IN THE PROCEEDINGS OF CAPTECH'98, NOVEMBER 1998

Goal-Directed Navigation for Animated Characters Using Real-Time Path Planning and Control

James J. Kuffner, Jr

Computer Science Robotics Lab, Stanford University
Stanford, CA 94305-9010, USA,
kuffner@stanford.edu,
http://robotics.stanford.edu/~kuffner/

Abstract. This paper presents a new technique for computing collision-free navigation motions from task-level commands for animated human characters in interactive virtual environments. The algorithm implementation utilizes the hardware rendering pipeline commonly found on graphics accelerator cards to perform fast 2D motion planning. Given a 3D geometric description of an animated character and a level-terrain environment, collision-free navigation paths can be computed between initial and goal locations at interactive rates. Speed is gained by leveraging the graphics hardware to quickly project the obstacle geometry into a 2D bitmap for planning. The bitmap may be searched by any number of standard dynamic programming techniques to produce a final path. Cyclic motion capture data is used along with a simple proportional derivative controller to animate the character as it follows the computed path. The technique has been implemented on an SGI Indigo2 workstation and runs at interactive rates. It allows for real-time modification of the goal locations and obstacle positions for multiple characters in complex environments composed of more than 15,000 triangles.

1 Introduction

Advances in computing hardware, software, and network technology have enabled a new class of interactive applications involving 3D animated characters to become increasingly feasible. Many such applications require algorithms that enable animated characters to move naturally and realistically in response to task-level commands. This paper addresses the problem of quickly synthesizing from navigation goals the collision-free motions for animated human figures in changing virtual environments. The method presented here combines a path planner, a path-following controller, and cyclic motion capture data. The resulting animation can be generated at interactive rates and looks fairly realistic. This work is part of a larger project to build autonomous animated characters equipped with motion planning capabilities and simulated sensing[1]. The ultimate goal of this research is to create animated agents able to respond to task-level commands and behave naturally within changing virtual environments.

A prolific area of research in the robotics literature has been the design and implementation of task-level motion planning algorithms for real-world robotic systems[2]. The inspiration for the fundamental ideas in this paper arises from this research. Section 2 provides a background and motivation for task-level control in the context of animation. Related work in building autonomous agents for the purposes of graphic animation is summarized in Section 3. Section 4 describes an approach for computing goal-directed navigation motions for animated human figures. Section 5 gives an overview of the algorithm, while Section 6 and Section 7 describe in greater detail the path planning and path following phases of the approach respectively. In Section 8 the current implementation and performance results are presented. Section 9 concludes with a discussion of the limitations and possible extensions to the ideas presented here.

2 Motivation

The primary motivation for task-level control in animation stems from the time and labor required to specify motion trajectories for complex multi-jointed characters, such as human figures. Traditional keyframe animation techniques are extremely labor-intensive and often yield motion that looks unrealistic or is physically invalid. Motion capture techniques offer a simple alternative for obtaining realistic-looking motion. Unfortunately, both keyframed-motion and motion capture data alone are inflexible in the sense that the motion is often only valid for a limited set of situations. Frequently, such motions must be redesigned if the locations of other objects or starting conditions change even slightly. Motion warping or blending algorithms[3, 4] offer some added flexibility, but usually can only be applied to a limited set of situations involving minor changes to the environment or starting conditions. Significant changes typically result in unrealistic motions.

Dynamic simulation and physically-based modeling techniques nicely handle the problems of physical validity and applicability to arbitrary situations. Given initial positions, velocities, forces, and dynamic properties, an object's motion is simulated according to natural physical laws[5, 6]. However, aside from specifying initial conditions, the user has no control over both the resulting motion and the final resting position of the object. Spacetime constraints provide a more general mathematical framework for addressing this problem of control [7, 8, 9]. Constraint equations imposed by the initial and final conditions, obstacle boundaries, and other desired properties of the motion are solved numerically. Unfortunately, the large number of constraints imposed by complex obstacle-cluttered environments can severely degrade the performance of such methods.

New approaches and algorithms are needed to compute natural, collision-free motions quickly in changing virtual environments. The method described in this paper combines a fast 2D path planner along with a proportional derivative (PD) controller to compute natural-looking motions for navigation tasks. The controller is used to synthesize cyclic motion capture data for an animated character as it follows a computed path towards a goal location. The goal location can be can be user-specified or defined by a behavior script. The implemented planner executes in roughly one-tenth of one second on average, thus allowing real-time modification of the goal location or obstacle positions.

3 Related Work

Previous work in motion synthesis for animated characters has traditionally been divided between real-time applications and off-line animation production. However, as processor speeds continue to increase, algorithms originally intended for off-line animations will gradually become feasible in real-time virtual environments.

