abstract Agent Architecture.

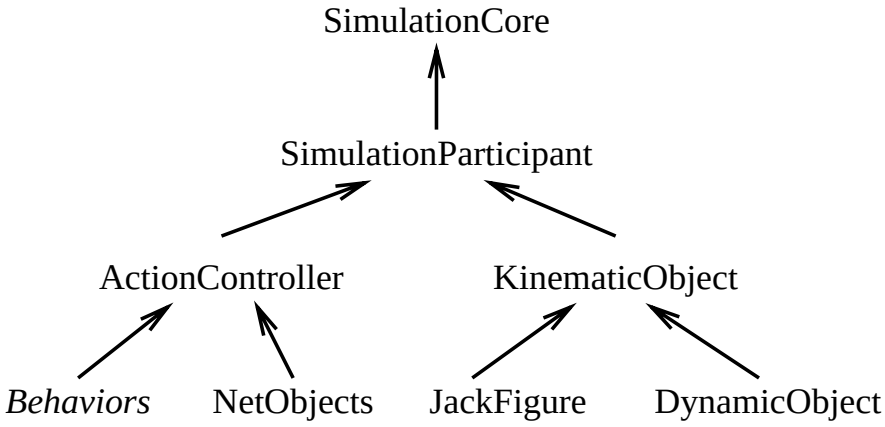

Figure 5.2: Outline of System Class Hierarchy.

1. The cognitive model cannot activate effectors directly.

2. There is no feedback directly from effectors — effector feedback is considered perception (usually proprioception, though pain from muscle fatigue is also possible) and is thus fed-back through the environment.

Raw perceptual information may go directly to the cognitive model or to efferent behaviors, but it is typically routed through perceptual behaviors which derive higher level information and are sensitive to various focusing control signals from the cognitive model, efferent behaviors, or perhaps even other perceptual behaviors. The cognitive model may attempt to re-focus perceptual information through signals to the perceptual behaviors or it may activate or deactivate efferent behaviors in order to accomplish some type of motion or physical change. Efferent behaviors may send signals to effectors, send feedback signals to the cognitive model, or attempt to focus perceptual behaviors.

One typical pattern of activity associated with high-level motivation may be that the cognitive model, for whatever reason, wants to accomplish a complex motion task such as going to the other side of a cluttered room containing several moving obstacles. The cognitive model activates a set of efferent behaviors to various degrees, perhaps an object attraction behavior (to get to the goal) and a variety of obstacle avoidance behaviors. The efferent behaviors then continually activate effectors based on activation levels from the cognitive model and from information directly from perceptual behaviors. Note that this final control flow from perception directly to efferent behavior is what is traditionally called *feedback control*. In another typical pattern of activity, *reflex behavior*, efferent behavior is initiated directly by perceptual behaviors. Note, however, that especially in high-level creatures such as humans, the cognitive model may be able to stop the reflex arc through a variety of inhibitory signals.

5.1.1 The Simulation Model

Rather than implementing models on real robots we will implement and test in detailed simulations that by analogy to the world have a physically-based, reactive environment where some objects in the environment are under the control of agent models that attempt to move their host objects.

For the agent modeler, the main advantage to testing in simulations is the ability to abstract over perception. Because agents are embedded in a simulation, they can be supplied with the high-level results of perception directly, abstracting over the fact that general machine perception is not available. At one extreme agents can be omniscient, having exact information about positions, locations, and properties of all objects in the environment, and at the other extreme they can be supplied with a color bitmap image of what would appear on the agent's visual plane. A good compromise that avoids excessive processing but that also provides for realistically limited perception, is suggested by [Rey88] and also by [RMTT90]. They use the Z-buffering hardware

on graphics workstations (or a software emulation) to render a bitmap projection of what the agent can see, except that the color of an object in the environment is unique and serves to identify the object in the image. The combination of the resulting image and the Z-buffer values indicate all visible objects and their distances, and this can be used for object location or determination of uncluttered areas.

Many models of reactive agents are accompanied by a simulation with 2D graphical output such as [AC87, PR90, HC90, VB90], however, these simulation environments are extreme abstractions over a real environment and assume discrete, two-dimensional, purely kinematic space. Such abstractions are, of course, necessary in initial phases of understanding how to model an intelligent reactive agent, but extended use of a system without real-valued input parameters and immense environmental complexity is dangerous. As will be discussed Section 5.1.3, Simon [Sim81] argues that complex behavior is often due to a complex environment, where the agent responds to environmental complexity through simple feedback mechanisms grounded in sensation. When environmental complexity is not present, the agent modeler, noticing the lack of complexity, may commit *agent bloating*, also discussed in Section 5.1.3, where environmental complexity is accounted for artificially in the agent model.

5.1.2 The Physical Execution Environment

In our model, kinematic and dynamic behavior has been factored out of the agent models and is handled by a separate, common mechanism. The networks of efferent behaviors controlling a conceptual agent in the environment will *request* motion by activating various effectors. The requested movement may not happen due to the agent's physical limitations, collision or contact with the environment, or competition with other behavioral nets.

Simulations of agents interacting with environments must execute on reasonably fine-grained physically-based simulations of the world in order to result in realistic, useful animations without incurring what we call the *agent-bloating* phenomenon, where motion qualities arising from execution in physical environment are stuffed into the agent model. One of Simon's central issues [Sim81] is that complex behavior is often not the result of a complex control mechanism, but of a simple feedback system interacting with a complex environment. Currently, for simplicity, our animations are done in a *kinematic* environment (one considering only velocity and position) and not a *dynamic* one (also considering mass and force). Using only kinematics has been out of necessity since general dynamics models have not been available until recently, and even then are so slow as to preclude even near real time execution for all but the simplest of environments. Kinematic environments are often preferred by some since kinematic motion is substantially easier to control with respect to position of objects since there is no mass to cause momentum, unexpected frictional forces to inhibit motion, and so on. But as we demand more of our agent models we will want them to exhibit properties

that result from interaction with a complex physical world with endless, unexpected intricacies and deviations from desired motion. Unless we execute on a physically reactive environment we will experience one form of agent-bloating where we build the physical environment into the agents. If we build an actual simulation model into agents we have wasted space and introduced organizational complexities by not beginning with a common physical environment. If we build the environmental complexity into the agents abstractly, perhaps through statistical models, we will have initial success in abstract situations, but never be able to drive a meaningful, correct, time-stepped simulation with multiple agents interacting with an environment and each other. We do not mean that statistical and other abstract characterizations of behavior are not necessary – just that abstract description is essential to understanding how the underlying process works and judging when a model is adequate.

The much cited loss of control in dynamic simulations needs to be overcome, and the message of emergent behavior research is that perhaps the most straightforward approach to this is by looking at the plethora of working existence proofs: real animals. Even the simplest of single-celled creatures executes in an infinitely complex physical simulation, and creatures we normally ascribe little or no intelligence to exhibit extremely effective control and goal-orientedness. Animals do this primarily through societies of feedback mechanisms where the lowest levels are direct sensation and muscle contraction (or hormone production or whatever).

In our system dynamic simulations should enjoy the following properties:

- Effectors request movement by applying a force at a certain position to an object.
- Collisions are detected by the system, which will communicate response forces to those participating in the crash or contact situation.
- Viscous fluid damping is simulated by applying a resistance force opposite and proportionate to instantaneous velocity.

For simplicity, and especially when the motion is intended to be abstract, a simulation may still be run on a purely kinematic environment which has the following properties:

1. Effectors request changes in position and orientation, rather than application of force.
2. Every object has some maximum velocity.
3. No motion takes place unless requested explicitly by effectors.
4. Collisions are resolved by stopping motion along the system's estimated axis of penetration.
5. The system adapts the time increment based on instantaneous velocity and size of object along that object's velocity vector so that no object could pass entirely through another object in one time step.

