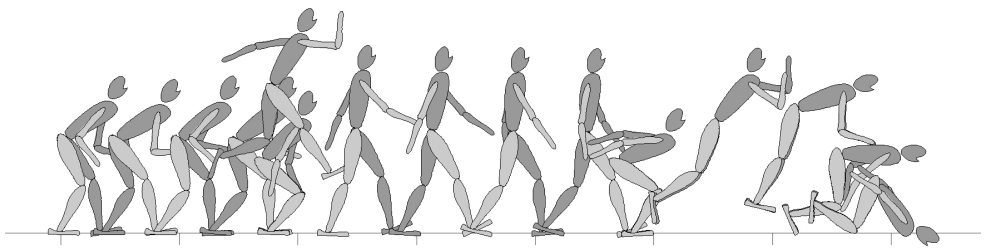
Simulating Biped Behaviors from Human Motion Data

Kwang Won Sok Manmyung Kim Jehee Lee

Seoul National University [*]



Abstract

Physically based simulation of human motions is an important issue in the context of computer animation, robotics and biomechanics. We present a new technique for allowing our physically-simulated planar biped characters to imitate human behaviors. Our contribution is twofold. We developed an optimization method that transforms any (either motion-captured or kinematically synthesized) biped motion into a physically-feasible, balance-maintaining simulated motion. Our optimization method allows us to collect a rich set of training data that contains stylistic, personality-rich human behaviors. Our controller learning algorithm facilitates the creation and composition of robust dynamic controllers that are learned from training data. We demonstrate a planar articulated character that is dynamically simulated in real time, equipped with an integrated repertoire of motor skills, and controlled interactively to perform desired motions.

CR Categories: I.3.7 [Three-Dimensional Graphics and Realism]: Animation—Virtual reality

Keywords: Human Motion, Physically Based Simulation, Biped Walk and Balance, Motion Capture, Controller Learning

1 Introduction

Animating biped characters in a physically-simulated virtual world has been an important issue in the context of computer animation, robotics and biomechanics. A number of techniques from control theory and machine learning have been adopted for designing physically-simulated controllers for each individual human behavior such as walking and balancing. One appealing approach to controller design is learning the control strategies from human performers. A rich set of motion data can be acquired from a live actor performing a specific behavior in a variety of situations. The data set can then be used to inform how animated characters should respond for any given situation. This imitation-based learning approach has demonstrated its potential capability of transferring stylistic human motion to simulated characters and humanoid robots.

Despite much research progress, learning a biped behavior from motion capture data is still challenging when the behavior requires subtle control for maintaining its balance, which is difficult to capture from recorded motion data. Another difficulty is the physical imprecision of the captured motion data. Since the dynamic character model is drastically simplified from the live actor, any biped character that exactly follows the captured joint angle trajectories would lose its balance and fall over in a few steps.

We present a new technique for allowing our physically-simulated planar biped characters to imitate human behaviors. We start with a set of motions that were recorded from a live actor performing actions repeatedly. The motion set is rectified via optimization in such a way that the simplified dynamic model can approximately reproduce the recorded motion trajectories by actuating its joints while maintaining its balance. The control policy of the character's behavior is learned from the rectified motion set. This control policy allows our characters to imitate biped behaviors robustly.

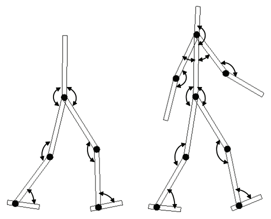
We also propose a method of composing learned dynamic controllers into an integrated framework that allows the character to transition between different behaviors. Robust transitioning from one motor controller to the other is achieved by learning a designated transition controller. The training data for the transition controller are either motion-captured or synthesized by blending existing data kinematically. Once all the controllers are learned, our character can be dynamically simulated in real time and controlled interactively to perform desired motions.

2 Related Work

The realistic biped motion of human characters in computer graphics applications such as feature animation films and video games is manually keyframed, procedurally defined, live captured or physically simulated. A specific form of biped human behavior such as walking and running has successfully been defined as a simple generative procedure [Bruderlin and Calvert 1989].

Recently, data-driven animation techniques using motion capture data have been extensively studied in computer graphics. A number of researchers have explored the method of representing a large amount of motion data as a directed graph and using it to animate

*email:{sinnagie,mmkim,jehee}@mrl.snu.ac.kr



Joints	PD gains k_s (N/rad)	k_d (Ns/rad)	Body segments	Length (m)	Mass (Kg)	Inertia (Kg m^2)
			head	0.210	3	0.011
neck	3000	60	upper arm	0.360	2	0.022
shoulder	4000	80	lower arm	0.320	1	0.009
elbow	3000	60	torso	0.460	10	0.176
hip	4000	80	thigh	0.455	7	0.121
knee	4000	80	shin	0.431	5	0.077
ankle	4000	80	foot	0.240	4	0.019

Figure 1: The planar biped dynamic model has 12 body links connected by 11 revolute joints. The lower-body of the model with six joints is used in some examples. The ground reaction is modelled as a damped spring. The ground spring and damping coefficients are $k_s = 27000N/m$ and $k_d = 675Ns/m$, respectively.

and control human characters interactively [Arikan et al. 2003; Lee et al. 2002; Kovar et al. 2002]. These data-driven techniques have been successfully applied to the animation of biped human behaviors such as locomotion [Sun and Metaxas 2001], jumping [Liu and Popović 2002] and balancing [Arikan et al. 2005; Yin et al. 2005], to name just a few. Zordan et al. [2005] suggested a character animation technique for allowing transition between motion capture data playback and physically based simulation.

In order to generate the physically correct motion of simulated characters and humanoid robots, dynamic control systems can be hand-designed. Hodgins and her colleagues [1995; 1997] designed dynamic controllers for a variety of human athletic behaviors and explored optimization techniques for automatically adapting a controller designed for one character to work on another character. van de Panne and his colleagues have explored various controller design methods. Laszlo et al. [1996] employed limit cycle control to stabilize open-loop trajectories for biped locomotion. They later proposed an interactive technique that allows the user to control planar biped figures capable of walking, running, and performing various gymnastic behaviors [Laszlo et al. 2000]. Faloutsos et al. [2001] explored a method of creating a composite controller that allows a human character to transition from one motor skill to another. Sharon and van de Panne [2005] synthesized planar bipedal walking controllers that mimic the style of a kinematic target trajectory. Yin et al. [2007] have further elaborated biped controllers to generate a variety of gaits that are resilient to unexpected pushes, steps, and slopes. Many proposed methods use the zero moment point (ZMP) to define a physically plausible motion trajectory and adapt the motion to pushes [Dasgupta and Nakamura 1999; Kajita et al. 2003; Komura et al. 2004; Oshita and Makinouchi 2001; Tak et al. 2000].