Much research effort in robotics has been focused on designing control architectures for autonomous agents that operate in the real world[10, 11]. Using rasterizing computer graphics hardware to assist robot motion planning algorithms was previously investigated by Lengyel, *et al*[12]. Recently, motion planning tools and algorithms have been applied to character animation. Koga *et al.* combined motion planning and human arm inverse kinematics algorithms for automatically generating animation for human arm manipulation tasks[13]. Hsu and Cohen combined path planning with motion capture data to animate a human figure navigating on uneven terrain [14]. Researchers at the University of Pennsylvania have been exploring the use of motion planning to achieve postural goals using their Jack human character model[15, 16], incorporating body dynamics[17], and high-level scripting[18].

Research in designing fully-autonomous, interactive, artificial agents has also been on the rise. Tu and Terzopoulos implemented a realistic simulation of autonomous artificial fishes, complete with integrated simple behaviors, physically-based motion generation, and simulated perception[19]. Noser, *et al.* proposed a navigation system for animated characters based on synthetic vision, memory and learning[20]. Other systems include Perlin and Goldberg's Improv software for interactive agents[21, 22], the ALIVE project at MIT[23, 24], Johnson's WavesWorld, and perhaps one of the earliest attempts at creating an agent architecture for the purposes of graphic animation: the Oz project at CMU[25]. The goals of the Oz project were to create agents with "broad" but "shallow" capabilities, rather than "deep" capabilities in a narrow area. Researchers at Georgia Tech have combined physically-based simulation with group behaviors for simulating human athletics[26]. They have also designed a controller for human running in 3D[27]. Despite these achievements, building autonomous agents that respond intelligently to task-level commands remains an elusive goal, particularly in real-time applications.

4 Goal-Directed Navigation

Consider the case of an animated human character given the task of moving from one location to another in a flat-terrain virtual environment. One would like to produce a reasonable set of motions to accomplish the task while avoiding obstacles and other characters in the environment. Our strategy will be to divide the computation into two phases: *path planning* and *path following*. The planning phase computes a collision-free path to the goal location using the graphics hardware for speed, while the path following phase uses a proportional derivative (PD) controller to guide the character's motion along the path. The path planning phase should run very quickly, since the locations of other characters or objects in the environment may change without warning. Such changes may invalidate the collision-free nature of the current path being followed, and necessitate re-planning. The controller should be fast and flexible enough to allow the current path being followed to be replaced without warning, and still generate smooth motions.

5 Algorithm Overview

Initialization: A general 3D description of the environment and characters suitable for rendering is provided, along with a goal location. Motion capture data for a single cycle of a locomotion gait along a straight line is pre-processed as described in Section 7.2.

Projection: The planning phase begins by performing an off-screen projection of the environment geometry into a 2D bitmap. An orthographic camera is positioned above the environment, pointing down along the negative normal direction of the walking surface. The *near* clipping plane of the camera is set to correspond to the maximum vertical extent of the character's geometry, and the *far* clipping plane is set to be slightly above the walking surface. All geometry within the given height range is projected (rendered) to either the back buffer or an offscreen bitmap via the graphics hardware.

Path Search: The bitmap from the previous step is searched directly for a collision-free path using any standard dynamic programming technique. Examples include Dijkstra's algorithm, or A* with some appropriate heuristic function. If planning is successful, the complete path is sent to the path following phase. Otherwise, the controller is notified that no path exists.

Path Following: Cyclic motion capture data along with a PD controller on the position and velocity is used to generate the final motion for the character as it tracks the computed path. If no path exists, an appropriate stopping motion or waiting behavior is performed.

6 Path Planning

The theory and analysis of motion planning algorithms is fairly well-developed in the robotics literature, and is not discussed in detail here. For a broad background in motion planning, readers are referred to [2]. For any motion planning, it is important to minimize the number of degrees of freedom (DOFs), since the time complexity of known algorithms grows exponentially with the number of

DOFs[28]. For character navigation on level-terrain, the important DOFs are the position and orientation (x, y, θ) of the base of the character on the walking surface. As detailed in Section 7, the orientation (forward-facing direction) of the character is computed by the controller during the path following phase. Thus, we need only to consider the position (x, y) of the base of the character during the planning phase. Bounding the character's geometry by a cylinder allows motion planning for level-terrain navigation to be reduced to planning collision-free trajectories for a circular disk in 2D. Figure 1 shows one of the characters used in our experiments along with the computed bounding cylinder.

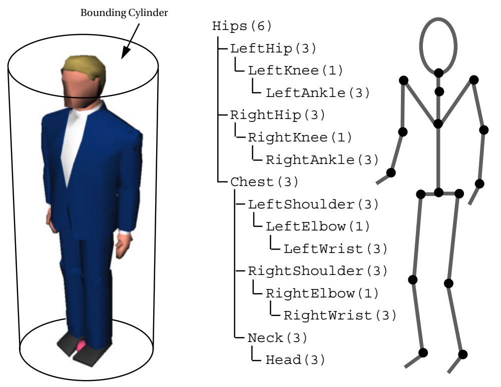

Fig. 1. The character's geometry is bounded by an appropriate cylinder. This effectively reduces the navigation problem to one involving motion planning for a circular disc among obstacles in 2D. The character's joint hierarchy is shown, along with the number of DOF for each joint.

