Control of complex, physically simulated robot groups

David C. Brogan

University of Virginia
Charlottesville, Virginia

ABSTRACT

Actuated systems such as robots take many forms and sizes but each requires solving the difficult task of utilizing available control inputs to accomplish desired system performance. Coordinated groups of robots provide the opportunity to accomplish more complex tasks, to adapt to changing environmental conditions, and to survive individual failures. Similarly, groups of simulated robots, represented as graphical characters, can test the design of experimental scenarios and provide autonomous interactive counterparts for video games. The complexity of writing control algorithms for these groups currently hinders their use. A combination of biologically inspired heuristics, search strategies, and optimization techniques serve to reduce the complexity of controlling these real and simulated characters and to provide computationally feasible solutions.

Keywords: Multiagent, dynamic simulation, group navigation, herds, swarms

1. INTRODUCTION

Animated characters are needed to play the role of teachers or guides, teammates or competitors, or just to provide a source of interesting motion in virtual environments. The characters in a compelling virtual environment must have a wide variety of complex and interesting behaviors and must be responsive to the actions of the user. The difficulty of constructing such synthetic characters currently hinders the development of these environments, particularly when realism is required. In this paper, we describe one approach to populating graphical environments, using dynamic simulation to generate the motion of characters (figure 1).

Figure 1. Images of 105 simulated one-legged robots and 6 simulated bicycle riders.

Motion for characters in virtual environments can be generated with keyframing, motion capture, or dynamic simulation. All three approaches require a tradeoff between the level of control given to the animator and the automatic nature of the process. Animators require detailed control when creating subtle movements that are unique or highly stylized. Generating expressive facial animations usually requires this low level of control. Automatic methods are beneficial because they can interactively produce motion for characters based on the continuously changing state of the user and other characters in the virtual environment.

Further author information:
E-mail: dbrogan@cs.virginia.edu

Keyframing requires that the animator specify critical, or key, positions for the animated objects. The computer then fills in the missing frames by smoothly interpolating between those positions. The specification of keyframes for some objects can be partially automated with techniques such as inverse kinematics, but keyframing still requires that the animator possess a detailed understanding of how moving objects should behave over time as well as the talent to express that information through the configuration of the character. A library of many keyframed animations can be generated off-line and subsequently accessed in an interactive environment to provide the motion for a character that interacts with the user.

Motion capture is one of the most commonly used animation techniques. Magnetic or vision-based sensors are placed on an actor to record the positions of body parts or joint angles as the actor performs a desired action. This recorded motion is then played back through a graphical character. Motion capture is growing in popularity because of the relative ease with which many human actions can be recorded. In particular, sports video games often use motion capture to generate the stylistic movements of athletes in an interactive environment. However, a number of problems prevent motion capture from being an ideal solution for all applications. As with keyframing, recorded motion capture sequences must be skillfully blended together to create realistic movements that change in response to the actions of the user. Discrepancies between the shapes or dimensions of the motion capture subject and the graphical character also can lead to problems. If, for example, the subject was recorded touching a real table, the hands of a shorter graphical character might appear to intersect the table. Finally, the current technology makes it difficult to record certain movements. Magnetic systems often require connecting the subject to a computer by cables, restricting the range of motion, and they produce noisy data when metal objects such as treadmills are close by. Optical systems have problems with occlusion caused by one body part blocking another from view. Motion capture will become easier to use in interactive environments as researchers develop automatic techniques that reuse motion captured segments to animate graphical characters of many shapes and sizes and increase the variety of character actions by automatically blending two motion captured movements with a smooth transition.

Unlike keyframing and motion capture, simulation uses the laws of physics to generate motion of figures and other objects. Virtual characters are usually represented as a hierarchy of rigid body parts connected by telescoping and rotary joints. The equations of motion that simulate these body parts calculate the movements that result from acceleration due to gravity, forces caused by the ground during collisions, and torques applied at a joint. Each simulation also contains control algorithms that calculate the appropriate torques at each joint to accomplish such desired behaviors as hopping, riding, and balancing. Higher-level algorithms can use these control algorithms to direct a group of simulations to move as a herd or to navigate along a narrow path.

Dynamic simulation offers two potential advantages over other sources of motion for synthetic characters in virtual environments. First, simulated motion generates physically realistic motion that may be difficult to create using keyframing. While not all environments need or even benefit from physical realism, it is required for a growing set of applications such as sports training, task training, and team-oriented games. Second, because their motion is computed on the fly, dynamically simulated characters offer a more precise form of interactivity than characters animated with a fixed library of precomputed or recorded motion. For example, in football video games, the motion resulting from a collision between opposing players is a function of the magnitude and direction of their velocities as well as their body configurations at the time of impact. Because the number of initial conditions is very large, modeling this interaction accurately with a library of fixed motions is difficult.

One disadvantage of dynamic simulation is the computational cost. The bicyclist example presented here requires a modern processor to simulate in real time (such that simulated time passes at the same rate as wall clock time). Dynamic simulation also imposes some limitations on the behavior of synthetic characters. Simulated characters are less maneuverable than those modeled as point-mass systems and those that move along paths specified by animators. For example, although a point-mass model can change direction instantaneously, a legged system can change direction only when a foot is planted on the ground. If the desired direction of travel changes abruptly, the legged system may lose its balance and fall. These limitations are physically realistic and therefore intuitive to the user, but they also make it more difficult to design robust algorithms for group behaviors, obstacle avoidance, and path following.

To illustrate the use of dynamically simulated characters, we created a group of simulated human bicyclists and a group of alien bicyclists that ride on a bicycle race course (figure 2). Our earlier results indicate[1,2] that we can generate algorithms that support characters of different types and groups of varying size, however, manual tuning was required to obtain good performance. In this paper we describe automatic tuning methods and algorithms that generate improved group performance.