Dynamic motion synthesis has often been formulated as a variational optimization problem. Popović and Witkin [1999] adopted spacetime constraints and variational optimization techniques for transforming an existing character motion in such a way that it achieves new constraints while preserving its essential physical properties. Liu et al. [2005] estimated "style" parameters of captured human locomotion data using inverse optimization. Pollard and her colleagues explored practical techniques that make spacetime optimization attainable by adopting weighted joint accelerations as optimization criteria [Fang and Pollard 2003], instead of joint torques that are more difficult to optimize, or by projecting the optimization problem into a low-dimensional space [Safonova et al. 2004]. Yamane and Nakamura [2000] proposed a dynamic filter that transforms a physically inconsistent motion into a plausible one through an optimization process. Sulejmanpasić and Popović [2005] presented an optimization technique for adapting performed ballistic motion to satisfy new kinematic and dynamic constraints. Optimal motion synthesis has also been exercised in biomechanics [Anderson and Pandy 2001]. The approaches using a spacetime framework commonly optimize the moving trajectories of the character, but usually do not generate its control policies. Therefore, the optimized trajectories may not be reproduced via forward dynamic simulation. We are interested in a different problem of creating dynamic controllers that control self-actuating articulated figures in a simulated environment.

Motion sets	# of clips	# of frames	Time (second)
WalkNormal	5	1017	33.9
WalkSneaky	4	1698	56.6
WalkAzuma	6	1040	34.7
WalkSoldier	6	1473	49.1
WalkLean	5	1193	39.8
WalkHandWave	4	647	21.6
WalkFast	4	627	20.9
WalkBackward	5	1263	42.1
Run	6	1002	33.4
PushForward	3	2617	87.2
PushBackward	2	1942	64.7
Jump	4	2886	96.2

Table 1: The motion sets collected for our experiments.

A variety of machine learning techniques have been adopted for controlling dynamic motions of synthetic characters [Hertzmann 2004]. Among those techniques, imitation-based learning is a promising method that allows physically simulated virtual characters and humanoid robots to imitate skillful human motion [Schaal et al. 2003]. A straightforward approach of imitation-based learning is to record motion data from a live actor using motion capture and then allow simulated characters to track the recorded joint trajectories using Proportional Derivative (PD) control [Safonova et al. 2003]. This simple PD control-based approach works well with stably balanced target motions, but would fail to track biped motions that require subtle control for maintaining balance. Once the simulated character loses its balance while tracking, simple PD servos cannot recover the character's balance. To avoid this problem, PD control is sometimes combined with a hand-designed, designated feedback balance controller that governs the lower extremities [Nakaoka et al. 2003; Zordan and Hodgins 2002]. Nakanishi et al. [2004] modeled 5-link planar biped locomotion as rhythmic movement primitives and estimated their parameters to imitate human demonstration. Loken [2006] used locally weighted learning for 3-link and 5-link planar biped locomotion.

3 Data Collection and Processing

Our system learns the dynamic behavior of a planar biped dynamic model from a collection of motion data. Our dynamic model has 11 revolute joints (see Figure 1). The dynamic simulation is generated using Open Dynamics Engine (ODE) [Smith 2006]. The mass of each body part is determined using statistical data [AIST Human Body Properties Database 2006]. The foot/ground contact is modeled as a spring-damper model. Our system monitors the heel and toe points and activates damped springs between the ground and the points if the points are within a certain threshold from the ground.

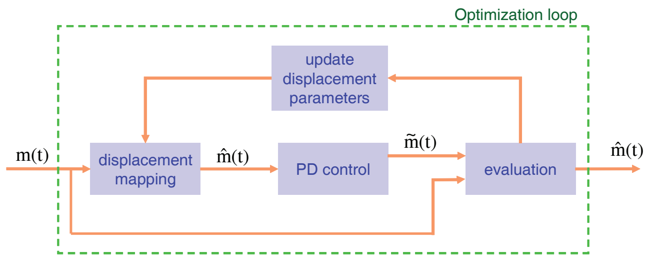


Figure 2: The overview of our optimization process

All of the motion data used in our experiments were captured from a Vicon optical system with sixteen cameras at the rate of 120 frames/second and then down-sampled to 30 frames/second. Each motion data was originally captured for a three-dimensional skeleton with 19 ball-and-socket joints (63 degrees of freedom including the position and orientation of the root segment) and then adapted for our planar dynamic model using hierarchical displacement mapping [Lee and Shin 1999]. In order to adapt the motion of a three-dimensional skeleton for a planar figure, we need to annotate body-environment contact information at each frame. A body segment and the ground surface are considered to be in contact if any joint adjacent to the segment is sufficiently close to the ground and its velocity is below some threshold. The position and orientation trajectories of the segment in contact establishes variational constraints to avoid foot sliding. Hierarchical displacement mapping solves the problem of adapting motion subject to the constraints.

In motion capture sessions, we collected a variety of human motions that can be easily adapted for a planar character (see Table 2). To create the motion library, our subject walked in various styles, walked backward, jogged, and jumped. In order to capture the ways our subject maintains his balance against external force, we pushed him on the chest and the back repeatedly and recorded him reacting to such pushes without falling down. He usually took one or two steps to recover his balance and then stepped back to an upright stand position.

4 Motion Rectification

Our system learns the active behavior of our dynamic figure model from a set of motion data captured from a live actor. A simple tracking method such as PD control cannot make the dynamic model to imitate the recorded motion because the dynamic model is physically different from the actual human skeleton. The dynamic model has fewer degrees of freedom and consists of rigid links and ideal revolute joints, whereas the human body has deformable muscles, skin and much more complicated joints. Measuring the precise mass and inertia of body segments of a live human is very difficult. Therefore, we determined the physical properties of the dynamic model somewhat arbitrarily based on statistical data. The physical imprecision of the dynamic model is a major difficulty of learning dynamic behaviors from motion capture data.