The path planning approach adopted in this paper is one instance of an *approximate cell decomposition* method[2]. The search space (in this case, the walking surface) is discretized into a fine regular grid of cells. All obstacles are projected onto the grid and "grown" as detailed in Section 6.1. Hence, the grid approximately captures the free regions of the search space at the given grid resolution, and can ultimately be used for fast collision checking.

6.1 Obstacle Projection

All obstacle geometry within the character's height range $[z_{min}, z_{max}]$ is projected orthographically onto the grid. The height range limitation assures that only obstacle geometry that truly impedes the motion of the character is projected. Cells in the grid are marked as either *FREE* or *NOT-FREE*, depending upon whether or not the cell contains any obstacle geometry. The resulting 2D bitmap $\mathcal{B}$ now represents an occupancy grid of all obstacles within the character's height range projected at their current locations onto the walking surface.

Cells in $\mathcal{B}$ marked as *NOT-FREE* are "grown" by R, the radius of a cylinder that conservatively bounds the character's geometry. This is done by marking all neighboring cells within a circle of radius R from each original *NOT-FREE* cell as *NOT-FREE*. In effect, this operation computes the Minkowski difference of the projected obstacles and the character's bounding cylinder, thus reducing the problem of planning for a circular disc, into planning for a point object. To check whether or not a disc whose center is located at (x, y) intersects any obstacles, we can simply test whether $\mathcal{B}(x, y)$, the cell in $\mathcal{B}$ containing the point (x, y) is marked *FREE*.

For increased speed, the obstacle projection operation is performed using rendering hardware. Here, the rendering pipeline enables us to quickly generate the bitmap needed for fast point collision checking. An orthographic camera is positioned above the scene pointing down along the negative normal direction of the walking surface with the clipping planes set to the vertical extents of the character's geometry. The other dimensions of the orthographic view volume are set to enclose the furthest extents of the walking surface as depicted in Figure 2. All geometry within the view volume is projected (rendered) into either the back buffer or an offscreen bitmap via the graphics hardware. Since we are only concerned with whether or not any obstacle geometry maps to each pixel location, we can effectively ignore the pixel color and depth value. Thus, we can turn off all lighting effects and depth comparisons, and simply render in wireframe all obstacle geometry in a uniform color against a default background color. Under most reasonable graphics hardware systems, this will significantly speed up rendering. Furthermore, if the graphics hardware supports variable line-widths and point-sizes, we can perform the obstacle growth at no additional computational cost! We simply set the line-width and point-size rendering style to correspond to the projected pixel length of R, the radius of the character's bounding cylinder. In this way, we can efficiently perform both obstacle projection and growth simultaneously.

6.2 Path Search

The bitmap $\mathcal{B}$ is essentially an approximate map of the occupied and free regions in the environment. Assume that the goal location $\mathcal{G} = (g_x, g_y)$ and the starting location $\mathcal{S} = (s_x, s_y)$ in the bitmap are both *FREE*. Let us consider $\mathcal{B}$ as an embedded graph of cells, each connected to its neighboring cells. By searching $\mathcal{B}$, we can conservatively determine whether or not a collision-free path exists from the start location to the goal location at the current grid resolution. Moreover, if we assign a cost to each arc between cells, we can search for a path that connects $\mathcal{S}$ and $\mathcal{G}$ while minimizing our cost function. Here, we use a relative measure of the Euclidean distance between adjacent cells as our cost. Arcs between directly adjacent cells are assigned a cost of 1, while diagonally-adjacent cells are assigned a relative cost of 1.4. Our task has been effectively reduced to that of searching for a path between two nodes in a weighted graph.

Any number of standard dynamic programming techniques may be used to search $\mathcal{B}$. For simplicity, Dijkstra's algorithm is used in the implementation

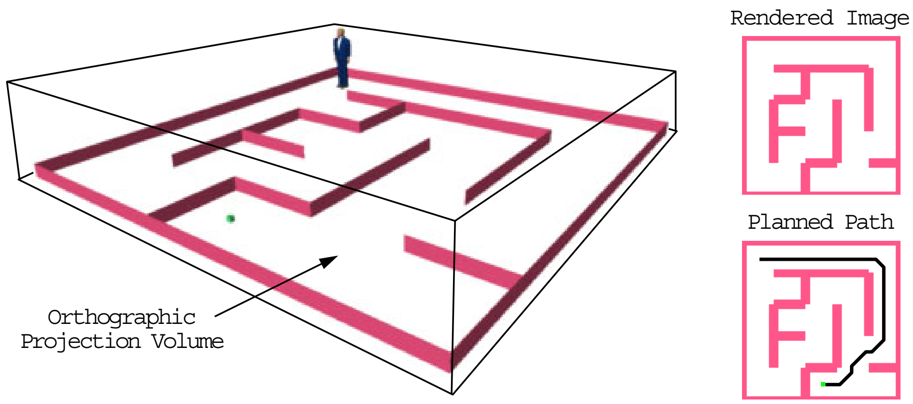

Fig. 2. The view volume resulting from an overhead orthographic camera is outlined. The near and far clipping planes of the camera are defined according to the vertical extents of the character geometry. All obstacle geometry contained within the view volume is projected to 2D via an off-screen renderer. The resulting bitmap is searched directly for a collision-free path connecting the character's starting position to the goal.