The particular physical simulation approach is to use a simple finite-difference approximation to the equations for elastic solids. Objects are modeled as meshes of point masses connected by springs (including cross connections to maintain shape), where tighter spring constants yield more rigid looking bodies. This approach is discussed by Terzopoulos [TPBF87] and Miller [Mil91] and has the advantage of extreme simplicity and generality. Because it is a discrete approximation to the "integral-level" analytical physics equations it can solve many problems for free, though in general the cost is limited accuracy and much slower execution times than the corresponding direct analytical methods (the results, however, are not only "good enough for animation" but are good enough considering our abstraction level). The model can easily account for phenomena such as collision response, elastic deformation, permanent deformation, breakage, and melting. Finite element analysis yields a better dynamic behavior to the point-mass mesh (for accuracy and execution time), but is not as general as the mass/spring approach and cannot model breakage and melting.

5.1.3 Networks of Behaviors and Events

The insulation of the cognitive model with networks of behaviors relies on emergent computation. It is important to understand, then, why emergent computation works where a strict hierarchy would not, and what problems an emergent computation approach poses for the agent designer and how these problems can be overcome.

For simplicity, existing high-level task-simulation environments tend to model activity in strict tree-structured hierarchies, with competition occurring only for end effectors in simulation models as in [Zel82], or for position of a body component in purely kinematic models. However, for some time behavior scientists and those influenced by them have argued that although there is observable hierarchy, behavior – especially within hierarchical levels – is not tree structured but may have an arbitrary graph of influence [Gal80, Alb81]. In particular a theory of behavior organization must anticipate behaviors having more than one parent and cycles in the graph of influence.

The central observation is that in many situations small components communicating in the correct way can gracefully solve a problem where a direct algorithm may be awkward and clumsy. Of course this approach of solving problems by having a massive number of components communicating in the right way is nothing new: cellular automata, fractals, approximation methods, neural networks (both real and artificial), finite-difference models of elastic solids [TPBF87], simulated annealing, and so on use exactly this approach.

The drawback to such massively parallel systems without central control is typically the inability to see beyond local minima. Certainly a high-level planner may periodically exert influence on various system components in order to pull the system state from a local minimum. The appropriate introduction of randomness into component behavior, however, can help a system settle in a more globally optimal situation. This randomness can be from explicit

environmental complexity, introduction of stochastic components, limited or incorrect information, or mutation.

This general approach is not limited to low-level interaction with the environment. Minsky proposes a model of high-level cognition in [Min86] where a "society of agents" interacts (organized as a graph) to accomplish high-level behavior. Pattie Maes [Mae90] has proposed an approach to high-level planning through distributed interaction of plan-transformation rules. Ron Sun proposed a distributed, connectionist approach to non-monotonic reasoning [Sun91].

All of these approaches rest on emergent computation — behavior resulting from communication of independent components. Common objections to such an approach are:

1. it is doomed to limited situations through its tendency to get stuck in local minima.

2. in order to implement, it requires an unreasonable amount of weight fiddling.

The first objection has already been addressed. The second is a serious concern. Our proposed solution will be to transfer the weight assignment process to some combination of the behavioral system and its environment. An evolution model is one way to do this, as Braitenberg [Bra84] does with his vehicles, or as the Artificial Life field would do. Another is to combine simple behavioral psychology principles and a connectionist learning model in a creature that wants to maximize expected utility [Bec92], then provide a reinforcement model that punishes the creature whenever it does something wrong (like hits something).

Making it easy for a human designer to engage in an iterative design and test process is another approach. Wilhelms and Skinner's [WS90] system does exactly this by providing a sophisticated user interface and stressing real-time or at least pseudo-real-time simulation of creatures interacting with the environment. However, we will not pursue this approach for the following reasons:

- Self-supervised weight assignment as agents interact with their environment is clearly more desirable from a simulation point of view, though it sacrifices direct control for realism and ease of use.

- For reasons discussed in Section 5.1.2, we encourage execution in complex physically-based environments — an emphasis precluding real-time playback on standard displays.

5.1.4 Interaction with Other Models

Our approach then, is to control physical simulation with the abstract findings of the animal sciences, beginning by using the tricks that low-level animals

use. Low-level animal behavior tends, through its extensive use of environmental feedback, to be incremental — it makes a new decision at every moment considering the current state of the environment. For this reason it is considered *reactive* because it will incorporate unexpected events in constant time as though they had been planned for in advance. Certainly human behavior exhibits short and long term planning that cannot be explained by purely reactive processes. We hope to discover and elaborate abstraction layers with long-term symbolic planning at the top and feedback mechanisms at the bottom.

However, there are countless areas in the neuro-physiological level study of humans that are not well enough understood to allow direct or even abstract implementation. Behavior such as human walking, to our knowledge, cannot be described accurately in terms of feedback from proprioceptive sensors and perhaps the vision system. Many such components of human behavior can, however, be modeled directly by abstract methods and we ought to be able to incorporate these as we progress. These direct methods will often be considerably faster than the corresponding neuro-physiological models which typically rely an massive parallel computation and will not run efficiently on sequential machines. So even if there were a neural-level walking algorithm for humans, in cases where the robustness and correctness of locomotion are unlikely to contribute to the usefulness of the overall simulation, say because the are very few obstacles and the terrain is simple, it would be useful to use the direct method to save time.

Algorithms that are designed to manipulate human body parts directly can be incorporated into the described system as long as an approach to conflict resolution is also implemented should there be more than one behavior attempting to control a given body segment (this can be weighted averaging, or strict prioritization, or whatever). Since the rest of the system is totally reactive and considers the current state of the environment at every instant, it does not matter whether the physical model, kinematic model, or some other process modified a given object.

As will be discussed later, if a collision is detected, all behaviors controlling the offenders will be sent messages indicating the points of collision and the impulse forces. For objects that do not have explicit velocity information a velocity is simply computed by looking at the system's current δt and how far the object was moved over the previous time step. The receiving behaviors can do whatever they wish with the information — replan, respond to it, ignore it, or anything appropriate. The only difficulty is when two objects both controlled by unyielding direct control methods collide — they both will fail to move any further. This can be avoided by keeping the number of objects under direct control limited, or by always implementing some sort of failure recovery method. Since communication is based on contact positions and forces, different control approaches can always communicate through their effects on the environment.

Even other physically-based object controllers such as finite element analysis, or direct rigid body dynamics can be incorporated. Direct control mes-

sages across computer networks or through operating system pipes to dynamics simulation packages can also be used. Direct manipulation is also possible though there must be a way to compensate for speed differences if the simulation is running much slower than real time. One way to do this is to have the user move an object while the simulation is frozen, ask the user how long in simulation time that action should take, then use a direct kinematic controller to do a spline-smoothed approximation of the user's motion as the simulation continues.

5.1.5 The Simulator

The simulation of intelligent agents interacting with a reactive environment is advanced incrementally in small adaptive time steps. The Δt for a time slice will be no greater than $\frac{1}{30}$th of a second (the typical video frame rate) and can be as small as floating point precision will allow. Typically, kinematically controlled objects will update on $\frac{1}{30}$ths of a second but dynamically controlled objects when experiencing high-impact collisions will want very small time steps. The distinction made in earlier sections between agents and the environment is only conceptual at the simulator level— both components of agent models and components of physical models are considered first-class participants in a single simulation. Every time step is broken down into a number of synchronizing phases. The synchronizing phases are motivated by Haumann and Parent's behavioral simulation system [HP88], but augmented with features for adaptive time steps. The following messages are broadcast in the given order to every participant on every time step:

start This tells participants a time step is beginning. Typically buffers for collecting messages in the affect stage are cleared here, and state information is saved in case there is a backup.

affect Participants that attempt to modify the state of other participants may do so here by looking at the state of the environment and sending messages calling for change. However, no participant is allowed to change the appearance of an internal state – all calls for change must be buffered and dealt with in the respond stage.

respond Objects are allowed to change their externally accessible state variables, such as position and color for environmental objects or activation level for behavioral network components.

data inject Rigid rules, such as static non-interpenetration, are enforced here after objects have had a chance to update themselves. Pure kinematic scripting may be done here also.

