Simulating 2D Gaits with a Phase-Indexed Tracking Controller

Yeuhi Abe ▪ *MIT*

Jovan Popović ▪ *Adobe*

Current time-indexed trackers lack robustness because they adhere strictly to the input motion's timing. A proposed phase-indexed tracker deviates from that timing and can withstand larger force perturbations than nonlinear quadratic regulators. The tracker also induces a reduced dynamics that can be used to design robust control policies that incorporate prediction.

Despite physical simulation's advantages, it's often considered an overly complex or obtuse way to animate characters. This is partly because existing character controllers require tuning esoteric parameters with little apparent connection to the motion they produce. Animators have difficulty tuning these parameters to produce a desired motion in a particular style.

Tracking controllers, a new breed of physical controller, look like a promising way to make controller design more intuitive. They require only a single motion as input. This approach automatically computes the control outputs for the character, such that the character mimics the input motion as closely as possible. Unfortunately, tracking controllers tend to lack robustness to even small perturbations from the input motion. Making robust tracking controllers has been an ongoing challenge.

Here, we propose a 2D tracking controller that improves robustness by allowing for variations in the timing of tracked motions. Previous tracking methods lack robustness partly because they adhere too rigidly to the input motion's original timing. For example, consider a character that's pushed from behind while walking. Previous methods would try to match the step's original timing, whereas a more robust reaction would be to take a quick step to regain balance. Owing to our controller's time-invariance, we call it a *phase-indexed tracking controller*, as opposed to previous controllers, which are best described as time-indexed.

To demonstrate phase-indexed tracking, we developed a walking controller that exhibits robustness to unanticipated terrain and to force perturbations while mimicking an input motion's style (see Figure 1). Our approach can withstand external force perturbations an order of magnitude larger than the state-of-the-art, time-indexed controller based on the nonlinear quadratic regulator (NQR).

At the heart of our controller are motion constraints. One of their benefits is that they induce a reduced dynamics we can use to design control policies that incorporate prediction. Prediction lets controllers take into account current actions' consequences, which is important for controlling characters that exhibit lifelike reaction. (For more information on biped controllers in animation, see the related sidebar.)

Phase-Indexed Tracking

A phase-indexed tracking controller mimics an input motion's joint configurations while allowing the overall timing to deviate. To do this, it first identifies in the input motion a state variable that's monotonic with respect to time. The controller uses motion constraints to synchronize the character's remaining degrees of freedom (DOF) with this variable, as opposed to time. For example, for walking, the controller synchronizes the character's internal DOF with the swing leg's angle (see Figure 2). This produces a robust gait that mimics the input motion's style.

Character Dynamics

First, we must describe our character's dynamics.

Published by the IEEE Computer Society

Biped Controllers in Animation

The simulation and control of biped characters has long been an area of interest for animation research. Some of the most successful biped controllers relied on procedural feedback laws to produce motion.[1–3] However, tuning the parameters of these controllers so that they produce a desired motion is difficult. Various automatic approaches exist for tuning the parameters, but they involve long stochastic searches with carefully chosen objective-shaping terms.[4,5] Recently, researchers have proposed a hybrid approach that combines procedural feedback laws with data-driven trajectories.[6]

However, an alternative approach is to directly track a single input motion using a tracking controller. Tracking controllers are easy for animators to use because they don't require tuning of parameters to produce a desired motion. The first tracking controllers used simple proportional-derivative (PD) joint servos that couldn't handle full-body balancing.[7] More sophisticated tracking controllers coordinated the full body to enforce static balance criteria[8,9] but were still inadequate for dynamic motions such as walking and running. Recently, researchers devised tracking controllers incorporating a predictive feedback strategy that successfully handles walking and running.[10,11] However, they lack robustness and tend to fail when confronted with unplanned disturbances in the environment. Evidence suggests that this is due partly to an overly strict adherence to the input motion's original timing.[12]