described here. This search strategy will always return a path containing a list of *FREE* cells between $\mathcal{S}$ and $\mathcal{G}$ if one exists at the given grid resolution. Moreover, since a relative measure of the Euclidean distance is used as the single-step cost between cells during the search, the returned path will be of minimal length (for the given grid resolution).[1] From the list of free cells connecting $\mathcal{S}$ to $\mathcal{G}$, a final path $\mathcal{P}$ is constructed by linking in sequence the line segments connecting the centers of adjacent cells along the path.

6.3 Complexity Analysis

If planning is successful, the complete path is sent to the path following controller. Otherwise, the controller is notified that no path exists. The planner is *resolution-complete*, meaning that it is guaranteed to find a collision-free path from $\mathcal{S}$ to $\mathcal{G}$ if one exists at the current grid resolution, and otherwise report failure[2].

The running time of the obstacle projection step is proportional to the number and geometric complexity of the obstacles. Searching for a path in the bitmap using Dijkstra's algorithm runs in quadratic time with respect to the number of free cells in the grid. Extracting the path (if one exists) runs in time proportional to the length of the path. Overall, this planning strategy can be implemented very efficiently and robustly even for complex environments. It is interesting to note that paradoxically, the search phase of the planner may run *faster* for com-

[1] In this implementation, fixed-point math is used for the distance and cost computations, resulting in a planner that runs almost entirely using fast integer arithmetic.

plex, obstacle-cluttered environments, since such environments result in fewer free cells to search. Detailed performance results are given in Section 8.

7 Path Following

Simply computing a collision free path in the environment is not enough to produce realistic animation. The implementation described here uses cyclic motion capture data applied to the joints of the character, plus a simple low-level proportional derivative controller to follow the computed path. The quality of the final motion arises primarily from the motion capture data, but there are other alternatives that could be used for path following. So-called "footstep"-driven animation systems could be applied to place the feet at points nearby the computed path, along with real-time inverse kinematics (IK) to hold them in place. As computer processing power increases, physically-based models of the character dynamics along with complex controllers such as the one presented in [27] could also potentially be used to simulate locomotion gaits along the path. For the purposes of these experiments, applying cyclic motion capture data proved to be a fast and simple method of obtaining satisfactory motion.

Although the path planning and following concept generally applies to many types of characters and motions, we will concentrate on generating walking or running motions for a human-like biped. We would like the character's motion to be smooth and continuous, natural-looking, and follow the computed path as closely as possible. Though many kinds of path following techniques could potentially be used, the one described here was chosen for its simplicity and efficiency.

7.1 Mathematical Model

Human figure walking or running is essentially a quasi-nonholonomic system, since the typical turning radius is usually subject to some minimum radius depending upon the velocity. Of course, a person can turn in place, but typically, this will only happen at the beginning or end of a path following procedure, not in the middle of a path. Humans tend to only walk forward, not backward or sideways (no direction reversals during path following).

With this in mind, the path following phase is modeled as one involving an oriented disc smoothly tracking a geometric path in the plane. The disc center corresponds to the projected point at the base of the character's geometry, and the orientation of the disc corresponds to the character's forward-facing direction as illustrated in Figure 3. The linear velocity of the disc is constrained to always lie along the forward-facing direction. This corresponds to the character's ability to walk or run forward. Turning is modeled by considering the disc's rotational velocity about its center.

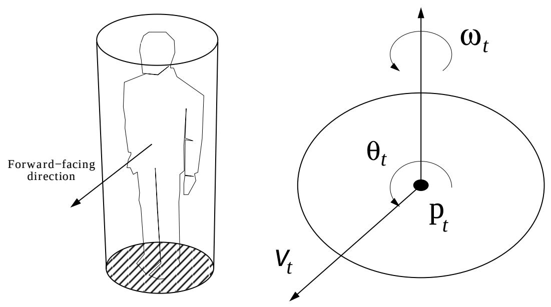

Fig. 3. The controller model considers the character's motion as that of an oriented disc in the plane. The center of the disc corresponds to the projection of the origin of the root joint of the figure onto the walking surface.

A discrete time simulation of the following state variables is used:

$\mathbf{p}_t$	position (x_t,y_t) if the disc center
θ_t	orientation (forward-facing direction)
v_t	linear speed along the direction of θ_t
ω_t	angular speed about $\mathbf{p}_t$

The tuple ($\mathbf{p}_t$, θ_t, v_t, ω_t) represents the simulated state of the character at time t. At each time step, any combination of the following two controls may be applied:

a_t	linear acceleration along the direction of θ_t
α_t	angular acceleration about $\mathbf{p}_t$

These controls model the four basic controls for our character: (*speed up, slow down, turn left, turn right*). Speeding up and slowing down are represented by positive and negative values of a_t respectively. Similarly, positive values of α_t correspond to left turns, while negative values correspond to right turns. Section 7.3 explains how these controls are calculated at each time step.