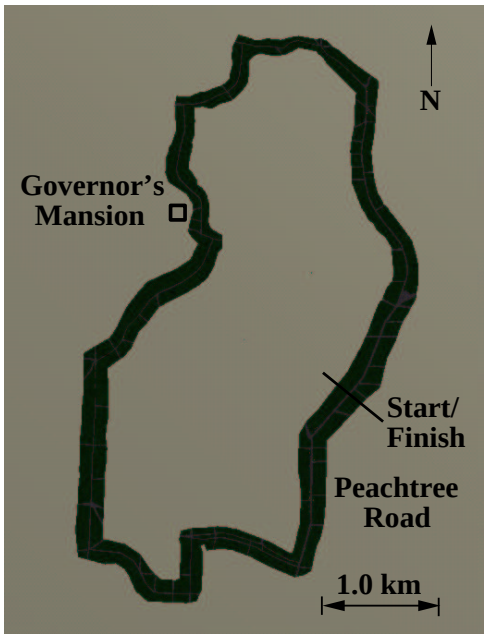

Figure 2. The 13-kilometer race course from the 1996 Olympics. This graphical course captures the elevation, side streets, and surrounding terrain of the streets from Atlanta, Georgia where the race was held.

2. BACKGROUND

Herding, flocking, and schooling behaviors of animals have been studied extensively over the past century, and this research has stimulated attempts to create robots and simulated characters with similar skills. Biologists have found that groupings in animals are created through an attraction that modulates the desire of each member to join the group with the desire to maintain a sufficient distance from nearby characters.[3] As an example of this attraction, Cullen, Shaw, and Baldwin[4] report that the density of fish is approximately equal in all planes of a school, as if each fish had a sphere around its head with which it wished to contact the spheres of other fish. Biologists have found that herding benefits group members by limiting the average number of encounters with predators (data summarized in Veherencamp[5]). Group behaviors also allow animals to hunt more powerful animals than those they could overpower as individuals. The success of behaviors such as these in biological systems argues the merit of exploring their use in robotic systems. An understanding of these behaviors is essential for realistic characters in virtual environments.

Early progress in the simulation of group behaviors was made by Reynolds.[6] Actors in his system are birdlike objects similar to the point masses used in particle systems except that each bird also has an orientation. The birds maintain proper position and orientation within the flock by balancing their desire to avoid collisions with neighbors, to match the velocity of nearby neighbors, and to move towards the center of the flock. Each bird uses only information about nearby neighbors. This localization of information simulates one aspect of perception and reaction in biological systems and helps to balance the three flocking tendencies. Reynolds's work demonstrates that realistic-looking animations of group formations can be created by applying simple rules to determine the behaviors of the individuals in the flock.

Sugihara and Suzuki demonstrated that multiple simulated robots can form stable formations when each robot executes an identical algorithm for position determination within the group.[7] Each robot perceives the relative positions of all other robots and has the ability to move one grid position during each unit of time. By adjusting their position relative to either the most distant or the closest neighbor, the robots can form a regular geometric shape such as a circle. By carefully constructing the algorithm that each robot uses in determining intragroup position, formations will emerge without *a priori* knowledge about the total number of robots or their initial positions. Designation of leaders allows the simple rules of the group to create leader-follower algorithms and to demonstrate the division of a formation into smaller groups.

Arkin explored the question of communication in a group of interacting mobile robots in the laboratory using schema-based reactive control.[8] Example schemas are *move-to-goal*, *move-ahead*, and *avoid-static-obstacle*. Each behavior computes a velocity vector that is combined with the velocity vectors from the other behaviors. The combined velocity vector is used to control the robot. Arkin demonstrated that multiple robots can effectively complete group tasks such as foraging and can retrieve large quantities of goal items with little or no explicit communication.

Mataric explored emergent behavior and group dynamics for 20 wheeled vehicles in the laboratory. These robots, like Arkin's, do not explicitly communicate state or goals and the system has no leaders. This work demonstrated that combinations of such simple behaviors as aggregation and dispersion can produce such complex relationships as flocking in physical robots in the laboratory.[9] The robots utilize the knowledge that they are all identical when executing behaviors, but an extension to these results found that heterogeneous agents, where robots moved in a predetermined order, do not perform significantly better than homogeneous ones in aggregation and dispersion experiments.[10] In these experiments, a hierarchy is created in which an ordering between the agents determines which agent will move first in completing such tasks as grouping and dispersing.

Tan and Lewis developed control algorithms for groups of non-holonomic simulated robots moving in formation.[11] The formation is defined by a virtual structure, a polygon with vertices that specify robot positions, which can be translated and rotated, but cannot shift, skew, or scale. Successful group formation navigation requires the robots maintain a rigid geometric relationship to each other by following the points of the virtual structure lattice. The goal of the control algorithm is to specify the virtual structure's trajectory to a destination position while minimizing the sum of distances between each robot and its assigned point on the virtual structure. A model of the robot's dynamic abilities estimates the positions each robot can reach at the next time step and provides an accurate penalty value for all spots the robot cannot reach. The evaluation function uses this model of a robot's reachable positions when evaluating the numerous translations and rotations that could be applied to the virtual structure. The distance between the virtual structure and the goal position is also used when evaluating new positions. The Davidson-Fletcher-Powell method is used to search the space of virtual structure transformations. This paper produces interesting results because the formation algorithm adjusts to robots with different locomotion capabilities. Likewise, the algorithm is robust to robot failure as the virtual structure adjusts to minimize the distance between the group and the failing robot while simultaneously moving towards the goal.

