Terrain Reasoning for Human Locomotion

Barry D. Reich, Hyeongseok Ko, Welton Becket, and Norman I. Badler
Center for Human Modeling and Simulation
University of Pennsylvania
Philadelphia, PA 19104-6389

Abstract

We describe a real-time model of terrain traversal by simulated human agents. Agent navigation includes a variety of simulated sensors, terrain reasoning with behavioral constraints, and detailed simulation of a variety of locomotion techniques. The path through the terrain is incrementally computed by a behavioral reasoning system configuring a behavioral feedback network. A number of sensors acquire information on object range, passageways, obstacles, terrain type, exposure to hostile agents, and so on. The behavioral reasoner weighs this information along with collision avoidance, cost, danger minimization, locomotion types, and other behaviors available to the agent and incrementally attempts to reach a goal location. Since the system is reactive, it can respond to moving obstacles, changing terrain, or unexpected events due to hostile agents or the effects of limited perception. A motor-level component provides general locomotion, including curved path walking and running that work in real-time with human figures.

1 Introduction

Consider the following scenario. An agent begins with an arbitrary position and orientation in an outdoor environment. There is a goal location. The agent moves toward the goal avoiding obstacles, ducking under low branches, climbing over objects, avoiding difficult terrain where feasible, and squeezing through tight spaces where necessary. In some situations the agent tries to avoid the sensory field of one or more hostile agents (hostiles).

Our aim is not to tell the agent how to achieve its goal. Instead, we make the agent aware of its environment and associate a set of actions with different perceptions. An interaction loop is created involving the agent and the environment which converges toward the goal. That is, the agent successfully navigates the terrain and arrives at the goal location.

Simulated sensors are used to detect environmental features such as terrain type, and obstacle and hostile locations. The combined sensory input is analyzed by the behavioral feedback network described in Section 2 and a decision is made as to where the agent should move. This decision is made incrementally, after each step the agent takes, so that a dynamically changing environment can be supported. For example: chasms may open up, trees may fall down, pits or craters appear, or the goal may change.

Once the decision is made where to move next, other sensors are used to examine the local terrain more closely (Section 3). These sensors determine whether or not behavioral modifications must be made and, if so, what the specifics of the modifications are. The agent may have to crawl, crouch, or step over an obstacle.

The behavioral net considers input from all the sensors. It decides where to move the agent, and the appropriate locomotion behavior, described in Section 4, is invoked.

2 Behavioral Control

Much of the planning necessary for terrain traversal can be modeled using *emergent behavior*, where complex observed behavior of the agent is the result of several independently modeled behaviors within the agent interacting with the environment and competing with one another for control of the agent. Agent control relying on emergent behavior has had tremendous success in robotics and artificial intelligence (AI) because it can accomplish essential, low-level agent behavior in complex, dynamic environments robustly and in real time. When used on low-level tasks such as navigation, sensor processing, and motor control, the traditional symbolic reasoning techniques tend to be *brittle* in that they will not apply to unforeseen situations or new environments [9], and non-*real-time*: symbolic reasoning tends to be extremely slow and in fact the general symbolic reasoning problem has been shown to be NP-complete [11]. In robotics and AI, the *emergent behavior* approach has been championed by Brooks [8, 9] and by many others (see Maes' collection of papers [16]). In computer graphics, this technique has been proposed independently as *behavioral control* [19, 18, 21]. In this paper we refer to this general technique as the *behavioral* approach.

The pure behavioral approach, however, focuses on low-level control and does not, in general, exhibit long-range planning or memory. The *behavioral architecture* [2] attempts to consider graphics *and* robotics behavioral approaches within the context of a larger symbolic architecture. This allows integrating the benefits of symbolic reasoning, in particular long-range planning, action sequencing, memory due to internal models of the world, and ease of specification with behavioral approaches. Our approach here will be to use a form of this combined approach along with an explicit internal map of the terrain. In this

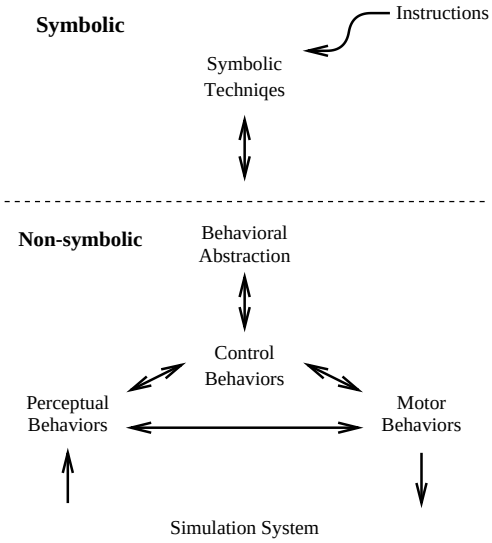

Figure 1: *Behavioral architecture* control hierarchy.

section, we briefly describe the behavioral architecture and the components within it that we use for terrain reasoning.