Once the controls a_t and α_t have been specified, the state variables are integrated forward discretely by the time step Δt. In these experiments, simple fixed-step Euler integration was used, but more sophisticated integration methods may be used if desired. For Euler integration, the state propagation equations are as follows:

$$\begin{aligned}
x_{t+\Delta t} &= x_t + (v_t \cos\theta_t)\Delta t \\
y_{t+\Delta t} &= y_t + (v_t \sin\theta_t)\Delta t \\
\theta_{t+\Delta t} &= \theta_t + \omega_t \Delta t \\
v_{t+\Delta t} &= v_t + a_t \Delta t \\
\omega_{t+\Delta t} &= \omega_t + \alpha_t \Delta t
\end{aligned}$$

The simulation proceeds in this fashion iteratively. As long as the values of the controls are reasonable relative to the size of the time step Δt , the motion will be smooth and continuous.

7.2 Preparing the Motion Capture Data

As a one-time pre-processing step, the motion capture data to be applied to the path is prepared for the path following phase. Here we assume that the original motion capture data is cyclic, and roughly follows a straight line. To begin, the transformation of the root joint (for example, the *Hip* joint) for each data frame is factored into two parts: the transformation of a *base point* (the point of projection of the origin of the root joint to the walking surface) moving in a straight line at a constant speed throughout the duration of the cycle, and the relative transformation of the root joint with respect to the moving base point.

The reason for performing this decomposition is to allow us to apply the motion capture data to an arbitrary curve much more easily. Traditionally, motion capture data comes in a format that specifies the transformation of the root joint relative to some fixed global frame. Instead of trying to control the root joint directly to follow a computed path, we can control the base point, which by construction travels in a straight line and at a constant speed throughout the duration of the motion cycle.

The trajectory of the base point is computed by first projecting the origin of the root joint onto the walking surface for both the first frame and the last frame of the motion cycle. The base point should move in the direction corresponding to the translational difference between these two projected points. The total distance traveled by the base point should be the length of this translational difference. The velocity of the base point is simply this distance divided by the time elapsed during playback of the motion cycle. The base point is the point to which we will map the center of our oriented disc model for path following. The direction of motion of the base point corresponds to the forward-facing direction of the disc model.

The velocity of the base point is the *canonical average velocity* V of the root joint over the duration of the motion. If the base point is made to move along a straight line at this velocity, the motion capture data will appear as it does in its raw form. Since we will be controlling the base point for path following, it will move along curved paths, and at differing velocities. In order to improve the appearance of the motion for velocities other than the average velocity, we can incorporate other motion capture data sets taken at different walking or running speeds. An alternative to this, is to pre-compute a table of interpolation factors for the joint rotations, indexed by velocity. Smaller base point velocities will result in smaller joint rotations. For example, for a basic walk cycle, the interpolation factors can be pre-selected for each base point velocity such that sliding of the character's feet along the walking surface is minimized. This simple interpolation method is used in the implementation described here, and results in fairly reasonable motions for transitioning between a standing position to a full-speed walk, and in coming to a stop at the end of a path. A more sophisticated method might utilize IK to enforce "no-sliding" constraints on the feet of the character.

7.3 Calculating the Controls

In this section, we describe a simple method for computing the two control variables, a_t and α_t for our character during each time step of the simulation. The method is based on proportional derivative (PD) control. Given the current state of the system $(\mathbf{p}_t, \theta_t, v_t, \omega_t)$, a *desired* state $(\hat{\mathbf{p}}_t, \hat{\theta}_t, \hat{v}_t, \hat{\omega}_t)$ is calculated. The controls a_t and α_t are then computed to move the system towards the desired state.

The computation proceeds as follows: Given a path $\mathcal{P}$ computed by the planning phase, a desired position along the path $\hat{\mathbf{p}}_t$ is calculated relative to the current position $\mathbf{p}_t$. The desired position is typically set to be slightly ahead of the current position along the path, as this tends to smooth out any sharp corners on the path. Next, the desired orientation $\hat{\theta}_t$ is computed so as to face the character towards the desired position $\hat{\mathbf{p}}_t$. The desired angular speed $\hat{\omega}_t$ is set proportional to the difference (error) between the current orientation θ_t and the desired orientation $\hat{\theta}_t$. The desired linear speed $\hat{v}_t$ has three alternatives. If the error in orientation is *small*, $\hat{v}_t$ is simply set to be the canonical average velocity V of the motion capture cycle. Otherwise, if the error in orientation is *large*, the character is facing the wrong direction, so the speed is set to be some small non-zero value to force the character to slow down and turn around. Lastly, if the character is nearing the end of the path (the goal location), $\hat{v}_t$ is calculated proportional to the difference between the current position and the goal location. After $\hat{v}_t$ is obtained, the controls a_t and α_t are calculated. The linear acceleration a_t is set proportional to the difference between the current and desired linear speed, while the angular acceleration α_t is set proportional to the difference between the current and desired angular speed. The state of the system is integrated forward by a discrete time step Δt , and the entire process is repeated for the next time step.