3. BICYCLE SIMULATION LOCOMOTION CONTROLLER

The bicyclists in these groups consist of three components: physical simulation, locomotion controller, and navigation controller (figure 3). The physical simulation is defined by equations of motion that represent a hierarchy of rigid body parts and the rotary and telescoping joints that connect them. The equations of motion for the bicyclist were formulated using a commercially available package.[12] A character's *locomotion controller* computes how to actuate its joints in order to move at a specified desired velocity. Due to kinematic and dynamic constraints, the locomotion controllers cannot instantaneously eliminate errors between a character's desired velocity and its actual velocity. These limitations to a character's maneuverability, or *mobility constraints*, are realistic and intuitive to the user, but they make it more difficult for *navigation controllers* to compute a desired velocity for each character that accomplishes group behaviors, obstacle avoidance, and path following.

For these experiments, we developed two similar characters, a human and an alien bicycle rider. The human bicycle rider[13] is modeled by a 15-segment rigid-body model connected by rotary joints with 22 controlled degrees of freedom. Some joints, like the knee, are modeled as a single-axis pin joint; others, like the wrist and shoulder, are modeled by two- and three-axis gimbal joints. The volume, mass, center of mass, moments of inertia, and distance between the joints are calculated from a polygonal representation of the human body (figure 4 and appendix table 2). The algorithm used to calculate the properties of the polygonal model integrates over the set of tetrahedra formed by the triangular faces of the model and the origin.[14] Density data obtained from the anatomical literature were used in calculating the dynamic properties of the body segments[15] (appendix table 3). The degrees of freedom of the bicycle model are shown in figure 4 and the parameters of the model are given in appendix table 2. The alien bicycle rider[16] has slightly different body masses and limb lengths (figure 5). Due to the similarities between the two types of bicyclists, the locomotion controller required only slight adjustments to work well for both and we will discuss the shared controller in this paper.

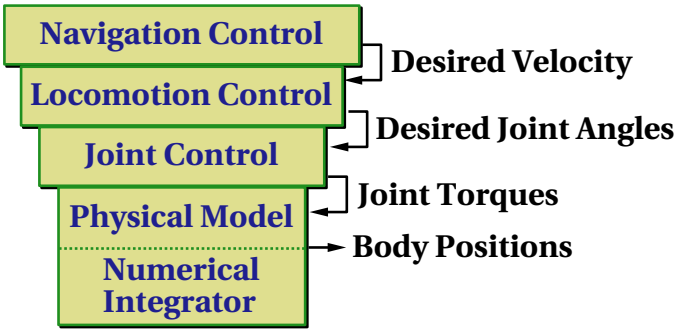

Figure 3. The simulation system used to animate physically simulated characters. The locomotion controller obtains a desired velocity from the navigation controller and computes the joint positions that will achieve it. The desired joint positions are passed to the joint controller, where joint torques are computed to eliminate errors in joint position. The joint torques are used by the numerical integrator to compute new positions for all the character's body parts.

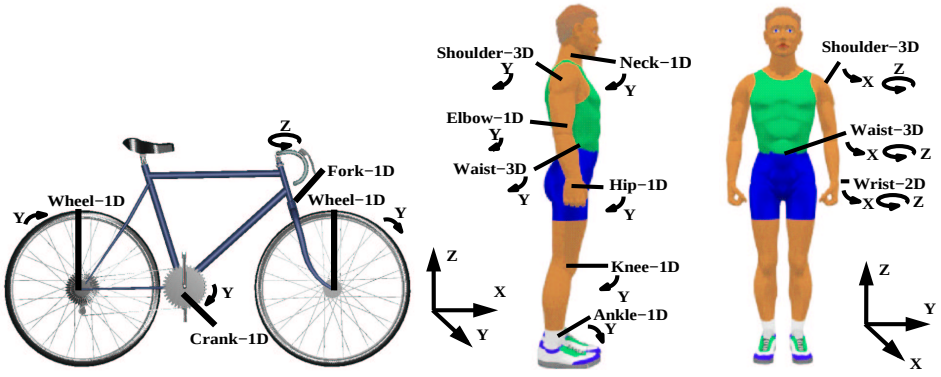

Figure 4. The controlled degrees of freedom of the bicycle and human models. The human model has fourteen joints; the degrees of freedom at each joint are shown in the diagram as are the four degrees of freedom of the bicycle model. The human rider is attached to the bicycle by a pivot joint between the seat and the pelvis. The polygonal models were purchased from Viewpoint Datalabs.

The navigation controller eliminates errors in the bicyclist's velocity by adjusting the facing direction and speed of the bicycle. The simulated human rider controls the facing direction and speed of the bicycle by applying forces to the handlebars with the hands and to the pedals with the feet. Spring and damper systems connect the hands to the handlebars, the feet to the pedals, and the crank to the rear wheel. The connecting springs are two-sided, and the bicyclist is able to pull up on the pedals as if the bicycle were equipped with toe-clips and a fixed gear.