2.1 The Agent Architecture

The behavioral architecture proposed in [2] is depicted in Fig. 1. There are two primary control paths shown:

the behavioral loop: This is a continuous stream of floating point numbers from the simulated environment, through simulated sensors providing the abstract results of perception, through control behaviors independently attempting to solve a minimization problem, out to simulated effectors or effector behaviors (in our case, walking), which enact changes on the world. This loop continuously operates, connecting sensors to effectors through a network of behavioral nodes which for descriptive convenience are divided into *perceptual behaviors*, *control behaviors*, and *motor behaviors*.

the cognitive pipeline: Symbolic reasoning, or other symbolic or non-behavioral control techniques monitor the behavioral loop and at strategic times, reconfigure the stream of control by altering connectivity, changing connectivity weights, or modifying node parameters. The *behavioral abstractions* manage the transition from floating point signals to discrete symbols and also package behavior into dependable units. This split allows addressing the *symbol grounding problem* (see [15]) and keeps symbolic operations at a sufficiently abstract level.

2.2 Behavioral Nets

The *behavioral loop* is modeled as a network of interacting behaviors, where perceptual, control, and motor behaviors are nodes connected by arcs across which only floating point messages travel. An individual, conceptual path from sensors to effectors is referred to as a *behavioral net*, and is analogous to a complete behavior in the traditional emergent behavior architectures such as Brooks' *subsumption architecture* [8], except that nodes may be shared between behaviors and arbitration (competition for effector resources) may occur throughout the behavioral path and not just at the end-effector level. The behavioral loop is modeled as a network with floating point connections in order to allow the application of low-level, unsupervised, reinforcement learning in the behavioral design process (this is being developed in [3]).

2.2.1 Perceptual behaviors

The perceptual behaviors used for our terrain reasoning model the abstract, geometric results of object perception. They continuously generate signals describing the polar coordinate position (relative to the agent) of a particular object or of all objects of a certain type within a specified distance and field of view. Two of the perceptual behaviors used are:

object sensors: These provide the current distance from the agent and angle relative to the forward axis of the agent of a particular object in the environment. Note that this sensor directly abstracts over object recognition (which we believe is a fair assumption for a simulated human agent in our circumstance).

range sensors: Collects all objects of a certain type within a given range and field of view, and performs a weighted average into signals giving the distance and angle of a single abstract object representing all detected objects. Signals into the sensor define the range, field of view, and weighting parameters (defining relative weights of distance and angle) and may be altered continuously in order to focus the sensor.

Another perceptual behavior is used that perceives an internal map of the terrain as if it were an external entity – this is discussed in detail in Section 3.

These geometric sensors are sufficient for the current abstraction level of our work. However, a more sophisticated approach is to simulate the high-level results of vision of the agent using Z-buffering hardware to create a depth map of what the agent can see. This is the approach used by Renault and Thalmann [18] and Reynolds [20], and we intend to incorporate this approach as our terrain reasoning model progresses.

2.2.2 Control behaviors

For terrain reasoning we use two simple control behaviors loosely based on Braitenberg's *love* and *hate* behaviors [7], but formulated as explicit minimization nodes using outputs to drive inputs to a desired value (similar to Wilhelms' [21] use of Braitenberg's behaviors). Control behaviors typically receive input signals

directly from perceptual behaviors, and send outputs directly to motor behaviors, though they could be used in more abstract control situations. Our two control behaviors are:

attract: Create an output signal in the direction of the input signal, but magnified according to distance and angle scalar multipliers and exponents. This behavior works only when input signals exceed a threshold distance or angle.

avoid: Create an output signal in the opposite direction of the input, magnified according to scalar multipliers and exponents, whenever inputs fall below a threshold distance or angle.

These behaviors incorporate both scalar multipliers and exponents, to allow modeling the non-linearities typically observed in animal responses to perceived inputs [19].