In addition, at any phase any object may request that the time step be restarted with a smaller time step if it feels the simulation is running too fast. A participant need only suggest a new Δt, perhaps half the previous Δt, then

call for a global backup. All participants are required to store persistent state information in the start phase, and restore this state if a backup is called. Participants may also request new Δt values without requesting a backup and if no requests are made, the system will try to double the Δt on every step until it reaches $\frac{1}{30}$th of a second.

The Class Hierarchy

An object-oriented approach is natural for implementing such a system, and in order to allow fast execution for interactive development, we chose to use C++. The cognitive model is based on Lisp and Prolog and will communicate through C-callout functions to C++.

The class hierarchy is outlined in Figure 5.2. The `SimulationCore` class manages the clock, the current Δt, and a list of participants to which it broadcasts synchronizing messages on each time step. `SimulationParticipant` encapsulates all participants in the simulation. A distinction is made between participants that have spatial qualities (the `KinematicObj` class, which tends to operate in the respond stage) and participants that try to modify the state of other participants (the `ActionController` class which operates in both the affect and respond stages).

Objects, actions, and networks

The `KinematicObj` class is broken into a `JackFigure` class which allows use of the *Jack* figures. The `DynamicObj` class is for objects controlled by dynamic simulation and is a subset of kinematic objects because any dynamic object ought to be able to respond to any message intended for a purely kinematic object.

`ActionControllers` are broken down into *Behaviors* which include perceptual and efferent behaviors discussed above and `NetObjects` which connect them. Our network package allows general neural network-type constructions, though it is important to note that the system is not a neural network because:

- the 'neurons' (nodes) may be of arbitrary complexity.
- the messages passed along network links may be very complicated, in particular, they can be pointers to objects.

Neural networks can be used within behaviors, however, and we have begun experimenting with backpropagation learning [FS91, HKP91] and recurrent networks [Bec92, Sch90] as ways of learning how to behave.

All `ActionController` instances must respond to messages requesting the start and end time of activation or an indication that the action has not started or ended. This allows general implementation of conditional action sequencing through meta-behaviors that on each time step check to see of a particular action has started or ended or if an action of a particular type has begun or ended.

5.1.6 Implemented Behaviors

Presently, reactive agents resemble Reynolds' birds [Rey87] in that on each time step the agent moves along its local z-axis. Efferent behaviors attempt to modify the global orientation of the local z-axis and also determine by how much it will move forward.

Our primary perceptual behavior is the `Closest-k` sensor. Its arguments are the number and type of objects to which it is sensitive. In addition the sensor needs to know what its relative position is to its host environmental object (the z-axis of this transformation will be the forward direction). The sensor produces k groups of outputs which will contain information on the closest k objects of the defined type. Each group will have three floating-point output nodes: the distance from the sensor's current global origin, the angle between the sensor's z-axis and a vector to the detected object's centroid, and the radius of the detected object (we currently use bounding cylinders and bounding spheres around objects to simplify calculations). We have found no pressing need yet to construct a corresponding `furthest-k` sensor.

Another perceptual behavior that is not behaviorally motivated, but useful for abstract control is the `ObjectSensor` that is sensitive only to a particular object in the environment and has outputs similar to a `closest-k` sensor.

The efferent behaviors we have implemented are loosely motivated by neuro-ethological findings about real creatures and are discussed by Wilhelms and Skinner [WS90] in their implementation and abstraction of Braitenberg's *Vehicles* [Bra84] and in Anderson and Donath's emergent reflexive behavior system [AD90]. What is novel about our behaviors is their dynamically adjustable tuning parameters:

1. threshold distance
2. field of view (angular threshold)
3. sensitivity of activation level to distance of object and angular distance of object. Distance from threshold and angle from center scaled by an exponent and a constant (both focusing parameters).

these can be adjusted directly by the cognitive component or by other behaviors. We have found the following behaviors particularly useful:

Attract go toward either closest-k of a certain type of object or a specific object. Loosely corresponds to *teleotaxis* in animals [Gal80].

Avoid go away from a particular object or from a certain type of object. Also a form of teleotaxis.

GoAround uses the incremental obstacle avoidance approach outlined by Reynolds [Rey88], which is based on how birds avoid obstacles while flying.

Achieve go to a particular place or a particular place relative to another object.

AttractLine go directly towards a line in space – used to follow walls.

AvoidLine go directly away from a line in space – used to avoid walls.

5.1.7 Simple human motion control

An important approach to developing human behaviors is to attempt to apply the behaviors appropriate for low level animals directly to humans and see where they appear awkward in order to understand what is missing. Our initial attempt allows the above efferent and perceptual behaviors to be used directly in a human model except that instead of simply moving along the z-axis the human agent attempts to reduce its current and desired headings by taking steps. The stepping is accomplished by using a locomotion algorithm (Section 5.2). Our walking algorithm is incremental in that it only needs to know where the next footstep should go and how it should be oriented. Our approach to supplying footsteps clips the difference between the current and desired headings to 45 degrees and places the next foot to be moved alongside the new heading. Heading is determined by the orientation of the lower torso (which is oriented by the walking algorithm). The size of the step is currently based on the curvature of the turn (smaller steps for larger turns), though certainly step length should have other influences.

Simulations involving this simple human agent model show very clearly that humans anticipate the effects of placing the next step (perhaps through behavioral gradient estimation) rather than blindly following influences of the current situation. In all complicated situations under the described model the agent tends to oscillate around the behavior gradient.

There are other agent behaviors that are in the process of migrating into this behavioral framework. In the next sections we look at locomotion, strength-guided lifting, collision avoidance, and posture planning.

5.2 Locomotion

[2]Locomotion provides a tremendous extension of the workspace by moving the body to places where other activities may be accomplished. A locomotion system should provide a reasonable configuration of the figure at any time as it moves along a specified input path. There have been many efforts to make this process more realistic and automatic, which can roughly be summarized into two major approaches: kinematic and dynamic controls. Rotoscopy data and biomechanics knowledge can be utilized to control locomotion kinematically, but empirical data must be generalized to get walking under parametric control. A dynamics computation can be done to get the locomotion path and some of the body motion characteristics, but biomechanics knowledge is

[2]Hyeongseok Ko.

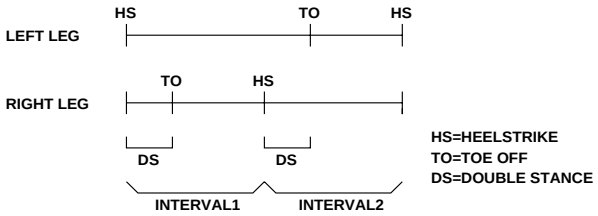

Figure 5.3: The Phase Diagram of a Human Walk.

useful in determining the details and reducing the complexity of the whole body dynamic system.