The locomotion control system adjusts the speed of the bicycle by moving the bicyclist's legs to produce a torque at the crank. The desired torque at the crank is $\tau_c = k(v - v_d)$, where k is a gain, v is the magnitude of the velocity, and v_d is the desired velocity (specified by the navigation controller). The force that can be applied by each leg depends on the angle of the crank because we assume that the legs are most effective when pushing downwards. When the crank is horizontal, the front leg can generate a positive torque and the rear leg can generate a negative torque. To compensate for the crank position, the desired forces for the legs are scaled by a weighting function between zero and one that depends on the crank position, θ_c:

$$w = \frac{\sin(\theta_c) + 1}{2}. \tag{1}$$

If $\tau_c > 0$, the force on the pedal that the left leg should produce is

$$f_l = \frac{w\tau_c}{l} \tag{2}$$

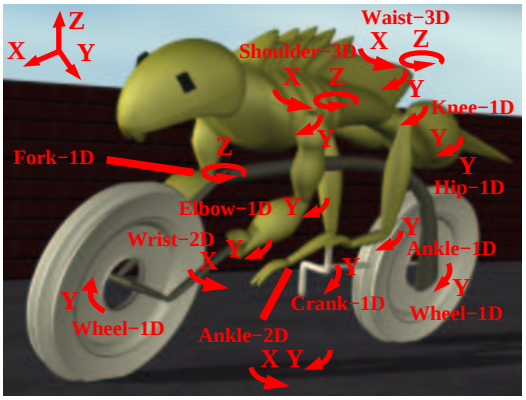

Figure 5. The DOFs of the bicycle and alien models. The alien model has fourteen joints; the DOFs at each joint are shown in the diagram as are the four DOFs of the bicycle model. The alien rider is attached to the bicycle by a pivot joint between the seat and the pelvis.

where l is the length of the crank arms. The desired pedal force for the right leg is

$$f_r = \frac{(1-w)\tau_c}{l} \tag{3}$$

If τ_c is less than zero, the equations for the left and right leg are switched. A kinematic model of the legs is used to compute hip and knee torques that will produce the desired pedal forces.

To steer the bicycle, the locomotion control system computes a desired angle for the fork based on the errors in roll and yaw:

$$\begin{aligned} \theta_f = \quad & -k_\alpha(\alpha - \alpha_d) - k_{\dot\alpha}\dot\alpha + \\ & k_\beta(\beta - \beta_d) + k_{\dot\beta}\dot\beta \end{aligned} \tag{4}$$

where α, α_d, and $\dot\alpha$ are the roll angle, desired roll angle, and roll velocity, respectively, and β, β_d, and $\dot\beta$ are the yaw angle, desired yaw angle (specified by the navigation controller), and yaw velocity. k_α, $k_{\dot\alpha}$, k_β, and $k_{\dot\beta}$ are gains. Inverse kinematics is used to compute the shoulder and elbow angles that would position the hands on the handlebars with a fork angle of θ_f; proportional-derivative servos move the shoulder and elbow joints toward those angles.

These locomotion control laws leave the motion of several joints of the bicyclist unspecified. The wrists and the waist are held at a constant angle with proportional-derivative controllers. The ankle joints are controlled to match data recorded from humans.[17]

4. BICYCLE SIMULATION NAVIGATION CONTROLLER

In the previous section, we described simulated characters that generate physically accurate movements, capture subtleties in bicycling actions, and convey important dynamic information to the viewer. The physical simulation algorithms that generate the accurate movements of these characters not only produce convincing animations, but they also constrain each character's ability to avoid collisions, follow a path, and ride as a group. These mobility constraints limit such capabilities as how quickly a bicyclist can turn, how far it can lean when cornering, and how fast it can ride. The navigation controller, which specifies a character's desired velocity, must operate within these constraints to generate character trajectories that satisfy high-level goals while avoiding undesirable collisions and crashes. Because the human and alien bicyclists have different mass distributions and limb leverage, they also have dissimilar mobility constraints. In this section we develop a navigation controller that easily adjusts to operate for characters with these different physical abilities.

The navigation controller uses data from a character's state, neighbors, and the graphical world to compute a desired velocity, It has three parts: a perceptual algorithm, a desired position algorithm, and a desired velocity

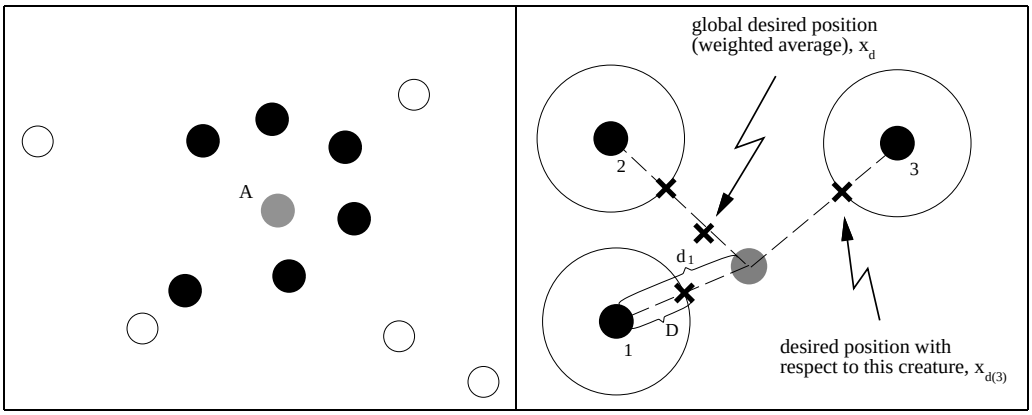

Figure 6. In the left half of the figure, one character is visible to another if it is one of the n closest characters (n is 6 for this example). The black circles represent the set of visible characters, N, and the white those that cannot be seen by A. In the right half of the figure, the locations of the visible characters are used to compute a global desired position for the individual under consideration. The algorithm computes a desired position with respect to each visible character, $x_{d(i)}$, by finding the point on the line between the individual and the visible character that is a constant distance D away from the visible character. These desired positions are averaged with a weighting equal to $1/d_i^2$ where d_i is the distance between the two characters.

algorithm. The perception algorithm determines the set of characters and components from the physical world that are visible to each individual in the group. For this navigation controller, each individual in the group can perceive the relative locations and velocities of the set of visible characters, N (figure 6). The desired position algorithm uses the locations and velocities of the N visible characters and obstacles to calculate a desired position for each individual. The desired velocity algorithm uses the output of the desired position algorithm and the nominal user-specified velocity to compute a desired velocity for each character. The locomotion controller for each character uses the desired velocity to compute joint torques that adjust speed and direction of travel, thereby reducing velocity errors.