In this section, we propose an optimization method for rectifying the motion data in order to compensate for the inherent imprecision of the dynamic model. Given a segment of motion data, the goal of the optimization is to allow the biped dynamic model to track the motion while maintaining its balance. This type of variational optimization problems are notorious for their high-dimensionality and having highly non-linear objective functions. We observed the dynamic figure model tracking target motions with simple PD servos. The dynamic model failed to track target motions mostly when it loses its balance due to a lack of feedback control, when a swinging foot accidentally touches the ground surface, and when the controller could not produce enough thrusts to keep it moving along desired trajectories. Such problems can be easily fixed by slightly modifying the target motions. For example, balance control can be achieved by a slight tilt of the upper body, accidental ground contact can be avoided by lifting up the swing foot trajectory, and stronger thrusts can be produced by stretching out the stance foot further when it pushes down against the ground. From this observation, our optimization algorithm refines the target motion kinematically in such a way that simple PD servos allow the tracking of the motion (see Figure 2).

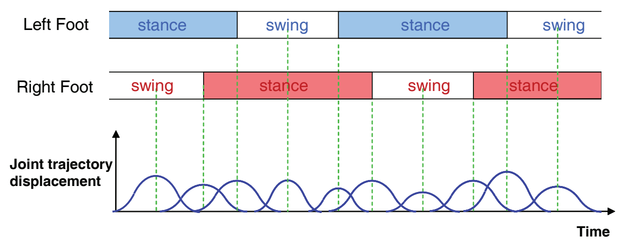


Figure 3: The displacement map is represented as a weighted sum of sinusoidal basis functions. The basis functions are distributed along the time axis synchronized with stance and swing phases.

4.1 Formulation

Given input motion data $\mathbf{m}(t) = (\theta_1(t), \cdots, \theta_n(t))$, the rectified target motion is represented using a motion displacement map such that $\hat{\mathbf{m}}(t) = \mathbf{m}(t) + \mathbf{d}(t)$, where n is the number of joints and $\mathbf{d}(t) = (d_1(t), \cdots, d_n(t))$ is an array of motion displacements. Each displacement map is represented in a parametric form with bell-shaped basis functions:

$$d_i(t) = \sum_{j=1}^{m} h_{ij} B_j(t; c_j, w_j), \tag{1}$$

where m is the number of node points and sinusoidal basis function

$$B_j(t; c_j, w_j) = \begin{cases} \frac{1}{2}\left[1 + \cos\left(\frac{\pi}{w_j}(t - c_j)\right)\right], & \text{if } c_j - w_j < t < c_j + w_j \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

has a peak at node point c_j and extends from $c_j - w_j$ to $c_j + w_j$. The function value is zero outside the interval. A careful selection of node points can improve the optimization performance significantly. In our experiments, node points $\{c_1, \cdots, c_m\}$ are spaced non-uniformly and each node point is coincident either with the takeoff and kickdown of a stance foot or with the halfway of a swing phase (see Figure 3). Our optimization algorithm takes $\mathbf{m}(t)$ as input and determines a set of coefficients $(h_{11}, \cdots, h_{nm}, w_1, \cdots, w_m)$ of motion displacements.

Joint PD control. Our optimization algorithm updates the coefficients of motion displacements iteratively and suggests an intermediate target motion $\hat{\mathbf{m}}(t)$ at every step of the iteration. We use PD servos to track a target motion starting from the first frame of the original input motion. The control equation for each joint is

$$\tau = k_s(\theta_d - \theta) + k_v(\dot{\theta}_d - \dot{\theta}) \tag{3}$$

where θ is the angle of the joint, θ_d is the desired angle, $\dot{\theta}$ is the angular velocity of the joint and $\dot{\theta}_d$ is the desired angular velocity.

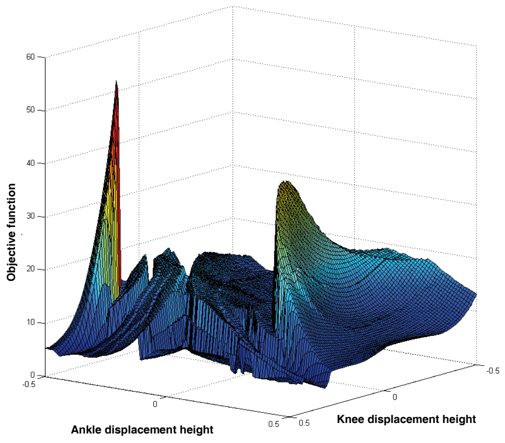


Figure 4: The objective function used to optimize motion data has many local minima. This plot shows a sampling of the objective function as two of the displacement parameters are varied. These two parameters correspond to the heights of displacement maps for the right ankle and the right knee. The narrow, curvy valleys in the objective function make the optimization challenging.

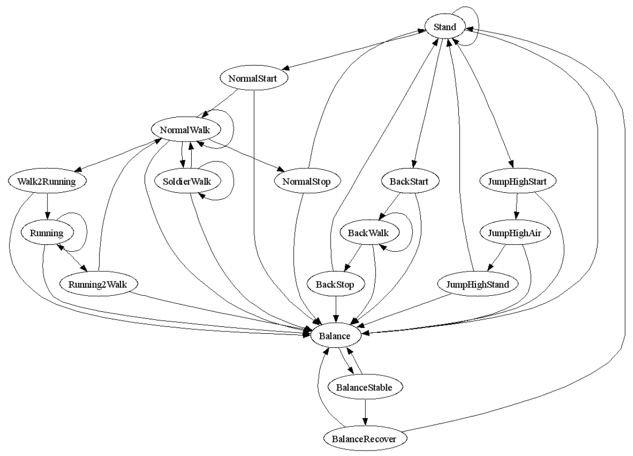


Figure 5: An example of finite state machines that govern transitioning between motor controllers. The nodes correspond to motor controllers and the edges show the transition possibilities between controllers.

k_s and k_v are the proportional and derivative gains. Tracking the target motion using PD servos produces a simulated motion $\tilde{\mathbf{m}}(t)$

Objective function. We expect the simulated motion $\tilde{\mathbf{m}}(t)$ to imitate the input motion as closely as possible. This goal is achieved by minimizing the objective function:

$$E = \int_0^T \left((\frac{t}{T})^2 + c \right) \operatorname{diff}\big(\mathbf{m}(t), \tilde{\mathbf{m}}(t)\big) dt, \tag{4}$$

which penalizes the discrepancy between the original and the simulated motions. The pose difference $\operatorname{diff}\big(\mathbf{m}(t), \tilde{\mathbf{m}}(t)\big)$ at a time instance is measured by comparing mass points on the skeleton in two poses. We assume that only the end points of body links have masses for distance computation. Two sets of mass points obtained from $\mathbf{m}(t)$ and $\tilde{\mathbf{m}}(t)$ are first normalized in such a way that they have the same average in x-coordinates, and then their difference is computed as the squared sum of Euclidean point distances multiplied by their masses. The weight term $\big((t/T)^2 + c\big)$ allows the optimized motion to deviate from the target trajectory in the short term in order to better follow the target trajectory at the end of the motion.