These two approaches can be applied to get straight path walking. The natural clutter and constraints of a workplace or other environment tend to restrict the usefulness of a straight path so we must generalize walking to curved paths. We have already seen the stepping behavior and the collision avoidance path planning in *Jack*, so a locomotion capability rounds out the ability of an agent to go anywhere accessible. First we give some necessary definitions for the locomotion problem, then look at feasible ways of implementing curved path walking.

At a certain moment, if a leg is between its own heelstrike (beginning) and the other leg's heelstrike (ending), it is called the *stance leg*. If a leg is between the other leg's heelstrike (beginning) and its own heelstrike (ending), it is called the *swing leg*. For example, in Figure 5.3, the left leg is the stance leg during interval 1, and the right leg is the stance leg during interval 2. Thus at any moment we can refer to a specific leg as either the stance or swing leg with no ambiguity. The joints and segments in a leg will be referenced with prefixes *swing* or *stance*: for example, swing ankle is the ankle in the swing leg.

Let $\Theta = [\theta_1, \ldots, \theta_J]$ be the joint angles and $\Lambda = [l_1, \ldots, l_S]$ be the links of the human body model. Each θ_i can be a scalar or a vector depending on the DOFs of the joint. Let Σ be the sequence of $(\vec{h_i}, \vec{d_i}, sf_i, lorr_i), i = 0, 1, \ldots, n$, where h_i is the heel position of the ith foot, d_i is the direction of the ith foot, sf_i is the step frequency of the ith step, and $lorr_i$ ("left" or "right") is 0 when the ith foot is left foot and 1 otherwise. The locomotion problem is to find the function f that relates Λ and Σ with Θ at each time t:

$$\Theta = f(\Lambda, \Sigma, t). \tag{5.1}$$

Usually the function f is not simple, so the trick is to try to devise a set of algorithms that computes the value of Θ for the given value of (Λ, Σ, t), depending on the situation.

5.2.1 Kinematic Control

The value of Θ can be given based on rotoscopy data. Two significant problems in this approach are the various error sources in the measurements and

the discrepancy between the subject's body and the computer model. When applying kinematic (empirical) data to the model, obvious constraints imposed on the walking motion may be violated. The most fundamental ones are that the supporting foot should not go through nor off the ground in the obvious ways depending on the situation, and that the global motion should be continuous (especially at the heel strike point). The violation of these constraints is visually too serious to be neglected. During motion generalization the error is likely to increase. Therefore in the kinematic control of locomotion, one prominent problem is how to resolve errors and enforce constraints without throwing away useful information that has already been obtained.

So how can we generalize empirical data? The walk function Θ depends on many parameters, and simple interpolation cannot solve the problem. Boulic, Magnenat-Thalmann and Thalmann's solution for this problem [BMTT90] is based on the relative velocity (RV), which is simply the velocity expressed in terms of the height of the hip joint H_t (e.g. $2H_t/sec$). For example the height of the waist O_s during the walk is given by

$$-0.015RV + 0.015RV \sin 2\pi(2t - 0.35)$$

where t is the elapsed time normalized by the cycle time. Because this formulation is based on both body size and velocity, the approach can be applied under various body conditions and velocities.

5.2.2 Dynamic Control

Bruderlin and Calvert built a non-interpolating system to simulate human locomotion [Bru88, BC89]. They generated every frame based on a hybrid dynamics and kinematics computation. They could generate a wide gamut of walking styles by changing the three primary parameters step length, step frequency, and speed. The example we use here is based on their work.

Their model is divided into two submodels. The one (stance model) consists of the upperbody and the stance leg. The other (swing model) represents the swing leg. In the stance model, the whole upperbody is represented by one link and the stance leg is represented with two collinear links joined by a prismatic joint. So the stance leg is regarded as one link with variable length ω. In the swing model, the two links represent the thigh and calf of the swing leg. In both models, links below the ankle are not included in the dynamic analysis and are handled instead by kinematics.

Two sets of Lagrangian equations are formulated, one set for each leg phase model. To do that, the five generalized coordinates, ω, θ_1, θ_2, θ_3, θ_4 are introduced: ω is the length of stance leg; θ_1, θ_2, θ_3 is measured from the vertical line at the hip to the stance leg, upperbody, and the thigh of the swing leg, respectively; and θ_4 is the flexion angle of the knee of the swing leg. During the stance phase the stance foot remains at (x, y), so x and y are regarded as constants. Once those five values of general coordinates are given, the configuration of the whole body can be determined by kinematics.

So the goal of the dynamics computation is to obtain the general coordinate values.

We will focus only on the stance model here. By formulating the Lagrangian equation on the stance model, we get the three generalized forces F_ω, F_{θ_1}, and F_{θ_2}.

$$\begin{aligned} F_\omega &= m_2\ddot{\omega} - m_2 r_2 \ddot{\theta_2}\sin(\theta_2-\theta_1) - m_2 r_2 \dot{\theta_2}(\dot{\theta_2}-\dot{\theta_1})\cos(\theta_2-\theta_1) \\ &\quad -m_2\omega\dot{\theta_1}^2 - m_2 r_2 \dot{\theta_2}\dot{\theta_1}\cos(\theta_2-\theta_1) \\ &\quad +m_2 g\cos\theta_1 \qquad (5.2) \\ F_{\theta_1} &= (I_1 + m_1 r_1^2 + m_2\omega^2)\ddot{\theta}_1 + 2m_2\omega\dot{\omega}\dot{\theta_1} \\ &\quad -(m_1 r_1 + m_2\omega)g\sin\theta_1 + m_2 r_2\ddot{\theta_2}\omega\cos(\theta_2-\theta_1) \\ &\quad -m_2 r_2\dot{\theta_2}^2\omega\sin(\theta_2-\theta_1) \qquad (5.3) \\ F_{\theta_2} &= -m_2 r_2\ddot{\omega}\sin(\theta_2-\theta_1) + m_2 r_2\omega\ddot{\theta_1}\cos(\theta_2-\theta_1) \\ &\quad +(I_2 + m_2 r_2^2)\ddot{\theta_2} - m_2 g r_2\sin\theta_2 \\ &\quad +2m_2 r_2\dot{\omega}\dot{\theta_1}\cos(\theta_2-\theta_1) + m_2 r_2\omega\dot{\theta_1}^2\sin(\theta_2-\theta_1) \qquad (5.4) \end{aligned}$$

which can be written in a matrix form as

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} \ddot{\omega} \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} = \begin{bmatrix} F_w + b_1 \\ F_{\theta_1} + b_2 \\ F_{\theta_2} + b_3 \end{bmatrix} \qquad (5.5)$$

Let $x(t)$ be the value of x at time t; x can be a scalar, a vector, or a matrix. In equation 5.5 at time t, everything is known except $\ddot{\omega}, \ddot{\theta}_1, \ddot{\theta}_2, F_w, F_{\theta_1}, F_{\theta_2}$. If we give the generalized force values at time t, the above linear equation can be solved for accelerations. The position at the next time step is

$$\vec{q_r}(t+\Delta t) = \vec{q_r}(t) + \Delta t\vec{q_r}(t) \qquad (5.6)$$

$$\vec{q}(t+\Delta t) = \vec{q}(t) + \Delta t\vec{q_r}(t). \qquad (5.7)$$

The internal joint torques $F(t) = [F_w, F_{\theta_1}, F_{\theta_2}]^T$ are not fully known. Bruderlin adopted some heuristics from biomechanics to handle this problem. For example, to get F_{θ_1}, he noted

> A significant torque at the hip of the stance leg occurs only just after heel strike and lasts for about 20% of the cycle time. Also, the torque during this time interval is such that it rapidly reaches a maximum value and decays quickly towards the end [IRT81, Win90].

and approximated it by a constant function

$$F_{\theta_1} = \begin{cases} c & \text{for the first 20\% of cycle time} \\ 0 & \text{for the remaining cycle time.} \end{cases} \qquad (5.8)$$

Similarly, in modeling the hip joint and waist joint as springs, the internal torques were given by the following formulas:

$$F_\omega = k_\omega(\omega_{des} + pa_3 - \omega) - v_\omega\dot{\omega} \tag{5.9}$$

$$F_{\theta_2} = -k_2(\theta_2 - \theta_{2\_des}) - v_2\dot{\theta_2}. \tag{5.10}$$

To handle the errors coming from these approximations, several checkpoints were set. For example, the posture at the heel strike after the current step can be derived based on the step symmetry. Integrating $F(t)$ until the checkpoint, we can compare the result with the desired one. The constants in the equations above (e.g. c, a_3) are adjusted according to the difference. This process is repeated until the integration brings it close enough to the desired posture.