4.1. Computing Desired Positions

In this stage of the navigation control algorithm, a set of desired positions is computed for each individual in the group. These desired positions correspond to the herding and path following goals of the navigation controller. Each character computes a desired position relative to its visible neighbors that preserves a close grouping while avoiding collisions. Each individual would be in the perfect position according to the grouping goals of the navigation controller if it could move to this position instantaneously. However, each character must also follow the path, and a desired position is computed that satisfies this goal. In this subsection we describe the algorithms that specify these two desired positions.

4.1.1. Desired position relative to neighbors

Each character's desired position algorithm uses the positions of the n visible neighbors (figure 6) to compute a desired position, $(x_{d(i)}, y_{d(i)})$, relative to each of these characters. This position is a distance D away from visible character, i, on the line between the two characters (figure 6):

$$y = \frac{y_i - y_A}{x_i - x_A} x, \tag{5}$$

where (x_i, y_i) is the current position of character i, and (x_A, y_A) is the current position of character A. When $x_i = x_A$, the two characters are on a line parallel to the y-axis and the slope of the line is assumed to be a large, but not infinite, number. The desired separation distance, D, specifies that $(x_{d(i)}, y_{d(i)})$ is D meters away from character i:

$$D = \sqrt{{x_{d(i)}}^2 + {y_{d(i)}}^2}. \tag{6}$$

Using equations 5 and 6, we can solve for $(x_{d(i)}, y_{d(i)})$:

$$x_{d(i)} = \frac{D}{\sqrt{\left(\frac{y_i - y_A}{x_i - x_A}\right)^2 + 1}}. \tag{7}$$

$y_{d(i)}$ is computed in a similar fashion. After computing the desired position relative to each character in N, the set of desired positions is weighted by the actual distance between each pair of characters, d_i, and averaged to compute a global desired position:

$$herd_x = x_A + \frac{\sum_{i \in N} \text{sgn}(x_A - x_i) \frac{x_{d(i)}}{{d_i}^2}}{\sum_{i \in N} \frac{1}{{d_i}^2}}. \tag{8}$$

sgn(p) returns -1 when p is negative and 1 otherwise. The desired y-position, $herd_y$, is computed similarly.

4.1.2. Desired position relative to path

In addition to avoiding collisions with other individuals in the group, the characters must also ride along a path. The path is constrained to be on a plane, but it is free to twist left and right. The center of the path is modeled as a Catmull-Rom spline and the path edges are 5.0 meters to either side of the center.

The bicyclist path following algorithm seeks to follow the curves of the path while providing bicyclists with the flexibility to ride anywhere within the path edges. By projecting a vector from the bicyclist's current position in its direction of travel, we are able to compute the position of a look-ahead point, $(look_x, look_y)$ that is t seconds away, based on the current speed:

$$look_x = x + speed \cdot t \cos(yaw) \tag{9}$$

$$look_y = y + speed \cdot t \sin(yaw) \tag{10}$$

We compute $(path_x, path_y)$, the closest point on the path's center spline to the look-ahead point, $(look_x, look_y)$ (figure 7 (a)). By steering towards $(path_x, path_y)$, each bicyclist attempts to track the path. However, the value of t greatly affects the performance of the navigation controller. For this reason, t is tuned through a series of offline trials. In these trials, the bicyclist must follow five paths of varying curvature. By using these trials to iteratively tune the value of t, a value is determined that minimizes the accumulated distance between the bicyclist and the path. In the experiments reported in this paper, a look-ahead time of 4.1 seconds is used for the human bicyclist and 1.1 seconds is used for the alien.

The bicyclist should ride towards $(path_x, path_y)$ to prevent straying from the center of the path. However, the bicyclists are unable to avoid collisions with neighbors when they all desire to be at the path's center. Instead, the path following algorithm attempts to position the group centroid at the center of the path. The algorithm computes the shortest distance between the bicyclist and the center of the path spline. It also computes the average position of all the visible neighbors and the shortest distance between this group centroid and the center of the path. The desired position for the bicyclist is defined to lie on a line that intersects, and is normal to, the path at $(path_x, path_y)$. The desired position along this line is cast from the center of the path by an offset distance equal to the bicyclist's current distance from the path center minus the group centroid's distance. This offset distance is capped to be no greater than half the path's width. By aiming towards this point, the relative positions of the group members are preserved and the bicyclist can follow the path without collisions (figure 7 (b)).

4.2. Computing Desired Velocities

The desired velocity algorithm must combine a character's desired positions, current state, neighbor velocities, and the user-specified nominal velocity to compute an appropriate desired velocity. This desired velocity must preserve the character's balance while effectively moving towards the desired positions. Therefore, the navigation control algorithm must take into account each character's physical qualities and locomotion controller. For example, human and alien bicyclists possess different mass distributions, which in turn affect how these characters must compute velocities to eliminate position errors. The navigation controller requires that the desired velocity algorithms be tuned to account for unique attributes of each character.