2.2.3 Motor behaviors

Motor behaviors connect to the underlying human body model and directly execute routines defined on the model (such as walking, balance, hand position, and torso orientation) and arbitrate among inputs, either by selecting one set of incoming signals or averaging all incoming signals. An example is the *walk controller*, which decides where to place the agent's next footstep and then connects to the locomotion generator discussed in Section 4 to achieve the step.

A human steps at discrete positions in a continuous space. We assume that the agent cannot change the targeted step location while a step is in progress. In order to solve sampling problems that may occur by only sampling at the agent's center at the beginning of every step (especially due to thresholds on avoidance behaviors), the walk controller interprets the incoming attract and avoid signals as a potential field, then steps at a position within the range of possible foot positions with minimum field strength. The field strength for a possible location of the agent is constructed as the sum of absolute values of all angle and distance inputs – which provides a stress value for the position. Discrete positions along a predefined number of arcs in front of the agent are sampled in order to find a suitable minimum position.

2.3 Parallel Automata: PaT-Nets

Although the *behavioral architecture* outlined above is designed to connect to a general symbolic reasoning process, we currently control the behavioral pipeline with a model of parallel automata called *Parallel Transition Nets* (PaT-Nets) [4]. A sample PaT-Net is shown conceptually in Fig. 2. Each net description is a class in the object-oriented sense and contains a number of nodes connected by arcs. Nodes contain arbitrary lisp expressions to execute as an *action* whenever the node is entered. A transition is made to a new node by selecting the first arc with a true condition (defined as a lisp expression). Nodes may also support probabilistic transitions where the probability of a transition along an arc is defined rather than

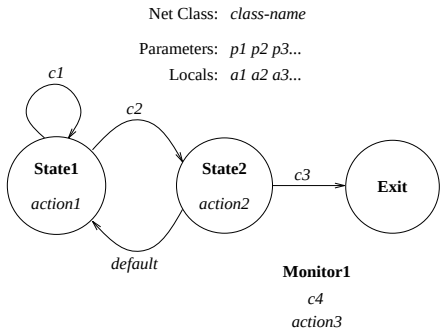

Figure 2: A sample PaT-Net shown graphically

a condition. *Monitors* are supported that, regardless of which state the net is in, will execute an action if a general condition evaluates to `true`.

A running network is created by making an instance of the PaT-Net class. Because a running net is actually an encapsulated, persistent object, it may have local state variables available to all actions and conditions, and may also take parameters on instantiation. The running net is time-sliced within the behavioral pipeline. It runs concurrently and constantly monitors the flow of numbers to watch for conditions that will trigger a change of state and possibly a reconfiguration or modification of the pipeline.

The running PaT-Net instances are embedded in a lisp operating system that time-slices them into the overall simulation. This operating system allows PaT-Nets to spawn new nets, kill other running nets, communicate through semaphores and priority queues and wait (sleep) until a condition is met (such as waiting for another net to exit, for specific time in the simulation, or for a resource to be free). Running nets can, for example, spawn new nets and then wait for them to exit (effectively a subroutine call), or run in parallel with the new net, communicating if necessary through semaphores.

Because PaT-Nets are embedded in an object-oriented structure, new nets can be defined that override, blend, or extend the functionality of existing nets.

3 Sensor-Based Navigation

In order to make intelligent decisions based on the local environment and terrain it is necessary to represent the world on a human scale. A standard must be adopted. Microterrain is too coarse a model for this purpose. The standard chosen is based on a regular grid of squares, one meter on a side. We call this *nanoterrain.*

Each nanoterrain grid element includes the terrain type such as water or grass. Only the terrain is gridded. There are no restrictions on the size, shape, position, or orientation of objects or obstacles in the environment.

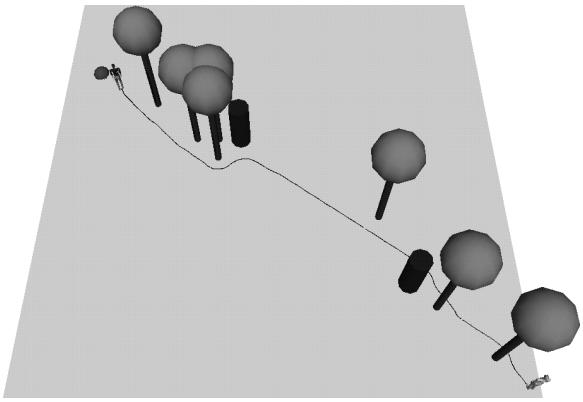

Figure 3: An agent avoiding obstacles

3.1 Sensors

The agent utilizes a set of sensors in order to interact with its environment. These sensors can be divided into several classes:

- Attractors
- Repulsers
- Terrain Sensors
- Hostile Agents' Sensory-Field Sensors

3.1.1 Attractors

An attractor draw the agent toward an object. The primary use for this type of sensor is to attract the agent to the goal. Another possible use is to attract the agent to objects to hide behind such as walls. This is important in some simulations where the agent is trying not to be seen. The force this sensor exerts must be strictly less than the force the goal sensor exerts at all times. Otherwise the agent may get stuck in a local minimum near large structures. In a multi-agent simulation, attractors can also be used to keep the agents together in a group.

3.1.2 Repulsers

A repulser exerts a force away from an object. They are particularly useful for collision avoidance. Trees, for example, are detected by repulsers with an infinitely high avoidance force inside, and zero avoidance force beyond a threshold equal to the radius of the tree. This allows the agent to come close to a tree in a simulation, but does not allow the agent to step inside one. In a multi-agent simulation, repulsers are also used to prevent agents from colliding with each other.

Fig. 3 is a simple example. The agent begins in the lower right corner. It is fitted with a sensor which attracts it to the goal in the upper left corner. It also senses trees and obstacles and avoids them. The agent walked in real-time towards the goal along the path shown.

3.1.3 Terrain sensor

The terrain sensor evaluates a position and orientation in the environment based on the local terrain-type. Each nanoterrain grid element has an associated weight. Easily navigable terrain such as grass is given a small weight. Regions that are difficult to traverse are given a high weight.

The terrain sensor does more than simply exert a force proportional to this weight. Instead it considers several steps in the direction the agent is facing. The weights for each of these steps are multiplied by a function that drops with distance and summed. The exerted force is proportional to that result. The advantage of this method is that the agent will tend to step into a region of difficult terrain if the region beyond is clear. For example, although the agent tends to avoid water, it will cross a stream if there is a grassy field on the other side.

3.1.4 Hostiles' sensory-field sensor

The agent attempts to achieve the goal location while avoiding the gaze of hostiles. Hostiles' views may be obscured by obstacles, structures, and terrain, and lessened by distance and atmospheric effects. A sensor is used to detect regions visible to the hostile agents and avoid them if possible. This avoidance desire is stronger when the region is closer to a hostile or when it is visible to more than one hostile.

Another consideration is that, although the agent may be standing in a region considered safe, the agent's head may be visible to one or more hostiles. If this is the case a sensor will detect the condition and the agent will crouch or drop to the ground and crawl.

3.2 Behavioral modifications

Sometimes a path which encounters obstacles is chosen for the agent. When this is the case the agent is forced to make behavioral changes to successfully navigate through the region.

3.2.1 Low obstacles

When the agent encounters low obstacles such as boulders or fallen trees one of three things happen. The agent climbs or jumps over or steps on the obstacle depending on its size and the agent's current behavior. The agent will not jump unless it is currently running.

3.2.2 High obstacles

In the case of a high obstacle such as a tree branch or a door frame the choices are to duck or crawl. The choice is made based on the height of the obstacle.

3.2.3 Narrow widths

When the agent encounters a narrow width it will squeeze through it. This may require turning sideways.

3.3 PaT-Nets for decision making

PaT-Nets are a mechanism for introducing decision-making into the agent architecture. They monitor the behavioral loop (which may be thought of as modeling instinctive or reflexive behavior) and make decisions in special circumstances. For example, the behavior resulting from the combined use of different types of sensors can sometimes break down. The agent may get caught in a dead-end or other local minimum. PaT-Nets recognize these situations, override the "instinctive" behavioral simulation by reconfiguring connectivity and modifying weights, and then return to a monitoring state.

4 Motor Level Locomotion Generation

Even though locomotion is a subconscious activity in everyday life, generating realistic animation of it is a difficult task. General techniques such as key framing produce very specific, non-generalizable results [12]. This task is complicated by requiring generation of everyday variations on normal rhythmic forward walking: curved path walking, lateral stepping, backward stepping, turning around, ducking, running, stepping over, running to walking and walking to running transitions, and so on. Efficiency of the algorithm is important for real-time simulation. The problem posed at this level is real-time generation of all realistic variations of human locomotion.