4.2 Optimization Method

The objective function has many local minima (see Figure 4). In order to find the minimum of the objective function, we randomly choose initial parameter values and run a downhill simplex method repeatedly with different initial parameter values to find a local extremum. The minimum among resultant local extreme values is selected as a solution. The downhill simplex method is very slow, but also known to be extremely robust in some cases [Press et al. 2002]. We also tested with other well-known optimization methods, such as Powell's method and a conjugate gradient method, which are presumably much faster than the downhill simplex method in most applications. In an informal evaluation of optimization techniques, the downhill simplex method appeared to converge more robustly than the other methods on our search space, which has a lot of narrow, curvy valleys to descend.

We preferred to record motion data in long clips in order to capture natural transitions between behaviors. The optimization process could be extremely slow and prone to diverge with a long segment of motion data because the dimensionality of the search space increases in proportion to the number of nodes sampled in the time axis. Inspired by spacetime windows [Cohen 1992], we use an incremental technique for rectifying a long clip of motion data efficiently. The basic idea is to separately optimize motion frames in a window that covers a small portion of the long sequence. This window shifts along the time axis in order to allow motion frames to be refined incrementally from the start to the end of the motion. More specifically, the size of the window is initially determined in such a way that it covers the support intervals of the first k basis functions ($k = 2$ or 3 in our experiments). Once the motion frames in the window are optimized, the window is shifted by dropping the first $k-1$ basis functions and including $k-1$ successive basis functions in the window interval. In this way, the shifted window has an overlap with the previous window in order to maintain temporal coherence across the window boundaries. The entire optimization process rectifies the frames in the window and shifts the window repeatedly until the window arrives at the end of the motion.

Practical implementation. Finding the optimal solution of a variational optimization problem is extremely demanding in computation. We use a couple of pragmatic techniques to accelerate the optimization process. Reducing the dimension of the search space is a common idea of practical optimization. In our experiments, the joints in the upper body are controlled by PD servos, but do not actively adjust the target trajectories to keep balance. This excludes the degrees of freedom in the upper body from the optimization parameters. The displacement maps are computed only for six joints (hips, knees, and ankles) in the lower body. The other acceleration technique reduces the total number of iteration steps by choosing promising initial configurations. The global optimization process requires the repeated execution of a local optimization method on many different initial parameter values sampled on the search space. A configuration of parameter values is considered to be promising if PD servos track the corresponding target motion closely. We do not run a local optimization method with all initial configurations, but select a few promising configurations to refine them further by optimization. In this way, we achieved significant performance gain without noticeable deterioration in the final result.

5 Behavior Control

The goal of our work is to create dynamically-simulated, interactively-controllable articulated characters equipped with an integrated repertoire of motor skills. A collection of motion data adapted for our character is used to learn such motor controllers as walking, jumping, and balancing. Motor controllers are integrated into a finite state machine that allows both voluntary and involuntary transitioning between controllers (See Figure 5). The user can control the character by specifying a transition to the desired action or applying external force directly to the body of the character. Transitioning to the balance controller is invoked involuntarily when the character loses its balance due to external force.

We build two types of controllers: stationary and transitioning. The stationary controller has a finite region of the state space, in which the actuation of the controller allows the character's state to change within the region if no explicit perturbation is introduced deliberately. Either stationary (such as balancing upright) or periodic (such as walking and jogging) behaviors are learned by stationary controllers. The transitioning controller produces transition motions from one behavior to the other. For example, start-to-walk and walk-to-stop controllers make transitions between the upright standing position and dynamic walking. We will explain how to learn stationary controllers in Section 5.1 and Section 5.2. Transition controller learning will be discussed in Section 5.3.

5.1 Controller Learning

The controller learning is the process of inferring a desired action for any given state from the observation of human motions. The observation is stored as a collection of state-action trajectories. Each state-action pair describes the observation how the actor moved (output pose at time $t + \Delta t$) in what situation (input state at time t). The controller can be considered as a function that takes the state of the character as input and produces a target pose of the character at the next time instance. The output pose is fed into PD servos to actuate the character. Given any novel state of the character, we use a simple regression technique that selects nearby samples in the state space and combines the output poses at those samples to produce a desired output. In order to measure the distance between motion frames, we consider four features of the frames:

- *Joint angles and velocities*: The differences in all of the joint angles and velocities are considered for distance computation. Our dynamic model has six joints in the lower body and five joints in the upper body. The joints in the upper body weighed less than the lower body, thus, were sometimes ignored because the upper body makes a smaller contribution to balancing biped motions than the lower body does.
- *Root position, velocity, and direction*: The height of the root node (where the spine link meets two thigh links) from the reference ground plane and the horizontal and vertical velocities of the root node are included in the feature vector. However, the horizontal coordinate of the root node is ignored so that the distance between frames can be invariant under horizontal translation. The signed direction of the spine link with respect to its upright position is also included in the feature vector.
- *Foot position and velocity*: The feature vector includes the position and velocity of both feet with respect to a local, moving coordinate system. The origin of the coordinate system is fixed at the root node and the direction of the spine link defines the coordinate axes. The foot positions are actually redundant because the root position and joint angles already reflect the foot positions. We include these features for the convenience in adjusting their weights with respect to other features.
- *Ground contact*: Contact with the environment is an important perceptual feature of motion. We use an array of boolean features to describe the ground contact. For bipedal motions, the feature vector includes four boolean (zero or one) values for encoding heel and toe contacts of both feet. The recovery motion from falling over requires more boolean features.