5.2.3 Curved Path Walking

Research on biped locomotion has focused on sagittal plane walking in which the stepping path is a straight line. Unfortunately, simply treating a complex walking path as a sequence of straight-line path segments does not work. The problems of turning and coordinating the limb motions at the turns is frequently neglected and the rigid appearance of the resulting abrupt mid-air turns is clearly unacceptable animation.

In building a general planar locomotion behavior, we utilized pre-existing straight path ideas. We will call a linear path locomotion algorithm a 1D system; we will use it as a process within our 2D behavior. For every 2D step, we will consider its *underlying 1D step*, and the 1D system will provide some needed information. Our generalization algorithm from 1D to 2D is based on the intuition that there should be a smooth transition between linear and curved locomotion. If the curvature is not large, the 2D walk generated should be close to the 1D walk given by the underlying 1D system. In particular, the degenerate 2D case of a straight line should be exactly the same as that produced by the underlying 1D system. Since no assumptions are made about the underlying 1D system, any 1D locomotion algorithm can be generalized into our 2D one. Moreover, the underlying 1D system will determine the stylistics (or faults) of the curved path walk.

When requested to create a step, the 1D step generation algorithm provides information to the 2D system. The 2D system first computes the center [of mass] site trajectory and the locations of both hip joints. The locations of the feet are computed based on the 1D step information. Because we have the hip and foot locations of both legs, the configurations of both stance and swing legs can be determined. The banking angle is computed, and the upper body is adjusted to move the center of mass to achieve the required banking. The parameters Θ that determine the configuration of the whole body is now available for *Jack* display. This entire process is incremental at the step level, so that it fits neatly into the behavioral simulation paradigm.

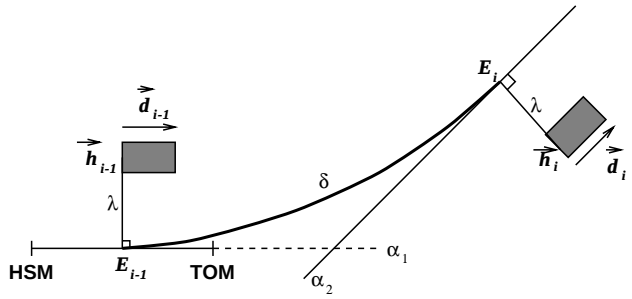

Figure 5.4: The Step Length of the Underlying 1D Step.

Specifying the Walk

The direct input to the locomotion behavior is a *step sequence* Σ of 4-tuples,

$$\sigma_i = (\vec{h_i}, \vec{d_i}, sf_i, lorr_i), i = 0, \ldots, n. \tag{5.11}$$

Each tuple σ_i is called the ith *foot description*. The pair of adjacent two foot descriptions (σ_{i-1}, σ_i) is called the ith *step description* or simply the ith *step*.

Even though we have maximum control of locomotion by using the step sequence, generating such a sequence directly is a tedious job. The behavioral simulation can generate a path incrementally, or an interactive user could specify a curved path. In either case, the specification is automatically transformed to a step sequence.

The specification of a walk in 1D can be done by giving a sequence of $(sl_i, sf_i), i = 1, \ldots, n$. Each (sl_i, sf_i) affects the type of current step, starting from the current heelstrike to the next one. For every step description (σ_{i-1}, σ_i) in 2D, we consider its *underlying 1D step*. The step frequency sf_{1D} of this 1D step is given by sf_i of σ_i. We can draw 2 lines α_1, α_2 on the horizontal plane as shown in the Figure 5.4: α_1 is in the direction of $\vec{d}_{i-1}$ displaced by λ from $\vec{h}_{i-1}$; α_2 is in the direction of $\vec{d}_i$ displaced by λ from $\vec{h}_i$. Let δ be the arc length of the spline curve from E_{i-1} to E_i, where E_{i-1} and E_i are the projections of the heel positions to the lines α_1, and α_2, respectively. (The derivatives at the end points of this spline curve are given by $\vec{d}_{i-1}$ and $\vec{d}_i$.) The step length sl_{1D} of the underlying 1D step is given by this δ.

Path of the Center Site

For this discussion we will keep the model simple by assuming that the center of mass moves along a straight line from the heelstrike moment of the stance leg (HS) to the toe off moment of the swing leg (TO) (Figure 5.5). The trajectory of the center site during this double stance phase (DS) is given by the current stance foot direction $\vec{d}_{i-1}$. From the TO to the next HS, the center site moves along a spline interpolation (Figure 5.5). At both ends of the spline, the derivative of the must match that of the adjacent line segments for

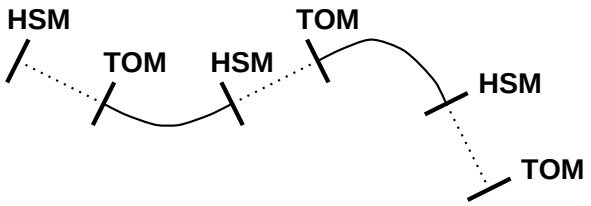

Figure 5.5: The Trajectory of the Center Site (Top View).

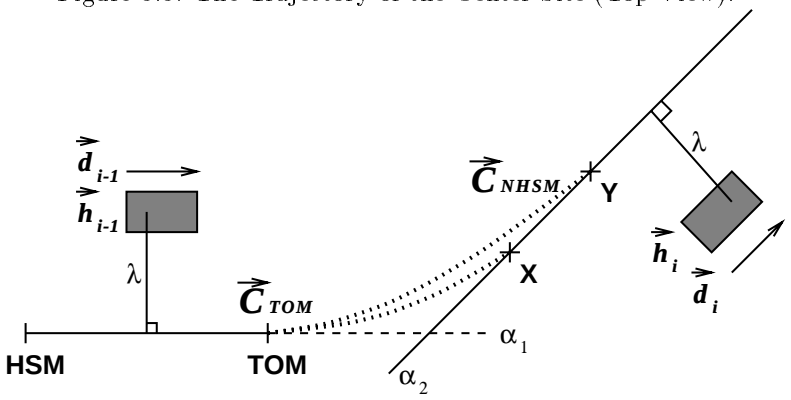

Figure 5.6: The Position of Center Site at Heelstrike Moment.

first order continuity. Through the whole locomotion, the pelvis is assumed to face the derivative direction of the center site path, and be vertical to the ground. The torso can be bent in any direction, a part of which is given by the underlying 1D algorithm, and another part is given from a banking adjustment.