4.1 Parametric Locomotion

We achieve all these locomotion patterns with three different parametric locomotion types. *Rhythmic walking* generates all periodic forward curved walks including the beginning and ending steps. *Non-rhythmic stepping* is used for non-periodic steps in arbitrary directions such as lateral, backward, direction reversal, etc. The *running* type is used for curved path runs and transitions between running and walking.

The walking pattern should be modified to reflect environmental constraints. For example, crouched walking might be required to avoid possible obstacles or to avoid being seen. This means that motor level locomotion should provide controls for the pattern of stepping as well as for advancing the required distance. So in each locomotion type, there are primary and attribute parameters that the higher level component may specify. Primary parameters *must* be specified to generate a step: they provide the position of the next heel strike, the direction of the next foot, and a designation of the stepping foot (left or right). For the attribute parameters, default values are used to produce a normal motion. Values may also be provided from the higher level components to achieve a specific goal. Attribute parameters consist of the swing ankle height at its apex, the pelvis height at its apex, and other rotational and translational attributes of the torso and pelvis (Section 4.2). For example, setting the swing ankle height to a greater value will allow the agent to step over a shallow obstacle.

Each step is initialized before it begins: **initialize-step**(walker, locomotion-type, left-or-right, position, direction, attributes). Then **advance-step**(walker, normalized time) actually generates the stepping motion. Normalized time reflects logical time to generate the whole step. Thus **advance-step**(walker, 0) and **advance-step**(walker, 1) will produce the poses just before and after the current step starts and ends, respectively. By controlling Δt we can adjust the number of frames to be generated for a step depending on the machine speed.

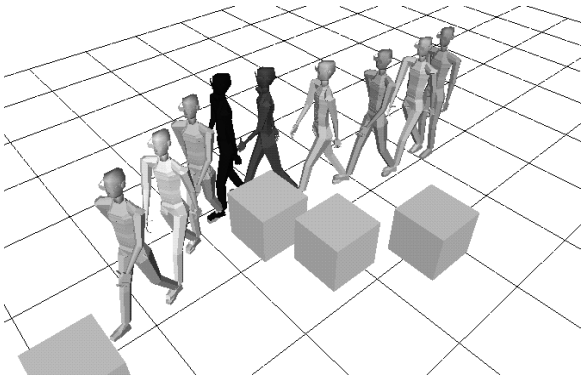

Figure 4: A time-lapse image of curved walking

For the walking animation, a biomechanical measurement of straight-line walking is generalized to the given body size and adjusted for any path curvature, turning, or steps [14, 13, 1, 5, 6]. The motion characteristics of the original gait are preserved during the generalization even if the step direction or step length is changed [14]. Therefore several different walking styles can be adopted merely by acquiring multiple sets of measurements. Fig. 4 shows a few frames from a curved path walk.

Running is produced by a similar algorithm except that no time exists during which both feet touch the ground. The transitions between walking and running must be considered for continuous motion flow, and are produced by mixing the walking and running proportionally during the transition interval. For example, the beginning of the walking-to-running transition is the same as walking, and as the normalized time flows, the weight of the running part increases until it is one at the end of the transition. Fig. 5 shows a running step over a fallen tree.

4.2 Attributes

Torso flexion and pelvic rotation have been parameterized to generate different styles of walking [10]. The torso can be flexed or twisted rhythmically in any (legal) direction. The pelvis position and orientation can be controlled relative to the normal walking over the walking cycle. Combining both the torso flexion and pelvic rotation produces stylistic changes, either major or minor, in walking. For example, crouched walking is obtained by bending the torso forward and lowering and rotating the hips.

Twelve parameters are used to specify pelvic rotation during locomotion. Even though actual pelvis rotation does not exactly follow the sinusoidal function, sine and cosine curves provide visually acceptable flex-

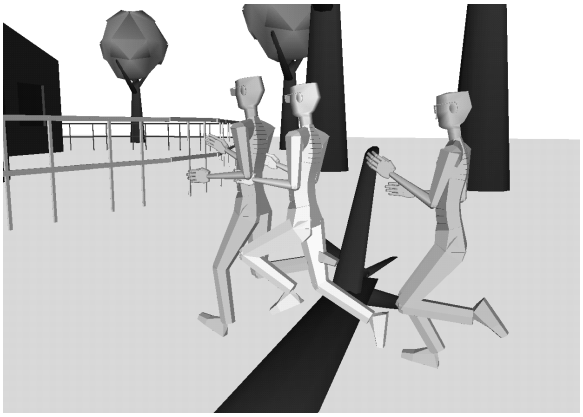

Figure 5: A time-lapse image of running

ibility. The 12 parameters (amplitudes and displacements of the sinusoidal functions) are: A_θ^x, D_θ^x, A_θ^y, D_θ^y, A_θ^z, D_θ^z, A_p^x, D_p^x, A_p^y, D_p^y, A_p^z, D_p^z. Our convention is that the x axis is forward, the y axis is lateral (right), and the z axis is down. With these parameters the actual pelvis rotation around the x axis at the normalized time t is given as

$$\theta_x(t) = A_\theta^x sin(2\pi t) + D_\theta^x. \quad (1)$$

The other angles or positions are described similarly except for the kind of sinusoidal function and its sign.