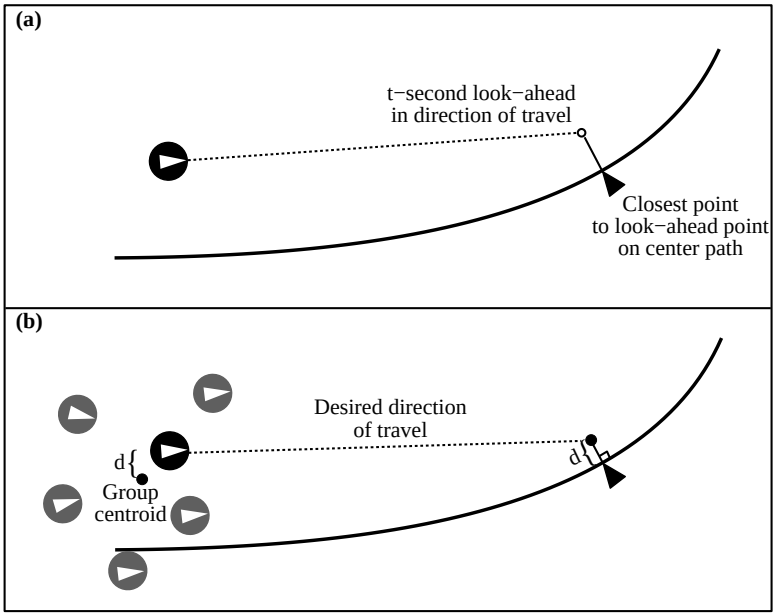

Figure 7. In frame (a), a line is projected from the bicyclist's current position along its direction of travel. A look-ahead point is computed $t \cdot speed$ meters along this line to evaluate the future position on the path. Frame (b) demonstrates how a distance g is computed to represent the distance between the bicyclist's current distance from the group centroid. The navigation controller aims towards a future position that places the group centroid on the path center and preserves the character's offset distance, g, from the center of the path.

4.2.1. Desired velocity for herding

The desired velocity algorithm first determines a desired riding direction and speed that will eliminate the error in position, (e_x, e_y), between the character's current position and the desired position relative to its neighbors, $(herd_x, herd_y)$:

$$e_x = herd_x - x \tag{11}$$

$$e_y = herd_y - y \tag{12}$$

The character's desired riding direction is:

$$herd_{yaw} = \arctan\left(\frac{\dot{y} + k_p e_y + k_d \dot{e}_y}{\dot{x} + k_p e_x + k_d \dot{e}_x}\right), \tag{13}$$

where $(\dot{x}, \dot{y})$ is the character's current velocity and the spring gain, k_p, and damping gain, k_d, are multiplied by the character's error in position and the velocity of the position error, respectively.

The spring and damping gains are tuned by an offline optimization process using simulated annealing. In this process, a bicyclist is commanded to shift its position 4.0 m to the right. The bicyclist must steer to the right and then to the left to complete this ess-maneuver, but the bicyclist's desired riding direction, yaw_{nom}, remains unchanged during the experiment, so the initial and terminal riding directions should be the same. The experiment terminates in a successful state when the character has arrived within ± 0.1 m of the desired position, yaw is within ± 0.1 radians of yaw_{nom}, and $\dot{yaw}$ is within ± 0.1 radians/s. The simulated annealing evaluation function penalizes the fork velocities that accumulate during the experiment with a weight of 0.0001. The time to complete the experiment and the terminal errors in velocity, riding direction (yaw), and lateral position are penalized with weights 1.0, 10.0, 1.0, and 1.0 respectively. Although these error terms are measured with different units, most of their values range between ± 0.1 upon successful termination. The error weights of the cumulative fork velocity error and the terminal bicycle velocity error are scaled to produce similar contributions to the evaluation metric. A successful termination

Character	k_p	k_d
Human Bicyclist	0.297	0.339
Alien Bicyclist	0.467	0.0598

Table 1. Simulated annealing experiments were used to automatically select spring and damping gains for the bicyclists' velocity computation algorithms.

completes the evaluation without any additional penalty, but a bicyclist crash, or a lack of success after 15 s results in a large penalty. Results of the simulated annealing experiments are summarized in table 1.

The system specifies a constant nominal speed, $speed_{nom}$, for all the bicyclists, but this speed must be adjusted to improve each character's ability to reach a desired position. The character's desired speed is increased when it is behind the desired position and it is decreased when it is ahead. The angle, *orient* measures the angle between the character's *yaw* and the line between the character and the desired group position, $(herd_x, herd_y)$. If *orient* is between 0.0 and π, the character is behind the desired position and must speed up, otherwise, the character must slow down:

$$orient = yaw - \arctan\left(\frac{e_y}{e_x}\right). \tag{14}$$

The degree to which the character changes speed depends on the value of *orient* and the distance between the character and the desired position:

$$speed_{error} = k_v \cdot \cos(orient)\sqrt{e_x^2 + e_y^2}, \tag{15}$$

where k_v is a scaling gain set to 0.5. The error in speed, $speed_{error}$, will be the largest when the character is traveling directly towards or away from the desired position. If the character is traveling directly alongside the desired position, the error in speed will be 0.0. The error in speed is added to the nominal speed to determine the speed, $herd_{speed}$, that moves the character towards its desired group position:$herd_{speed} = speed_{nom} + speed_{error}$.