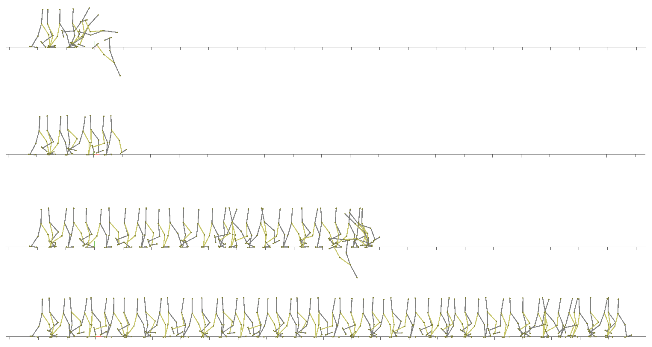


Figure 6: Learning walk controllers. (Top) The original motion data are kinematically retargetted to the two-dimensional dynamic model and then fed into PD servos to simulate. The character falls over in a few steps. (Second row) The motion data are optimized not to fall over. (Third row) The walk controller is learned from a set of walking motion data of a duration of 30 seconds by using a simple regression technique without adaptive refinement. The character walks stably for an extended period of time, but eventually falls over. (Bottom) The final walk controller allows walk cycles to repeat indefinitely over a flat terrain without falling over.

The above four features are weighted, squared, and summed to compute the squared distance between motion frames (see Table 2 for weight values used in our experiments).

Given a feature vector $F(t)$ of the character being simulated, our controller decides the target pose $P(t+\Delta t)$ at the next time instance referring to a regression of training data. The controller searches the k-nearest neighboring samples $\{(F_i, P_i) | i = 1, \cdots, k\}$ and combines the associated poses in the samples with weights inversely proportional to the distances:

$$P(t+\Delta t) = \begin{cases} \frac{\sum_i w_i P_i}{\sum_i w_i}, & \text{if } d_i > \varepsilon \text{ for } \forall i, \\ P_{argmin(d_i)}, & \text{otherwise,} \end{cases}$$

where $d_i = \text{distance}\big(F(t), F_i\big)$, $w_i = \frac{1}{d_i}$, and ε is a small constant. If the distance to any sample is below ε, we select the nearest sample in order to avoid division by zero. In order to locate k-nearest neighbors efficiently, we store data in a kd-tree and search k-nearest neighbors approximately within a small error bound using ANN library [Mount and Arya 2006].

5.2 Adaptive Refinement

The controller thus obtained provides no guarantee that it will produce stationary cycles without falling over (see Figure 6 (third row)). The performance of the controller depends significantly on the diversity and distribution of the training data. It is very difficult to acquire training data that guarantees the desired quality of the controller.

We solve this problem by adaptively refining the controller. The basic strategy is to test the controller by running it on the simulator. If the character falls over or deviates excessively from the training data set, we add a new state-action trajectory to the training set so that the character can avoid the unsuccessful situation. An abnormal (possibly leading to falling over) situation during simulation can be detected by monitoring the distance to the nearest sample in the training set. If the distance between the character's state and its nearest neighbor in the training data is above a user-specified threshold for a certain period of time (5 frames in our experiments), we decide that the character's state is not recoverable to stable cycles using the current controller.

Once an unsuccessful situation is detected, we roll back the whole system in time to the moment when the nearest cycle of motion begins. In order to avoid the unsuccessful situation, the controller has to make a different output at that moment. To do so, we synthesize a new sample trajectory by warping existing data. From the training set, we select a cycle of motion that starts with the configuration nearest to the character's current state. This motion trajectory is warped kinematically in such a way that the motion starts with the character's current state and blends smoothly back to its original trajectory using displacement mapping. We run the optimization method explained in the previous section to make the warped trajectory physically-feasible, and then add this new trajectory back to the training set. The newly added trajectory will guide the character back to a stable cycle at the regression process. In this way, we can refine the controller incrementally until no more failure is observed.

5.3 Transition Control

Given two stationary (or cyclic) controllers, we can build a transition controller between them in the same way as we build the stationary controllers if an adequate training data set is available. For example, we captured our subject starting to walk from an upright standing position, taking six steps, and stopping to the standing position, repeatedly. The first and the last two steps of the walk data were used to learn stand-to-walk and walk-to-stand transition controllers, respectively. The walk cycle controller was learned from the intermediate two steps of the training set. However, acquiring a large collection of transitioning motions between every pair of simulated behaviors is often tedious and laborious. In case of lacking adequate training data, we can synthesize training data by blending existing motions kinematically. For example, the walk-to-run transition sample can be synthesized by blending a cycle of walking motion and a cycle of runing motion. We blend them simply by fading one in while fading the other out. Over the fading duration, dynamic time warping is used to find the correspondence between two motions and a sinusoidal transition function is used to blend joint angles smoothly. Since joint angle blending may cause feet to slide, we enforce foot contact constraints using hierarchical displacement mapping [Lee and Shin 1999].

Transitioning control at run time is straightforward. Assume that the character is currently controlled by a stationary controller and intends to change its behavior via a transition controller. While being simulated, the character monitors the distance between its current state and the nearest sample in the transition controller's training set. If the distance is below a certain threshold, the transition controller gains control over the motion of the character. Transitioning from a transition controller to a stationary controller can be handled similarly. The transition could fail if the training set of the participating transition controller is not sufficient. Whenever a failure is detected, the transition controller can be refined adaptively as explained in Section 5.2.