To derive the spline curve, we need the position $\vec{C}_{NHS}$ and derivative $\dot{\vec{C}}_{NHS}$ of the center site at the *next* HS, as well as $\vec{C}_{TO}$ and $\dot{\vec{C}}_{TO}$ at TO which are provided by the underlying 1D system (Figure 5.6). The assumptions above imply that $\dot{\vec{C}}_{NHS} = \vec{d}_i$, and $\vec{C}_{NHS}$ should be put at point X somewhere on the line α_2. Let η_{1D} and τ_{1D} be the length of the center site trajectory from HS to TO, and from TO to the next HS, respectively, during the underlying 1D step. Let η_{2D} of corresponding 2D step be similarly defined. Let $\tau_{2D}(X)$ be the arc length (top view) of the spline from $\vec{C}_{TO}$ to X in Figure 5.6. Now the position of the center site $\vec{C}_{NHS}$ at the next HS is set to the point X on the line α_2 such that

$$\frac{\eta_{1D}}{\tau_{1D}} = \frac{\eta_{2D}}{\tau_{2D}(X)}. \tag{5.12}$$

This definition of $\vec{C}_{NHS}$ is based on the smooth transition assumption from 1D locomotion to 2D. By a mapping which preserves arc length ratio [Gir87, SB85, Far88], we can find the correspondence between the 2D trajectory of the center site and underlying 1D one. Note that this definition also makes

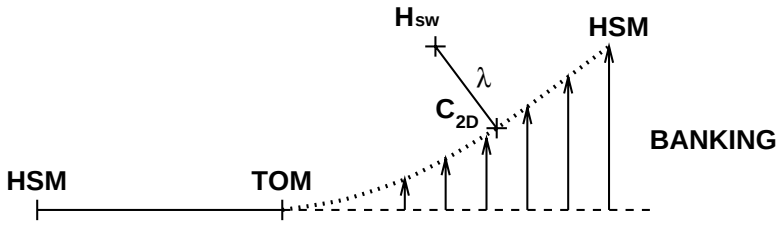

Figure 5.7: Banking of the Center Site.

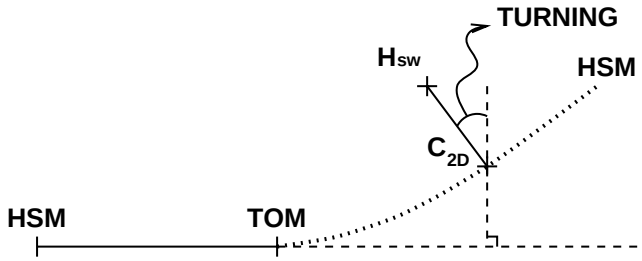

Figure 5.8: Turning of the Center Site.

the degenerate case of 2D walk exactly same with the corresponding 1D walk.

The displacement of the curved path from the underlying linear path is produced by banking as shown in Figure 5.7. The position of the displaced center site in 2D step is C_{2D}, and H_{sw} is the position of the swing hip. Banking mostly results from ankle joint adjustment. Even though the center site is put on the spline curve by the ankle angle, the upper body has not bent yet to generate the overall correct banking of the whole body. The banking should be considered in terms of the center of mass of the body. The overall banking is given by

$$\phi = \arctan(\frac{\kappa v^2}{g}) \tag{5.13}$$

where v is the velocity, g is the gravity, and κ is the curvature of the path [Gir87]. Here we use the spline curve of the center site as an approximation to get the curvature. The upper body should be bent so that the center of mass (which is in the upper body) may make the angle ϕ around the stance ankle with respect to the ground. Iteration can be used to compute the current center of mass and reduce the difference from the current one and the desired one.

We assume that the length of the dynamic leg ω in 2D locomotion at a moment t, is the same as ω at the corresponding moment in the underlying 1D step. So the displaced center site C_{2D} will be lower than the corresponding 1D center site C_{1D}. The position (x_1, y_1, z_1) of the hypothetical ankle is available from the old foot location. Let (x_2, y_2, z_2) be the position of the swing hip H_{sw} in Figure 5.7. The horizontal components x_2 and z_2 can be computed

from the derivative at C_{2D} and λ. In Figure 5.7, λ is the distance between H_{sw} and C_{2D}, and this distance is along the perpendicular direction of the derivative.

Because we assumed that ω is the same in 1D and 2D locomotion, we have

$$| (x_1, y_1, z_1) - (x_2, y_2, z_2) | = \omega \tag{5.14}$$

where ω is given by the underlying 1D system. The value of y_2 that satisfies the above equation is the height of both H_{sw} and C_{2D}. Because the pelvis is assumed to be upright through the steps, the height of the stance hip H is also y_2.

The Stance Leg

The center site movement of 2D locomotion during DS is the same as that of the 1D one, including the height component, so the stance leg configurations are given by the underlying 1D system. During the single stance phase, we still use 1D system to get the joint angle at the ball of foot. But because the center site begins to deviate, the other joint angles should be computed.

In the stance leg, after the foot is put flat on the ground, the toetip is regarded as the root because that point is not moved until the next toe off. Because the joint angle at the ball of the foot is provided by the 1D algorithm, we have the location A of the ankle. Since the location of the hip is also available the configuration of the stance leg can be determined.

The Swing Leg at the Double Stance Phase

Because a revolute joint is assumed at the ball of foot, if we exclude the possibility of sliding, the toe group of the swing foot should stay fixed on the ground during the DS. Because there are 3 links between the swing hip and the ball of foot, we should resolve the redundancy in a reasonable way. If we use the ball of the foot joint angle in the 1D algorithm this redundancy goes away. This approximation works well in most of the cases. But when both the direction change and the step length (the distance between the adjacent steps) are extremely large, the distance $| \vec{\rho}_{sw} |$ from H_{sw} to A_{sw} becomes too long to be connected by the lengths of thigh and calf. This problem is solved by increasing the angle at the ball of foot until $| \vec{\rho}_{sw} |$ becomes less than the sum of thigh and calf. Then $\vec{\rho}_{sw}$ is used to get the joint angles at ankle, knee, and hip.

The Swing Leg at the Single Stance Phase

The trajectory (top view) followed by the swing ankle is approximated by a second degree Casteljau curve [Far88]. The 3 control points are given by the position D_1 of the current swing ankle at TO, D_2 which is the symmetric point of the stance ankle with respect to the line α, and the ankle position D_3 at the next heel strike point (Figure 5.9).

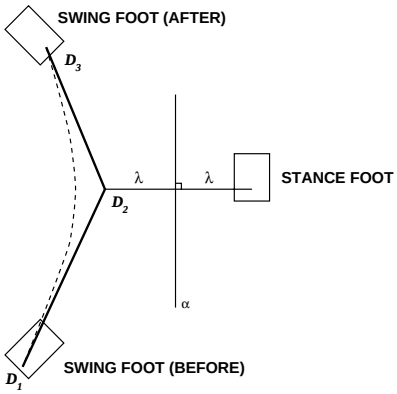

Figure 5.9: The Path of the Swing Foot.

The height component of the ankle is determined by the underlying 1D algorithm. So now we have the locations A_{sw} and H_{sw}, and can determine the swing leg configuration except for the two indeterminacies. At the moment of heelstrike, the swing leg should have been prepared for the next step. Because linear walking is assumed from HS and TO, the hip and ankle angles in swing leg should become 0 except for the bending direction. So we should somehow adjust the swing leg and foot from the *chaos configuration* (at TO) to the *ordered configuration* (at the next HS).