4.2.2. Desired velocity for path following

To follow a path, a desired position ahead of the character was computed. The desired velocity that accomplishes the path following behavior must control the character towards this desired position. First, we compute the angle, $path_{yaw}$, between the character's current position (x, y) and the desired position, $(path_x, path_y)$:

$$path_{yaw} = \arctan\left(\frac{path_y - y}{path_x - x}\right). \tag{16}$$

This direction will keep the bicyclist on the path. The desired path following speed, $path_{speed}$ is set to be the nominal speed:

$$path_{speed} = speed_{nom}. \tag{17}$$

4.2.3. Computing the unified velocity

Finally, the two desired angles are averaged, as are the two desired speeds, to generate desired x and y velocities that move the character towards the two desired positions:

$$\begin{aligned} \dot{x}_d &= \cos\left(\frac{1}{2}(herd_{yaw} + path_{yaw})\right) \cdot \frac{1}{2}\left(herd_{speed} + path_{speed}\right) \\ \dot{y}_d &= \sin\left(\frac{1}{2}(herd_{yaw} + path_{yaw})\right) \cdot \frac{1}{2}\left(herd_{speed} + path_{speed}\right) \end{aligned}$$

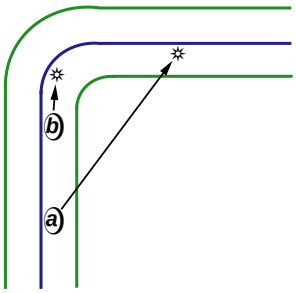

Figure 8. In this figure, the effects of using incorrect values for the bicyclist's look-ahead value, t, are demonstrated. Bicyclist (a) has a large value for t and as a result aims towards points on the center of the path that cause it to turn into corners early, perhaps riding off the road. Bicyclist (b) has a value of t that is too small and as a result, it has not begun to steer into the turn, despite its proximity to the apex. Bicyclist (b) may ride beyond the outside edge of the road as a result of this delayed reaction.

5. RESULTS AND DISCUSSION

Results reported elsewhere[1,2] indicate that these navigation controllers are robust to group size and character type. The controllers generate groups of human bicyclists as they avoid obstacles and ride along a path. The results verify that distributed navigation controllers can be used with the locomotion controllers of physically simulated characters to generate goal-oriented behavior. However, the unique dynamic abilities of each character impose limitations on the character's ability to react to the environment. The navigation controllers described in this paper partially account for these variations by performing simulated annealing tuning experiments for the herding gains and the path following look-ahead time. Additional manual adjustments must be made to prevent the navigation controllers from specifying unreasonable desired velocities to the locomotion controllers.

The tuning procedures described in this paper automatically adjust to many of the unique qualities of the different characters, but they provide indirect parameterizations and fail to capture the variety of mobility constraints that result from a character's dynamic state. The tuned values of k_p and k_d reflect how the characters compute desired velocities from errors in position with respect to visible neighbors. The ratio between the spring and damper gains demonstrates significant differences between the human and alien characters' mobility constraints. The human bicyclist has values of 0.297 and 0.339, which indicate a very high damping coefficient. Subjective analysis of the bicyclist reveals that these gain settings compensate for the character's relatively high degree of lateral momentum. Once the bicyclist begins to lean and turn, it has difficulty returning to an upright posture. Alternatively, the alien bicyclist has a higher value of k_p, 0.467, and a low value for k_d, 0.0598. The relatively low damping term indicates that the character is able to quickly correct orientation errors without the risk of reaching a high yaw velocity. Indeed, qualitative observation of the alien bicyclist reveals that the character can quickly initiate and eliminate lateral velocities. Although these parameters are tuned automatically, they provide only a superficial connection between the navigation controller and the mobility constraints of the character. The computed k_p and k_d values are tied to the exercises used to tune them, rather than being absolute and related only to the character. In our experiments, for example, the tuned parameters allowed the bicyclist to quickly shift its position 4.0 m to the right, but these parameters may be ineffective for smaller transitions. By selecting an appropriate set of tuning exercises, a family of tuned parameters can be created for use in many scenarios. Gain scheduling allows the run-time selection of the most appropriate gains from this set, but successful scheduling algorithms require a good set of optimized gains.

The amount of time in the future, t, that the bicyclists use to evaluate the look-ahead point is also a very important parameter. A large value of t smoothes out the effects of variations in the path's trajectory and provides the opportunity to anticipate and initiate changes in riding direction. Due to the trigonometric relationship between an observed shift in the path position and the magnitude of t, turns in the path result in smaller changes in desired riding direction when t is large. Large t values allow characters to transition into corners more gradually, but it also risks eliminating some curves entirely, thus causing bicyclists to ride off the path (figure 8). The t value of 4.1 s for the human bicyclist reflects its inability to initiate a turn quickly. This high look-ahead value provides ample time to begin a gradual cornering maneuver, but it also reduces the effect of smaller turns in the path. The alien bicyclist

utilizes a much smaller look-ahead value, 1.1 s, which generates a navigation controller that is more responsive to changes in the path's direction. As was the case with the herding gains, the value of t mirrors subjective observations of the two bicyclist's mobility constraints. Unfortunately, the look-ahead parameter is tuned on a particular set of curves, which might not accurately reflect the type of curves encountered at run time. Furthermore, we cannot precompute a set of optimal t look-ahead values that cover the wide range of curvatures present on the bicycle's path.

The herding and look-ahead parameters reflect differences between the mobility constraints of two characters, but additional variations occur within each individual character's range of state configurations. For example, the turning ability of the bicyclists is nonlinear with respect to their roll angle. A perfectly upright bicyclist requires a significant amount of time to complete even a small change in riding direction. Attempts to quickly execute large changes in riding direction result in the character losing balance and falling. A bicyclist already leaning to one side, however, can easily complete more aggressive maneuvers if the turn is in the direction of the lean. The navigation controller ignores how the bicyclist's dynamic abilities change as a result of roll when computing a desired velocity. Instead, the navigation controller heavily clamps the changes in riding direction attempted by the bicyclist. This conservative approach prevents failures when the bicyclist is upright, but it also retards achievable actions when the character is already leaning to one side. An improved navigation controller would adjust its actions based on the dynamic state of the character.

In our future work, we will develop simplified versions of characters that will serve to support navigation controllers and other high-level behaviors. We propose that by reducing the accuracy of some degrees of freedom and eliminating others entirely, the simplified character will be fast to compute while preserving the mobility constraints present in the original simulation. In this way, the simplified character will support the navigation controller to better adapt to different types of characters and changing run-time mobility constraints.