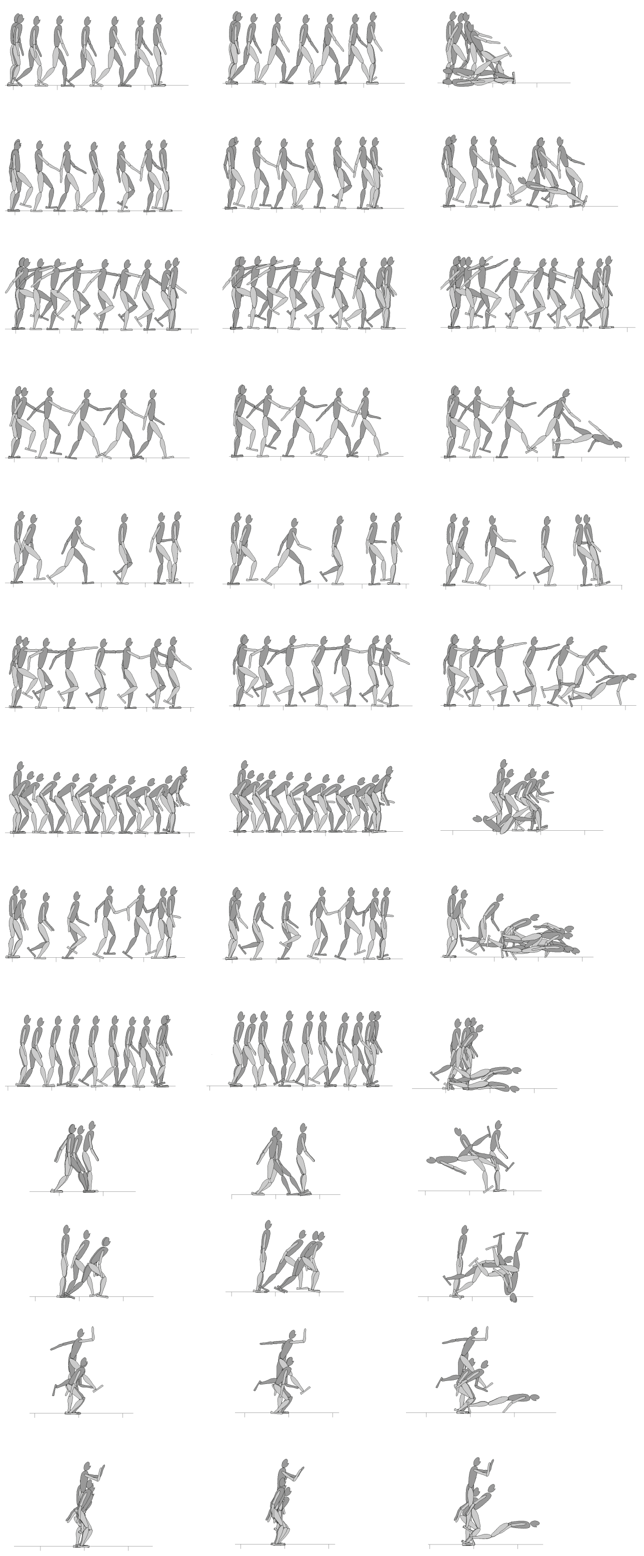


Figure 7: The results of motion optimization. (Left column) Original motion capture data. (Center column) Rectified motions are simulated by PD controllers. (Right column) Original motions are simulated by PD controllers. Most of the data result in falling down. (From top to bottom) WalkNormal, WalkLean, WalkSoldier, WalkHandWave, WalkFast, WalkAzuma, WalkSneaky, Run, WalkBackward, PushBackward, PushForward, Jump, HighJump

Controllers	# of frames	KNN	Feature weights			
			POS	TOR	ANG	ROO
Stand	50	6	15	2	0.1	1
Balance	790	3	15	2	15	1
BalanceStable	200	3	15	2	15	1
BalanceRecover	1200	3	15	2	15	1
NormalStart	164	6	15	2	0.1	1
NormalWalk	410	6	15	2	0.1	1
NormalStop	186	6	15	2	0.1	1
BackWalkStart	129	5	15	2	0.1	1
BackWalk	479	5	15	2	0.1	1
BackWalkStop	143	5	15	2	0.1	1
SoldierStart	182	6	15	2	0.1	1
SoldierWalk	655	6	15	2	0.1	1
SoldierStop	182	6	15	2	0.1	1
Walk2Run	305	6	15	2	0.1	1
Run2Walk	227	6	15	2	0.1	1
RunStart	158	6	15	2	0.1	1
Run	301	6	15	2	0.1	1
RunStop	203	6	15	2	0.1	1
SlowStart	136	6	15	2	0.1	1
SlowWalk	537	6	15	2	0.1	1
SlowStop	173	6	15	2	0.1	1
JumpStart	162	2	15	2	0.1	4
JumpAir	19	2	15	2	0.1	4
JumpStand	173	2	15	2	0.1	4

Table 2: Motor controllers built in our experiments. The size of the neighborhood was determined empirically. The controllers are learned from motion capture data except Walk2Run and Run2Walk transition controllers, whose training data were kinematically synthesized by blending walking and running data. KNN is *k*-nearest neighbors. The feature weights from left to right represent (POS) feet position, (TOR) torso up direction, (ANG) joint angles, (ROO) The y-position and y-velocity of the root. The weights for the other features are one.

6 Experimental Results

The timing data provided in this section was measured on a 2.8GHz Intel Pentium 4 computer with 1Gbyte main memory.

Optimization. The number of initial configurations sampled for global optimization is important for the convergence and performance of our optimization algorithm. Rectifying a single walk cycle required us to sample several hundreds of initial configurations in order to avoid getting stuck at local minima. Encouragingly, running the local optimizer for a few promising (filtered by running on PD controllers) configurations produced almost the same result as running the local optimizer for all sampled configurations. The performance can further be improved by using heuristic rules for sampling initial configurations. For example, the height of a swing foot from the ground is often important for optimizing walking motions. In that case, we can sample initial configurations as the height of the swing foot is varied using an inverse kinematics solver. The performance of optimizing an extended sequence of motion data can be significantly improved by using the incremental optimization method explained in Section 4.2. For capturing each segment of walking motion data, our subject started from a standing position, walked for five to six steps (as far as allowed in the motion capture region), and stopped to a standing position. The dimension of the search space for such data is prohibitively high, and we are unable to make the optimization converge in such high-dimensional space without the use of the incremental method. The computation cost for incremental optimization increases linearly with the duration of motion data. Our system took about 15 minutes for rectifying a segment of walking data that include start, six steps of walk and stop motions (see Figure 7 (top row)).

Walk. The "NormalWalk" controller shown in Figure 6 was learned from walk data that correspond to 20 walk cycles. The adaptive refinement of the controller added six walk cycles to the training data in order to make the controller repeat indefinitely without falling over. "NormalStart" and "NormalStop" controllers did not require adaptive refinements. Similarly, walking in various styles were learned and simulated.

Balance. The balance controllers were learned from training data that include about 30 pushes and corresponding reactions. The balance behavior we captured includes a series of actions: being pushed, stepping out, and recovering. Many character poses while being pushed are quite similar to its poses while recovering back to an upright stand position. If these similar poses heading opposite directions are mixed and chosen in the *k*-nearest neighborhood at a state, the character could be immobilized at the state. Each individual controller can accommodate a single, relatively-simple behavior. In order to avoid such problems, we designed three subcontrollers that are invoked in succession. When the character loses its balance, the "Balance" controller is invoked involuntarily in order to regain balance by repositioning feet. The control can also be switched to the balance mode in immediate response to a push if its magnitude is above a certain threshold. This implementation decision was made to imitate the actual behavior of our motion capture subject. "BalanceStable" gains control if the character's pose is stabilized. This controller actually does nothing but notifying the system that the character is ready to return to a stand position. The training data of this controller includes statically balanced character poses that are sampled between being pushed and recovering phases. Then, the system invokes "BalanceRecover" that controls the character to a stand position.