All of the calculations involving proportional derivative terms above require the following gains[2] to be specified:

k_p position gain	k_θ orientation gain
k_v linear speed gain	k_ω angular speed gain

As long as the gains are set to reasonable values relative to the size of the time step Δt , the resulting motion will be smooth and continuous.

To summarize, all of the control calculations are listed below. First, we calculate the desired state $(\hat{\mathbf{p}}_t, \hat{\theta}_t, \hat{v}_t, \hat{\omega}_t)$, and then compute the controls needed

[2] The gains represent how quickly errors (differences between the current and the desired) are resolved. Since a discrete time step is being used, some care must be taken when setting the gains. Gains set too high will cause oscillations (an *underdamped* system), while gains set too low will fail to correct errors (an *overdamped* system). Setting the gains properly will result in a *critically-damped* system, that asymptotically corrects errors without overshoot. The reader is referred to any textbook on feedback control for more detailed information.

to move towards the desired state.

$$
\begin{aligned}
\hat{\mathbf{p}}_t &= (\hat{x_t},\, \hat{y_t}) \text{ from path } \mathcal{P} \\
\hat{\theta}_t &= \operatorname{atan2}(\hat{y_t} - y_t, \hat{x_t} - x_t) \\
\hat{\omega_t} &= k_\theta(\hat{\theta}_t - \theta_t) \\
\hat{v_t} &= \begin{cases} \text{canonical speed } V & \text{if } |\hat{\theta}_t - \theta_t| \le \theta_{turn} \\ \text{small speed } \epsilon & \text{if } |\hat{\theta}_t - \theta_t| > \theta_{turn} \\ k_p(|\hat{\mathbf{p}}_t - \mathbf{p}_t|) & \text{if near the goal} \end{cases} \\
a_t &= k_v(\hat{v_t} - v_t) \\
\alpha_t &= k_\omega(\hat{\omega_t} - \omega_t)
\end{aligned}
$$

In the computation of $\hat{\theta}_t$, $\operatorname{atan2}(y, x)$ is the standard two-argument arctangent function.

After all controls are calculated, the state is integrated forward discretely by the time step Δt , as described in Section 7.1. The subsequent state becomes the new location and orientation for the base point of the character. To animate the remaining joints, the current velocity v_t is used to index into the motion interpolation table as described in Section 7.2.

8 Experimental Results

The algorithm described here has been implemented on a 200MHz SGI Indigo2 running Irix 6.2 with 128MB of RAM and an SGI EXTREME graphics accelerator card. Good performance has been achieved, even on complex scenes with multiple characters and environments composed of more than 15,000 triangle primitives. During an interactive session, the user can click and drag on the goal location or obstacles, and the path planner will calculate an updated, minimal-length, collision-free path (if one exists) in approximately one-tenth of one second on average. The path is then sent directly to the controller, and the character will immediately begin following the new path. The precise timing results are summarized in Section 8.1.

The generation of the projection bitmap for path planning was accomplished by rendering all obstacles to the back buffer using standard OpenGL calls. The rendering style optimizations for fast obstacle growth were implemented as described in Section 6.1. The resulting pixel values were subsequently read directly from the framebuffer. The Dijkstra's algorithm implementation uses fixed-point math, resulting in a planner that runs almost exclusively using fast integer arithmetic.

For the purposes of path following, the simple PD controller described in Section 7 was implemented for a human-like character with 17 joints. Two sets of motion capture data were used in the experiments: a walk cycle and a jog cycle. As expected, the slower canonical speed of the walk cycle facilitated much better tracking performance along the path compared with the jogging motion. The values of the gains used for the controller were as follows: (k_p= 1.0, k_θ= 5.0, k_v= 5.0, k_ω= 10.0). These gain values are compatible with standard units of meters, radians, and seconds, which were used throughout the experiments. The value of the time step was $\Delta t = 0.0333$ seconds (1/30 sec). Although these gain values were obtained via trial and error, research is underway to automatically compute the optimal gains for path following.[1] Sample output is illustrated in Figure 4. Multiple characters were run simultaneously, each planning around the other characters as they followed their own computed paths.