The torso flexion and twist is given by a sinusoidal function similar to Equation 1, with the amplitudes and displacements $A_{\theta'}^x$, $D_{\theta'}^x$, $A_{\theta'}^y$, $D_{\theta'}^y$, $A_{\theta'}^z$, $D_{\theta'}^z$.

Thus motor level locomotion provides twenty (twelve for the pelvis, six for the torso, swing ankle height, hip height) attributes for higher level control.

If a few steps ahead of the current step are specified, the locomotion algorithm can actually generate a more natural walking motion by twisting the torso and neck to maintain anticipated eye gaze towards the future path. With the specification of only the next step – as in this real-time, reactive application – the anticipation cannot be done correctly. Thus the walker appears to look somewhat aloof during real-time simulation; however, more generalized focus of attention can correct this.

5 Discussion

The current trend in designing autonomous agents is away from the *deliberative thinking* paradigm and toward a more direct coupling of perception to action. Flexibility, distributedness, parallelization, and dynamic interaction with the environment are emphasized more and more [17]. Our approach exhibits these features.

Sensor-based navigation as a method of path planning has several desirable properties. The iterative approach allows for a dynamically changing environment which includes changing terrain, obstacle, and goal positions. The fact that only a few local decisions need to be made for each step make the algorithm fast and implementable in real-time. Moreover, it is versatile. New types of sensors may be designed and easily integrated; additional layers of behavioral control may be overlaid on the architecture.

Acknowledgments

This research is partially supported by ARO DAAL03-89-C-0031 including U.S. Army Research Laboratory (Aberdeen); U.S. Air Force DEPTH through Hughes Missile Systems F33615-91-C-0001; Naval Training Systems Center N61339-93-M-0843; Sandia Labs AG-6076; DMSO through the University of Iowa; NASA KSC NAG10-0122; MOCO, Inc.; Robotics Research Harvesting, Inc.; NSF IRI91-17110, CISE CDA88-22719, and Instrumentation and Laboratory Improvement Program #USE-9152503. We wish to thank Brett Douville for comments and contributions to editing.

References

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

[2] Welton Becket and Norman I. Badler. Integrated behavioral agent architecture. In *The Third Conference on Computer Generated Forces and Behavior Representation*, Orlando, Florida, March 1993.

[3] Welton M. Becket. PhD thesis, University of Pennsylvania, 1994. In preparation.

[4] Welton M. Becket. The *Jack* lisp api. Technical Report MS-CIS-94-01/Graphics Lab 59, University of Pennsylvania, Philadelphia, PA 19104-6389, 1994.

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

[6] Ronan Boulic and Daniel Thalmann. Combined direct and inverse kinematic control for articulated figure motion editing. *Computer Graphics Forum*, 11(4):189–202, 1992.

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

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

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

[10] Armin Bruderlin and Tom Calvert. Interactive animation of personalized human locomotion. In *Proceedings of Graphics Interface '93*, pages 17–23, Toronto, Canada, May 1993.

[11] David Chapman. Planning for conjunctive goals. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 537–558. Morgon Kaufmann Publishers, Inc., 1990.

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

[13] Hyeongseok Ko and Norman I. Badler. Intermittent non-rhythmic human stepping and locomotion. In *Proceedings of the First Pacific Conference on Computer Graphics and Applications, Pacific Graphics '93*, pages 283–300, Seoul, August 1993. World Scientific.

[14] Hyeongseok Ko and Norman I. Badler. Straight line walking animation based on kinematic generalization that preserves the original characteristics. In *Graphics Interface '93*, Toronto, Canada, May 1993.

[15] Libby Levison. PhD thesis, University of Pennsylvania, 1994. In preparation.

[16] Pattie Maes, editor. *Designing Autonomous Agents*. MIT Press, 1990.

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

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

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

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

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