References

1. I. Mordatch, M. de Lasa, and A. Hertzmann. "Robust Physics-Based Locomotion Using Low-Dimensional Planning," *ACM Trans. Graphics*, vol. 29, no. 3, 2010, article 71.
2. M.H. Raibert and J.K. Hodgins, "Animation of Dynamic Legged Locomotion," *ACM Siggraph Computer Graphics*, vol. 25, no. 4, 1991, pp. 349–358.
3. K. Yin, K. Loken, and M. van de Panne, "Simbicon: Simple Biped Locomotion Control," *ACM Trans. Graphics*, vol. 26, no. 3, 2007, article 105.
4. M. van de Panne and A. Lamouret, "Guided Optimization for Balanced Locomotion," *Proc. 6th Eurographics Workshop Computer Animation and Simulation*, Eurographics Assoc., 1995, pp. 165–177.
5. J.M. Wang, D.J. Fleet, and A. Hertzmann, "Optimizing Walking Controllers," *ACM Trans. Graphics*, vol. 28, no. 3, 2009, article 168.
6. Y. Lee, S. Kim, and J. Lee, "Data-Driven Biped Control," *ACM Trans. Graphics*, vol. 29, no. 4, 2010, article 129.
7. V.B. Zordan and J.K. Hodgins, "Motion Capture-Driven Simulations That Hit and React," *Proc. 2002 ACM Siggraph/Eurographics Symp. Computer Animation* (SCA 02), ACM Press, 2002, pp. 89–96.
8. Y. Abe, M. da Silva, and J. Popović, "Multiobjective Control with Frictional Contacts," *Proc. 2007 ACM Siggraph/Eurographics Symp. Computer Animation* (SCA 07), ACM Press, 2007, pp. 249–258.
9. A. Macchietto, V. Zordan, and C.R. Shelton, "Momentum Control for Balance," *ACM Trans. Graphics*, vol. 28, no. 3, 2009, article 80.
10. M. da Silva, Y. Abe, and J. Popović, "Interactive Simulation of Stylized Human Locomotion," *ACM Trans. Graphics*, vol. 27, no. 3, 2008, article 82.
11. U. Muico et al., "Contact-Aware Nonlinear Control of Dynamic Characters," *ACM Trans. Graphics*, vol. 28, no. 3, 2009, article 81.
12. Y. Ye and C. Karen Liu, "Optimal Feedback Control for Character Animation Using an Abstract Model," *ACM Trans. Graphics*, vol. 29, no. 4, 2010, article 74.

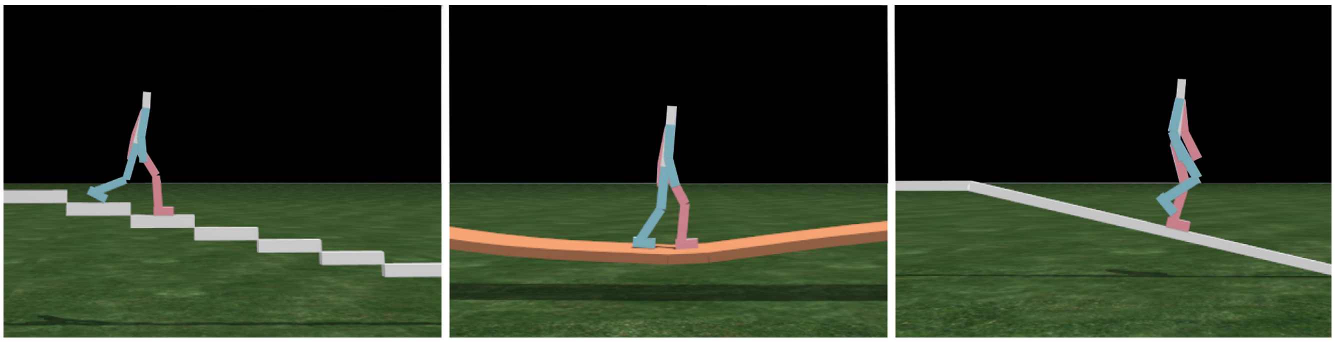
Figure 1. A phase-indexed tracking controller adapts robustly to new terrain.

For both control and forward simulation, we model our character as a collection of rigid bodies adjoined by joints. We describe the configuration by a vector of joint angles $\boldsymbol{q} \in \mathcal{Q}$. The controller provides a vector of joint forces $\boldsymbol{u} \in \mathcal{U}$. We then determine the character's motion by integrating a standard set of motion equations.[1]

Here, we write the equations in a decoupled form that explicitly delineates the root coordinates $\boldsymbol{q}_r$ (and forces $\boldsymbol{u}_r$) from the internal DOF $\boldsymbol{q}_a$ (and forces $\boldsymbol{u}_a$):

$$\begin{bmatrix} \boldsymbol{M}_r & \boldsymbol{M}_{ra} \\ \boldsymbol{M}_{ra}^{\mathsf{T}} & \boldsymbol{M}_a \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}}_r \\ \ddot{\boldsymbol{q}}_a \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_r \\ \boldsymbol{u}_a \end{bmatrix}.$$