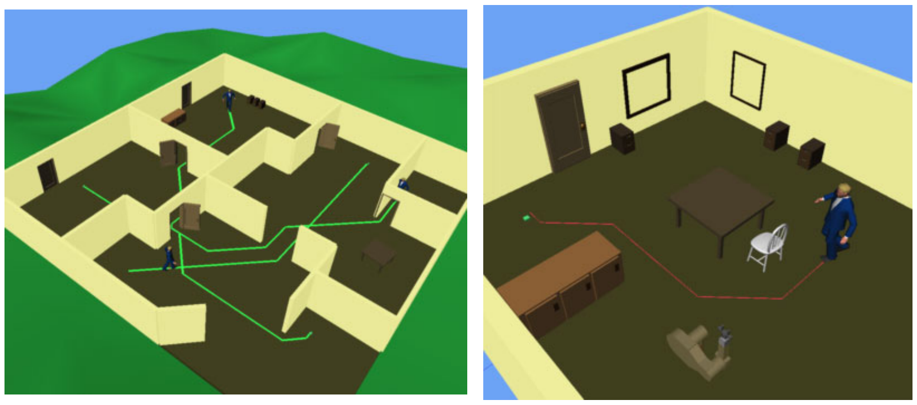
Fig. 4. Screen shots of interactive animation sessions. The left image shows multiple characters navigating in an office environment. The image on the right shows a single character in more detail.

8.1 Timing Results

The average projection, search, and total *elapsed* execution times during repeated invocations of the planner during an interactive session were tabulated. The timing results are summarized in Table 8.1. All values listed in the table are in units of milliseconds, and were averaged from N = 100 independent trials with varying goal locations and obstacle positions. Different grid resolutions were tested ranging between 45 and 150 cells on a side. The total number of triangle primitives in the Maze scene and the Office scene were 2,780 and 15,320 respectively.

9 Discussion and Future Work

Graphic Animation by computer lies at the boundary of modeling and simulating the real world, and shares much in common with the design and control of robotic systems. In this paper, a fast path planner along with a simple path following controller is used to quickly synthesize collision-free motions for level-terrain navigation goals for human-like animated characters. Navigation goals are

Table 1. Average Total Execution Time for Path Planning.

Scene (grid size)	Project	Search	Total (msec)
Maze (50 x 50)	9.5	7.2	16.7
Maze (100 x 100)	34.5	37.4	72.0
Maze (150 x 150)	82.9	79.8	163.0
Office (45 x 45)	47.7	13.9	61.7
Office (90 x 90)	62.7	27.4	90.2
Office (135 x 135)	139.2	56.8	196.0

specified at the task level, and the resulting animation is derived from underlying motion capture data driven by a simple proportional derivative controller. The speed of the planning algorithm enables it to be used effectively in environments where obstacles and other characters move unpredictably.

Although useful in its present form, the algorithm could be improved in a number of important ways. The most severe limitation of the planner is the level-terrain requirement. Extending the algorithm to handle uneven-terrain is possible, but it would involve redesigning the geometry clipping and projection operations. Perhaps the approach taken by Hsu and Cohen would be more appropriate in this situation [14]. Possible extensions to the basic algorithm, include incorporating into the planning process the ability to step over low obstacles, or duck under overhangs. One idea might be to utilize the depth information information that is generated, but is currently being ignored during the projection process. The hardware Z-buffer stores a relative measure of the depth, yielding a simple height field of the environment, which might be useful for deciding a navigation strategy. The same basic idea could perhaps be applied to even more aggressive means of circumventing obstacles, such as utilizing stairs/elevators, climbing, jumping, or crawling. Another limitation of the current approach is the approximate nature of the grid, which may fail to find a free path when one exists, especially when it involves navigating through narrow passages. Perhaps a multi-resolution strategy would be appropriate.

The path following controller as described here is overly-simplistic, and ignores such subtleties of human motion as turning in-place, or side-stepping between narrow openings. Incorporating more motion capture data sets at different velocities, or along curved paths would also likely improve the final appearance of the animation. In addition, the ability to automatically compute the optimal values for the controller gains based on the simulation constants and the canonical speed of the motion capture data would be a very useful improvement. Knowing these optimal gains might also facilitate the calculation of conservative error-bounds on the performance of the path following controller.

Efforts to incorporate active perception based on simulated vision into the planning process are currently underway[1]. In addition, some simple velocity prediction to take into account the estimated motion of other characters and obstacles during planning is also being investigated. Clearly, many challenging research issues must be faced before more interesting motions and intelligent behaviors for autonomous animated characters can be realized.

Acknowledgments

Much gratitude is extended to David Hsu for his thoughtful insights and helpful suggestions, and to Diego Ruspini for reviewing the initial manuscript in detail. Thanks also to Yotto Koga, Craig Becker, Michael Svihura, David Lin, Jing Xiao, and Jean-Claude Latombe, who through informal conversations helped to formulate the ideas presented here. This research is supported in part by a National Science Foundation Graduate Fellowship in Engineering, and MURI grant DAAH04-96-1-007 (Army).

References

[1] J. J. Kuffner Jr., *An Architecture for the Design of Intelligent Animated Characters*, Ph.D. thesis, Stanford University *(in preparation)*.

[2] J. C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA, 1991.

[3] A. Witkin and Z. Popovic, “Motion warping,” in *Proc. SIGGRAPH '95*, 1995.

[4] A. Bruderlin and L. Williams, “Motion signal processing,” in *Proc. SIGGRAPH '95*, Robert Cook, Ed. ACM SIGGRAPH, Aug. 1995, Annual Conference Series, pp. 97–104, Addison Wesley, held in Los Angeles, California, 06-11 August 1995.