Jump. We capture three types of jump motions, five trials for each type. Controlling jump motions also required three controllers that are invoked in succession. "JumpStart" makes the character jump upwards into the air. "JumpAir" notifies that the character is at the highest position and passes control to "JumpStand", which controls the character to land safely.

Transitioning. Transitioning between controllers at (maybe dynamically balanced but) statically unstable states is an important issue in biped dynamic simulation. In our experiments, changing styles while walking can be accomplished without much care, probably because the stable domains of walk controllers have a good overlap. Transitioning between walking and running required designated transition controllers. We build the training sets for transition controllers in a generate-and-test manner. We randomly select one cycle of walking and one cycle of running from existing data sets and then blend them to produce a training example. This new example is added to the training set, which is evaluated on our simulator. If the performance is good, we just stop there. Otherwise, we repeat the generate-and-test step. In this way, our "Walk2Run" controller was learned from 8 sample trajectories and "Run2Walk" was learned from 6 sample trajectories.

7 Discussion

Our contribution is twofold. We have presented an optimization method that transforms any (either motion-captured or kinematically synthesized) biped motion into a physically-feasible, balance-maintaining simulated motion. Our optimization method allowed us to collect a rich set of training data that contain stylistic, personality-rich human behaviors. We have also presented a controller learning algorithm that facilitated the creation and composition of dynamic controllers. Our controllers have a simple mechanism that exploits the essential information in training data. Remarkably, our controllers learned subtle details of biped balance control from pure kinematic sample trajectories without feedback control.

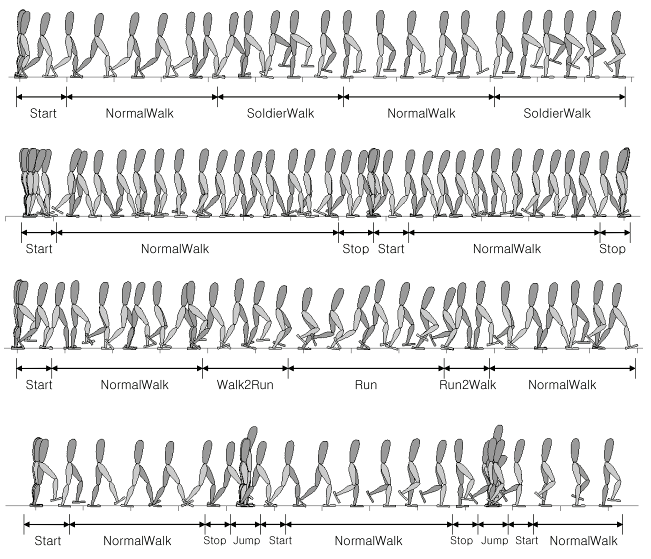


Figure 8: Transitioning between controllers

The robustness of our controllers is largely depending on the size and diversity of training data. It is certainly very difficult to build robust controllers if the training data are severely ill-sampled. Even with well-sampled training data, building a robust controller is challenging. We evaluate the robustness of our controllers by simply running the simulator and waiting overnight to see if it falls over. The adaptive refinement method certainly improves the robustness of controllers. However, the robustness cannot be guaranteed because all plausible situations in the state space cannot be examined comprehensively for adaptive refinement. In practice, each individual controller can be made quite robust if no external force is applied. However, our controllers could easily break down if external force beyond a certain range is applied.

The memory and computation costs of our regression-based learning algorithm increase with the amount of training data. The memory cost is not generally a problem, because it increases linearly with the size of training data. The computation cost is more serious, because the controller performs a neighborhood search for regression at run time. We circumvented this problem by maintaining data in a *k*d-tree, which facilitates efficient spatial query processing.

Each controller was learned from a small set of sample trajectories with respect to the dimensionality of our dynamic model. In our experiments, each behavior of a biped model with six joints (nine degrees of freedom in total including the position and orientation of the body) was learned from 5 to 20 sample trajectories, which are exceptionally small training data in most machine learning applications. It has been possible probably because human motion is highly coordinated at the joint level. Provided that the intrinsic dimension of human motion is much lower than its ostensible dimension, dimensionality reduction by PCA might be a good idea for faster optimization and effective learning, as accomplished by Safonova et al. [2004].

We employed a very simple regression method that performed well on our controller learning problem. We also tested with LWLR (Locally Weighted Linear Regression) and LWPR (Locally Weighted Projection Regression), which presumably provide a better, smoother regression than the simple method used in our work. In our experiments, the simplest method worked better on our learning problem than more sophisticated methods, probably because our training sets were too small to cope with that level of sophistication.

Both dynamic simulation and controller learning are notorious for their sensitivity to the selection of parameter values. Our method is no exception. We have an array of parameters that require careful tuning for successful simulation and learning. Such parameters include ground reaction and friction coefficients for dynamic simulation, the proportional and derivative gains of PD servos, the relative weights of features of the character's motion, and the number of neighbors (*k*-nearest neighbors) chosen for regression. Parameter tuning is not extremely difficult, but requires appreciable time and efforts. A promising aspect was that we were able to create all the examples in this work using a single configuration of physical parameters. Once a stable configuration of parameters were found, we did not have to tune physical parameters any more. Learning parameters depend on the size and diversity of training data. It might be possible to search a feasible configuration of physical and learning parameters automatically using approximate inverse optimization [Liu et al. 2005].

There are exciting avenues for future work. We would like to generalize our methods to control three-dimensional simulated characters and humanoid robots. Presumably, learning three-dimensional behaviors would require a larger collection of training data and more sophisticated control and learning algorithms. Adapting a dynamic controller for new characters and new geographical and physical environments is a fascinating, yet very challenging task [Hodgins and Pollard 1997]. We anticipate that this task can be done in our framework by adapting training data for new characters and environments kinematically [Lee and Shin 1999] and learning a new controller from new training data.

Acknowledge

We would like to thank all the members of the SNU Movement Research Laboratory for their help in collecting motion data. This research work was supported by the IITA under the ITRC program. We thank anonymous reviewers for their suggestions.

References

AIST HUMAN BODY PROPERTIES DATABASE, 2006. http://www.dh.aist.go.jp/bodydb.

ANDERSON, F., AND PANDY, M. 2001. Dynamic optimization of human walking. *Journal of Biomechanical Engineering 123*, 381–390.

ARIKAN, O., FORSYTH, D. A., AND O'BRIEN, J. F. 2003. Motion synthesis from annotations. *ACM Transactions on Graphics (SIGGRAPH 2003) 22*, 3, 402–408.