APPENDIX A. TABLES

Table 2. The distance from the center of mass of each link to the distal and proximal joints in x, y, and z. A positive distance along the y axis refers to a location on the left side of the body; a negative distance refers to the right side. The z axis is vertical and the x axis is positive in the direction that the model is facing.

Link	COM to Proximal (x, y, z m)			COM to Distal (x, y, z m)		
Torso to neck				0.012	0.0	0.32
Torso to waist				0.012	0.0	-0.22
Torso to shoulder				-0.048	±0.164	0.12
Head	-0.009	0.0	-0.064			
Pelvis	0.023	0.0	0.103			
Pelvis to hips				0.005	±0.098	-0.11
Pelvis to bike seat				0.0	0.0	-0.15
Upper Leg	0.024	±0.006	0.120	-0.05	±0.019	-0.21
Lower Leg	0.005	±0.019	0.165	-0.002	±0.009	-0.25
Foot	-0.046	±0.009	0.048			
Upper Arm	-0.0002	±0.056	0.120	-0.005	±0.036	-0.17
Lower Arm	-0.025	±0.007	0.090	0.012	±0.014	-0.11
Hand	-0.026	0.0	0.085			
Frame to bike seat	-0.15	0.0	0.028			
Frame to rear wheel				-0.42	0.0	-0.36
Frame to fork				0.46	0.0	0.09
Frame to crank				0.022	0.0	-0.36
Fork to front wheel				-0.026	0.0	0.085
Fork	-0.05	0.0	0.108			

Table 3. Human model's rigid-body parameters. The moments of inertia are computed about each link's COM.

Link	Density (g/cm^3)	Mass (kg)	Moment of Inertia (x, y, z kgm^2)		
Head	1.17	5.89	0.030	0.033	0.023
Torso	1.01	29.27	0.73	0.63	0.32
Pelvis	1.03	16.61	0.23	0.18	0.16
Upper Leg	1.04	8.35	0.15	0.16	0.025
Lower Leg	1.08	4.16	0.055	0.056	0.007
Foot	1.07	1.34	0.002	0.008	0.007
Upper Arm	1.07	2.79	0.025	0.025	0.0050
Lower Arm	1.10	1.21	0.0050	0.0054	0.0012
Hand	1.07	0.551	0.002	0.002	0.001

REFERENCES

1. D. C. Brogan and J. K. Hodgins, "Group behaviors for systems with significant dynamics," *Autonomous Robots* **4**, pp. 137–153, 1997.
2. D. C. Brogan, R. A. Metoyer, and J. K. Hodgins, "Dynamically simulated characters in virtual environments," *IEEE Computer Graphics & Applications* **18**, pp. 58–69, September/October 1998.
3. E. Shaw, "Schooling in fishes: Critique and review," in *Development and Evolution of Behavior*, pp. 452–480, W. H. Freeman: San Francisco, California, 1970.
4. J. M. Cullen, E. Shaw, and H. A. Baldwin, "Methods for measuring the three-dimensional structure of fish schools," in *Animal Behavior*, vol. 13, pp. 534–543, 1965.
5. S. Veherencamp, *Handbook of Behavioral Neurobiology, Volume 3: Social Behavior and Communication*, Plenum Press: New York, NY, 1987.
6. C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," in *Computer Graphics (SIGGRAPH '87 Proceedings)*, M. C. Stone, ed., vol. 21, pp. 25–34, July 1987.
7. K. Sugihara and I. Suzuki, "Distributed motion coordination of multiple mobile robots," in *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pp. 138–143, 1990.
8. R. C. Arkin, "Cooperation without communication: Multi-agent schema based robot navigation," in *Journal of Robotic Systems*, vol. 9(3), pp. 351–364, 1992.
9. M. Mataric, "Designing emergent behaviors: From local interactions to collective intelligence," in *Proceedings of the Second International Conference on Simulation of Adaptive Behavior: From Animals to Animats 2*, pp. 432–441, 1992.
10. M. Mataric, "Kin recognition, similarity, and group behavior," in *Proceedings of the Fifteenth Annual Cognitive Science Society Conference*, pp. 705–710, 1993.
11. K. H. Tan and M. A. Lewis, "Virtual structures for high-precision cooperative mobile robotic control," in *IROS 1996*, November 1996.
12. D. E. Rosenthal and M. A. Sherman, "High performance multibody simulations via symbolic equation manipulation and kane's method," *Journal of Astronautical Sciences* **34**(3), pp. 223–239, 1986.
13. J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien, "Animating human athletics," in *Proceedings of SIGGRAPH '95 (Los Angeles, California, August 6–11, 1995)*, R. Cook, ed., Computer Graphics Proceedings, Annual Conference Series, pp. 71–78, ACM SIGGRAPH, ACM Press, Aug. 1995.
14. S. Lien and J. T. Kajiya, "A symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra," *IEEE Computer Graphics and Applications* **4**(10), pp. 35–41, 1984.
15. W. T. Dempster and G. R. L. Gaughran, "Properties of body segments based on size and weight," *American Journal of Anatomy* **120**, pp. 33–54, 1965.
16. J. Hodgins and N. Pollard, "Adapting simulated behaviors for new characters," in *Proceedings of SIGGRAPH '97 (Los Angeles, California, August 3–8, 1997)*, Computer Graphics Proceedings, Annual Conference Series, pp. 153–162, ACM SIGGRAPH, ACM Press, August 1997.
17. P. Cavanagh and D. Sanderson, *Science of Cycling*, 1986.