[5] D. Baraff, “Analytical methods for dynamic simulation of non-penetrating rigid bodies,” in *Proc. SIGGRAPH '89*, 1989, pp. 223–231.

[6] B. Mirtich, *Impulse-Based Dynamic Simulation of Rigid Body Systems*, Ph.D. thesis, University of California, Berkeley, CA, 1996.

[7] A. Witkin and Kass M., “Spacetime constraints,” in *Proc. SIGGRAPH '88*, 1988, pp. 159–168.

[8] J. T. Ngo and J. Marks, “Spacetime constraints revisited,” in *Proc. SIGGRAPH '93*, 1993, pp. 343–350.

[9] Z. Liu, S. J. Gortler, and F. C. Cohen, “Hierachical spacetime control,” in *Proc. SIGGRAPH '94*, 1994, pp. 35–42.

[10] R. A. Brooks, “A layered intelligent control system for a mobile robot,” in *Robotics Research The Third International Symposium*. 1985, pp. 365–372, MIT Press, Cambridge, MA.

[11] R. C. Arkin, “Cooperation without communication: Multiagent schema based robot navigation,” *Journal of Robotic Systems*, pp. 351–364, 1992.

[12] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg, “Real-time robot motion planning using rasterizing computer graphics hardware,” in *Proc. SIGGRAPH '90*, 1990.

[13] Y. Koga, K. Kondo, J. Kuffner, and J.-C. Latombe, “Planning motions with intentions,” in *Proc. SIGGRAPH '94*, 1994, pp. 395–408.

[14] D. Hsu and M. Cohen, “Task-level motion control for human figure animation,” Unpublished Manuscript, 1997.

[15] M. R. Jung, N. Badler, and T. Noma, “Animated human agents with motion planning capability for 3D-space postural goals,” *The Journal of Visualization and Computer Animation*, vol. 5, no. 4, pp. 225–246, October 1994.

[16] J. P. Granieri, W. Becket, B. D. Reich, J. Crabtree, and N. L. Badler, "Behavioral control for real-time simulated human agents," in *1995 Symposium on Interactive 3D Graphics*, Pat Hanrahan and Jim Winget, Eds. ACM SIGGRAPH, Apr. 1995, pp. 173–180, ISBN 0-89791-736-7.

[17] E. Kokkevis, D. Metaxas, and N. I. Badler, "Autonomous animation and control of four-legged animals," in *Graphics Interface '95*, Wayne A. Davis and Przemyslaw Prusinkiewicz, Eds. Canadian Information Processing Society, May 1995, pp. 10–17, Canadian Human-Computer Communications Society, ISBN 0-9695338-4-5.

[18] N. Badler, "Real-time virtual humans," *Pacific Graphics*, 1997.

[19] X. Tu and D. Terzopoulos, "Artificial fishes: Physics, locomotion, perception, behavior," in *Proc. SIGGRAPH '94*, Andrew Glassner, Ed. ACM SIGGRAPH, July 1994, Computer Graphics Proceedings, Annual Conference Series, pp. 43–50, ACM Press, ISBN 0-89791-667-0.

[20] H. Noser, O. Renault, D. Thalmann, and N. Magnenat Thalmann, "Navigation for digital actors based on synthetic vision, memory and learning," *Comput. Graphics*, vol. 19, pp. 7–19, 1995.

[21] K. Perlin and A. Goldberg, "IMPROV: A system for scripting interactive actors in virtual worlds," in *Proc. SIGGRAPH '96*, Holly Rushmeier, Ed. ACM SIGGRAPH, 1996, Annual Conference Series, pp. 205–216, Addison Wesley.

[22] K. Perlin, "Real time responsive animation with personality," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 1, pp. 5–15, March 1995, ISSN 1077-2626.

[23] B. M. Blumberg and T. A. Galyean, "Multi-level direction of autonomous creatures for real-time virtual environments," in *Proc. SIGGRAPH '95*, Robert Cook, Ed. ACM SIGGRAPH, Aug. 1995, Annual Conference Series, pp. 47–54, Addison Wesley, held in Los Angeles, California, 06-11 August 1995.

[24] P. Maes, D. Trevor, B. Blumberg, and A. Pentland, "The ALIVE system full-body interaction with autonomous agents," in *Computer Animation '95*, Apr. 1995.

[25] J. Bates, A. B. Loyall, and W. S. Reilly, "An architecture for action, emotion, and social behavior," in *Artificial Social Systems : Proc of 4th European Wkshp on Modeling Autonomous Agents in a Multi-Agent World*. 1994, Springer-Verlag.

[26] D. C. Brogan and J. K. Hodgins, "Group behaviors with significant dynamics," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1995.

[27] J. K. Hodgins, "Three-dimensional human running," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 1996.

[28] J. H. Reif, "Complexity of the mover's problem and generalizations," in *Proc. 20th IEEE Symp. on Foundations of Computer Science (FOCS)*, 1979, pp. 421–427.