At toe off, the displaced (rotated around the forward axis) foot gets to the normal position very quickly and it is approximated by an exponential function that decreases rapidly to 0: for some positive constant G, we let θ_3^x at time t be

$$\bar{\theta}(t) = \bar{\theta}(0) \cdot \exp^{\frac{-Gt}{t_{sw}}} \tag{5.15}$$

where t is the elapsed time after TO, and t_{sw} is the duration between the TO and the next HS. The rotation of the swing leg around the the axis from the swing hip to the swing ankle is approximated by a parabola given by

$$\tilde{\theta}(t) = \tilde{\theta}(0) \cdot (\frac{t_{sw} - t}{t_{sw}})^2 \tag{5.16}$$

If the 2D locomotion path is actually a straight line, the walk generated is exactly the same as the walk given by the underlying 1D system. For example, in the consideration of swing leg motion during the single stance phase, D_1, D_2, and D_3 will be collinear and parallel to the walking path. Also in the equations 5.15 and 5.16, both $\theta_3^x(0)$ and $\theta_7^z(0)$ will be 0 and the trajectory of the swing leg will be exactly the same as that of 1D walking.

5.2.4 Examples

Figure 5.10 shows foot steps generated by the interactive *step editor*. Figure 5.11 shows the walking path generated by the interactive *path editor*. Fig-

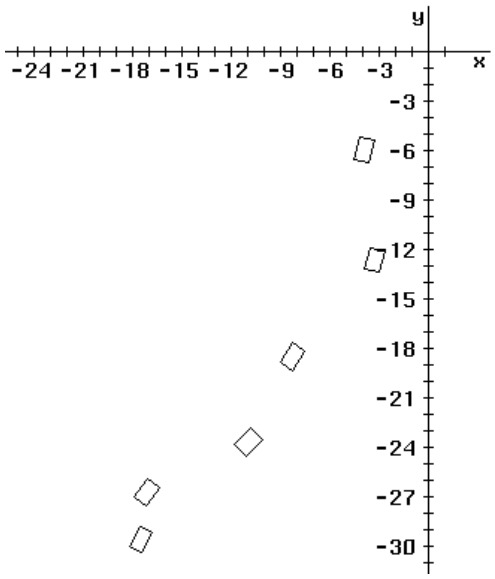

Figure 5.10: Steps Generated by the Step Editor.

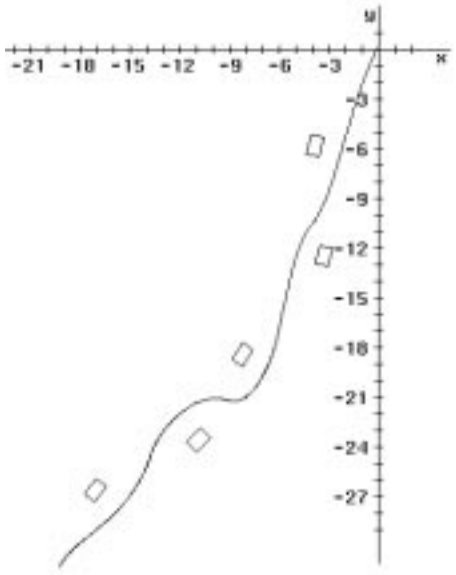

Figure 5.11: Steps Generated by the Path Editor.

ure 5.12 shows snapshots during a turning step. Finally, Figure 5.13 shows a path generated incrementally by the approach and avoidance behaviors.

5.3 Strength Guided Motion

[3]Human motion is likely to be hybrids of many motion and path generation techniques. The task is to find effective combinations that provide realistic motion behaviors while simultaneously offering the user reasonable and intuitive control mechanisms. We have already seen several methods using constraints and other simple control methods that implement various *basic* human activities such as reaching, looking, grasping, and balancing. Now we will look closer at a task-level control algorithm for object manipulation by an end-effector, such as lifting a load to a specified position in space. The resulting motion is certainly dictated by the geometric limits and link structure of the body; but more importantly the motion is strongly influenced by the *strength* and *comfort* of the agent.

5.3.1 Motion from Dynamics Simulation

Torques may be used to physically simulate motions of a figure. Typically the joint responses are conditioned by springs and dampers so that responses to external forces can be computed. Such force- and torque-based methods are called dynamic simulations [Gir87, Gir91, AGL87, IC87, WB85, Wil87, FW88, Wil91, Hah88, HH87, Bar89]. Solving the dynamic equations, an initial value problem, is computationally expensive, especially if the joints are stiff [AGL87]. Natural external forces such as gravity and collision reactions easily yield motions which are free-swinging or passive (purely reactive), but which give the unfortunate impression of driving a hapless mannequin or puppet. As might be expected, the best examples of dynamic simulation come from crash studies [Pra84] where rapid deceleration produces forces that typically overwhelm any agent-initiated torques. In less violent motions, the torques may be derived from a spring or vibration model. Such have been used to create convincing motions of worms, snakes, and other flexible objects [Mil88, PW89], but this cannot be the same mechanism used for human figure motion. Dynamic simulations are annoyingly difficult to control by an animator since force space specifications are highly non-intuitive.

Kinematic and inverse kinematic approaches are easier to manipulate and may create the right "look," but suffer from potentially unrealistic (or unspecified) velocities or torques in the body joints. These problems have been addressed as boundary value problems with objective functions. The trajectories are then solved by global optimization approaches [WK88, Bre89] or control theory [BN88], but their methods presume complete knowledge of the driving conditions and overall constraints.

[3]Philip Lee.

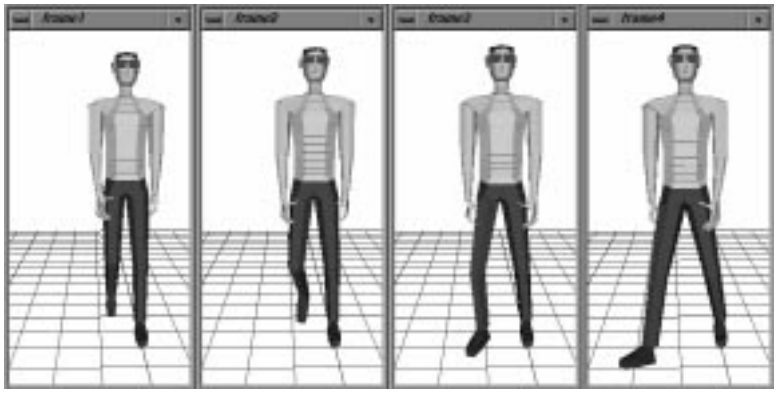

Figure 5.12: Snapshots during a Turning Step.

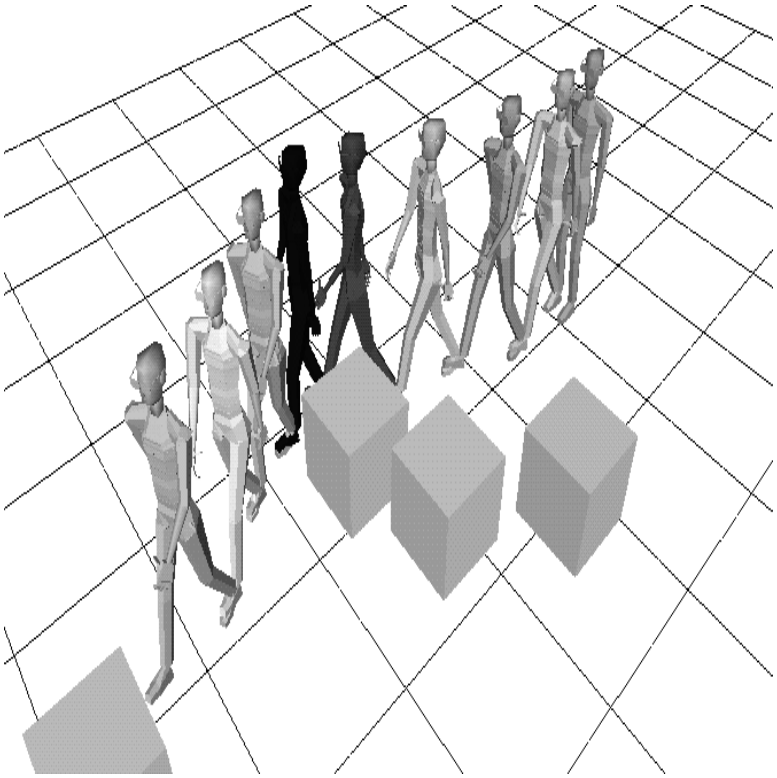

Figure 5.13: Incremental Walk Behavior.