ARIKAN, O., FORSYTH, D., AND O'BRIEN, J. 2005. Pushing people around. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 59–66.

BRUDERLIN, A., AND CALVERT, T. W. 1989. Goal-directed, dynamic animation of human walking. In *Computer Graphics (Proceedings of SIGGRAPH 89)*, vol. 23, 233–242.

COHEN, M. F. 1992. Interactive spacetime control for animation. In *Proceedings of SIGGRAPH 92*, 293–302.

DASGUPTA, A., AND NAKAMURA, Y. 1999. making feasible walking motion of humanoid robots from human motion capture data. In *Proceedings of IEEE Intl. Conference on Robotics and Automation (ICRA)*, 1044–1049.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composable controllers for physics-based character animation. In *Proceedings of SIGGRAPH 2001*, 251–260.

FANG, A. C., AND POLLARD, N. S. 2003. Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics (SIGGRAPH 2003) 22*, 3, 417–426.

HERTZMANN, A., 2004. Introduction to bayesian learning, siggraph course notes.

HODGINS, J. K., AND POLLARD, N. S. 1997. Adapting simulated behaviors for new characters. In *Proceedings of SIGGRAPH 97*, 153–162.

HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., AND O'BRIEN, J. F. 1995. Animating human athletics. In *Proceedings of SIGGRAPH 95*, 71–78.

KAJITA, S., KANEHIRO, F., KANEKO, K., FUJIWARA, K., HARADA, K., YOKOI, K., AND HIRUKAWA, H. 2003. Biped walking pattern generation by using preview control of zero-moment point. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1620–1626.

KOMURA, T., LEUNG, H., AND KUFFNER, J. 2004. Animating reactive motions for biped locomotion. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*, 32–40.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. *ACM Transactions on Graphics (SIGGRAPH 2002) 21*, 3, 473–482.

LASZLO, J., VAN DE PANNE, M., AND FIUME, E. 1996. Limit cycle control and its application to the animation of balancing and walking. In *Proceedings of SIGGRAPH 96*, 155–162.

LASZLO, J., VAN DE PANNE, M., AND FIUME, E. 2000. Interactive control for physically-based animation. In *Proceedings of SIGGRAPH 2000*, 201–208.

LEE, J., AND SHIN, S. Y. 1999. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of SIGGRAPH 99*, 39–48.

LEE, J., CHAI, J., REITSMA, P. S. A., HODGINS, J. K., AND POLLARD, N. S. 2002. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics (SIGGRAPH 2002) 21*, 3, 491–500.

LIU, C. K., AND POPOVIĆ, Z. 2002. Synthesis of complex dynamic character motion from simple animations. vol. 21, 408–416.

LIU, C. K., HERTZMANN, A., AND POPOVIC, Z. 2005. Learning physics-based motion style with nonlinear inverse optimization. *ACM Transactions on Graphics (SIGGRAPH 2005) 24*, 3, 1071–1081.

LOKEN, K. 2006. *Imitation-based Learning of Bipedal Walking Using Locally Weighted Learning*. Master's thesis, Computer Science Department, The University of British Columbia.

MOUNT, D., AND ARYA, S., 2006. Ann: Library for approximate nearest neighbor searching, http://www.cs.sunysb.edu/ algorith/implement/ann/distrib/index1.html.

NAKANISHI, J., MORIMOTO, J., ENDO, G., CHENG, G., SCHAAL, S., AND KAWATO, M. 2004. Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems 47*, 79–91.

NAKAOKA, S., NAKAZAWA, A., AND YOKOI, K. 2003. Generating whole body motions for a biped humanoid robot from captured human dances. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 3905–3910.

OSHITA, M., AND MAKINOUCHI, A. 2001. A dynamic motion control technique for human-like articulated figures. *Computer Graphics Forum (EUROGRAPHICS 2001) 20*, 3, 192–202.

POPOVIĆ, Z., AND WITKIN, A. P. 1999. Physically based motion transformation. In *Proceedings of SIGGRAPH 99*, 11–20.

PRESS, W. H., TEUKOLSKEY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 2002. *Numerical Recipes in C++ (2nd Edition)*. Cambridge University Press.

SAFONOVA, A., POLLARD, N. S., AND HODGINS, J. K. 2003. Optimizing human motion for the control of a humanoid robot. In *Proceedings of 2nd International Symposium on Adaptive Motion of Animals and Machines (AMAM2003)*.

SAFONOVA, A., HODGINS, J. K., AND POLLARD, N. S. 2004. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics (SIGGRAPH 2004) 23*, 3, 514–521.

SCHAAL, S., IJSPEERT, A., AND BILLARD, A. 2003. Computational approaches to motor learning by imitation. *Philosophical Transaction of the Royal Society of London: Series B, Biological Sciences 358*, 537–547.

SHARON, D., AND VAN DE PANNE, M. 2005. Synthesis of controllers for stylized planar bipedal walking. In *International Conference on Robotics and Automation (ICRA 2005)*, 18–22.

SMITH, R., 2006. Open dynamics engine, http://www.ode.org.

SULEJMANPASIĆ, A., AND POPOVIĆ, J. 2005. Adaptation of performed ballistic motion. *ACM Transactions on Graphics 24*, 1, 165–179.

SUN, H. C., AND METAXAS, D. N. 2001. Automating gait animation. In *Proceedings of SIGGRAPH 2001*, 261–270.

TAK, S., SONG, O.-Y., AND KO, H.-S. 2000. Motion balance filtering. *Computer Graphics Forum (Eurographics 2000) 19*, 3, 437–446.

YAMANE, K., AND NAKAMURA, Y. 2000. Dynamics filter - concept and implementation of on-line motion generator for human figures. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 688–695.

YIN, K., PAI, D. K., AND VAN DE PANNE, M. 2005. Data-driven interactive balancing behaviors. In *Pacific Graphics*.

YIN, K., LOKEN, K., AND VAN DE PANNE, M. 2007. Simbicon: Simple biped locomotion control. *ACM Transactions on Graphics (SIGGRAPH 2007) 26*, 3.

ZORDAN, V. B., AND HODGINS, J. K. 2002. Motion capture-driven simulations that hit and react. In *Proceedings of ACM SIGGRAPH Symposium on Computer Animation*, 89–96.

ZORDAN, V. B., MAJKOWSKA, A., CHIU, B., AND FAST, M. 2005. Dynamic response for motion capture animation. *ACM Transactions on Graphics (SIGGRAPH 2005) 24*, 3, 697–701.