Robotics emphasizes accomplishing a motion within constraints and optimizing it with respect to some criteria such as time, torque, energy, or obstacles [Bob88, HS85b, KN87, Kha87, MK85, SL87]. Bioengineers try to determine if human motion conforms to some optimality criterion, such as time or energy [BC68, CJ71, YN87, Yeo76]. Given its range and diversity, human motion is not optimal with respect to a single criteria.

Despite these various approaches to the human motion problem, none has been successful at specifying a task by describing a load and a placement goal, and then completing the task in a realistic (though possibly suboptimal) manner. There have been efforts to generate a path between two endpoints [AAW74, Ayo91, SH86, KR79, SSSN85], but the usual solution incorporates constraints and a single objective function that is optimized.

5.3.2 Incorporating Strength and Comfort into Motion

We offer a solution which blends kinematic, dynamic and biomechanical information when planning and executing a path. The task is described by the starting position, the load (weight) that needs to be transported, and a goal position for the load. Some simple additional parameters help select from the wide range of possible paths by invoking biomechanical and performance constraints in a natural fashion. Thus a path is determined from a general model rather than provided by default or by an animator. In addition, the algorithm is incremental: it has the ability to adapt to changing forces that are required to complete a task. The basic premise of the method is that a person tends to operate within a *comfort* region which is defined by *available strength*. This is even more probable when the person has to move a heavy object.

We assume that a person tends to operate within a comfort region dictated by muscular strength, especially when moving a heavy object. When a person has to accomplish a given task, say lifting a box or cup, he starts from some initial posture and then plans the direction for his hand to move. This planning is based on the person's perception of his strength, comfort range, and the importance of staying along a particular path. After a direction is determined, he tries to move in that direction for a short distance with joint rates that maintain the body's discomfort level below a particular threshold. Once the position is reached another direction is selected by balancing the need to finish the task as directly as possible with restrictions derived from the body's limitations. Again, joint rates can be determined once a new direction is established.

The objective is to find the trajectories, both joint and end-effector, that a human-like linkage would traverse to complete a lifting task The task can be specified as a force that has to be overcome or imparted to reach a goal position over an entire path. The task specification can be generalized to describe a complicated task by letting the force be a function of body position, hand position, time, or other factors. In general, task specification can be represented by a *force trajectory*. In addition to task specification by a force

trajectory, we have seen other behaviors in which human motion is guided by constraints limiting joint and end-effector trajectories. Constraints that guide the end effector motion are *discomfort level*, *perceived exertion*, and *strength*.

Discomfort level is defined in a mechanical sense. It is found by calculating, over the entire body, the maximum torque ratio: current torque divided by the maximum torque at each individual joint for the current joint position and velocity. The *Comfort level* is just $1 - discomfort$. In general, when humans move they try to maintain their effort below a particular discomfort level. Therefore, it is desirable to dictate a motion that minimizes the maximum torque ratio of a body in order to maximize the comfort level.

Perceived exertion is a variable used to indicate the expected level of difficulty in completing a task. It depends on the perception of the amount of strength required (an implicit function of the force trajectory) and the amount of strength available. If perceived exertion is low then the discomfort level is not expected to be exceeded for the paths "likely" to be taken to satisfy a task, especially for a path that travels a straight line between the initial body position and the goal. However, if the perceived exertion is high, then the end-effector path needs to deviate from a straight path in order to abide by the comfort constraint. Perceived exertion is represented by a cone which is defined by the maximum deviation angle of a path from its current position.

Strength – the maximum achievable joint torque – also dictates end effector motion and path. For testing the strength-guided motion behavior any suitable strength formulation would suffice; we used empirical data collected by Abhilash Pandya of NASA Johnson Space Center [PMA+91]. There are two strength curves to represent the two muscle group strengths (flexor and extensor) at each DOF.

5.3.3 Motion Control

The motion controller consists of three components (Figure 5.14):

1. Condition Monitor which monitors the state of a body and suggests motion strategies.

2. Path Planning Scheme (PPS) which plans the direction that an end-effector will move.

3. Rate Control Process (RCP) which determines the joint rates for motion.

The *condition monitor* reports on the current state of a body: current position, maximum strength for a current position, current joint torques, etc. It then suggests motion strategies to the *path planning scheme* which determines an end-effector's direction of travel. The amount of motion of the end-effector in the suggested direction of travel can be arbitrarily set. The rate of travel, constrained by torque, for a path interval can then be computed by the *rate control process*. After the joint rates are resolved and new joint positions are found, these procedures are repeated until the entire joint path is mapped

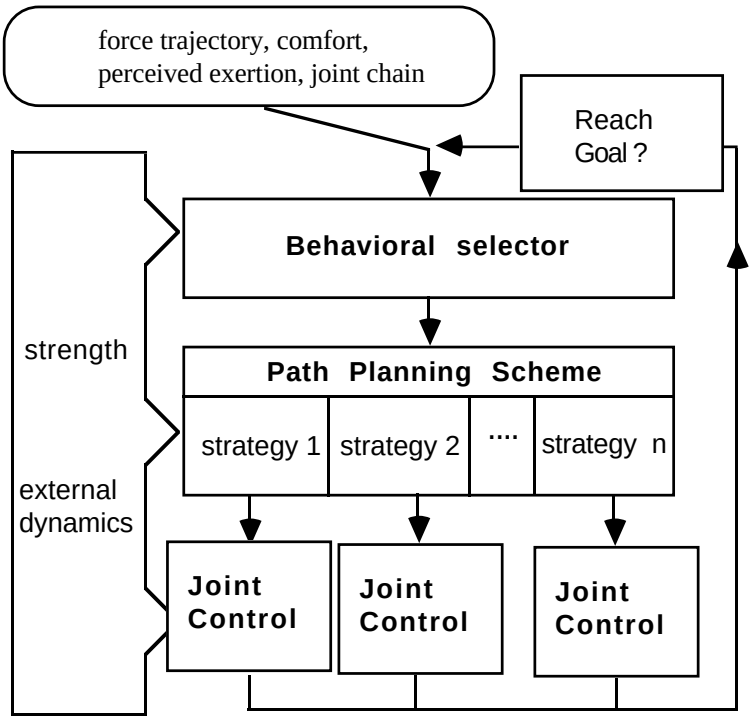

Figure 5.14: Strength Guided Motion Architecture.

out in a manner that satisfies the specified task. This system architecture is an iterative process which allows changes to the parameters at any time through other external processes. Possible situations to alter any of the parameters are dropping or changing the mass of a load, redirecting the goal, or encountering an obstacle. This is different from the global nature of optimal control-based algorithms. We handle similar global considerations through an external process [LWZB90].

Condition Monitor

The condition monitor gathers information about the current state of a body, assembles the information, and suggests a motion strategy for the next procedure to process. The motion strategies are a function of the constraint parameters: comfort, perceived exertion, and strength. Each motion strategy, based on the constraints, concentrates on a separate fundamental aspect of motion. The strategies can be divided into those that represent indirect joint control and those that represent direct joint control. For indirect joint control strategies, the end-effector's need to reach a goal is more important than joint considerations; and for direct joint control, joint considerations are more important than reaching a goal. We can also interpret the strategies as