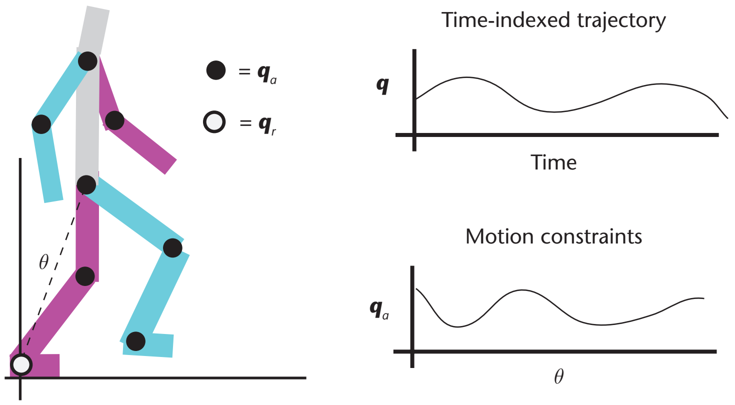

Figure 2. Motion constraints enforce a kinematic relationship between a state variable θ and the actuated degrees of freedom (DOF) $\boldsymbol{q}_a$, with $\boldsymbol{q}_r$ indicating the root coordinates. Phased-indexed tracking fits the constraints' parameters to match a specific input motion. The constraints serve a function similar to the trajectories tracked by time-indexed controllers, but without enforcing a specific timing.

In this equation, $\boldsymbol{M}$ is the mass matrix. For forward simulation, $\boldsymbol{q}_r \in \mathbb{R}^3$ describes the 2D character's global rotation and translation, which we assume to be unactuated (that is, $\boldsymbol{u}_r = 0$). We also use the motion equations to derive the feedback in the controller.

The character dynamics we use depends on the character's contact state with the environment. For example, in the single- or double-support stages of walking, it's easier to assume a fixed connection between the character's stance foot and the ground. (In single support, one foot swings through the air; in double support, both feet touch the ground.) So, for control, we abuse notation and use $\boldsymbol{q}_r \in \mathbb{R}$ to describe the ankle angle of the flat-stance foot.

Motion Constraints

We use motion constraints primarily to derive the feedback that enforces the relationship between the character's internal DOF. However, they have much broader applicability. (For more information, see the "Motion Constraints" sidebar.) Here, we describe motion constraint theory in the most general terms.

Motion constraints rely on an invertible coordinate transformation,

$$\boldsymbol{H}^{-1}(\boldsymbol{q}) = [\boldsymbol{h}_u(\boldsymbol{q})^T, \boldsymbol{h}_a(\boldsymbol{q})^T]^T : \mathcal{Q} \to \mathbb{R}^n,$$

which maps joint angles to a partitioned set of controlled, $\boldsymbol{h}_a \in \mathbb{R}^m$, and uncontrolled, $\boldsymbol{h}_u \in \mathbb{R}^{n-m}$, coordinates. Fortunately, coming up with such transformations is easy; we discuss a simple one useful for bipeds later. We assume the size of $\boldsymbol{h}_a$ is the same as the number of actuated DOF.

We also define a scalar variable $\theta(\boldsymbol{h}_u)$ as a function of the unactuated coordinates $\boldsymbol{h}_u$. The choice of θ will vary depending on the type of motion being tracked, but it is assumed to be monotonic in time. For example, for walking or running motions, a natural choice for θ is the absolute angle of the vector between the stance ankle and the pelvis (see Figure 2).

We define the motion constraints $\boldsymbol{y}$ by the relationship

$$\boldsymbol{y} = \boldsymbol{h}_a - \boldsymbol{c}_w(\theta) = 0,$$

where $\boldsymbol{c}_w(\theta)$ is a parametric function (a spline for our purposes) controlled by parameters $\boldsymbol{w}$.

Because the size of $\boldsymbol{y}$ is the same as the number of actuated DOF, precise control is possible through a partial feedback linearization of the form

$$\ddot{\boldsymbol{h}}_u = \boldsymbol{A}^0_{\boldsymbol{h}_u}\boldsymbol{u}_a + \boldsymbol{A}^1_{\boldsymbol{h}_u}\boldsymbol{u}_r + \boldsymbol{b}_{\boldsymbol{h}_u} \quad (1)$$

$$\ddot{\boldsymbol{y}} = \boldsymbol{A}^0_{\boldsymbol{y}}\boldsymbol{u}_a + \boldsymbol{A}^1_{\boldsymbol{y}}\boldsymbol{u}_r + \boldsymbol{b}_{\boldsymbol{y}}, \quad (2)$$

where $\boldsymbol{A}$ and $\boldsymbol{b}$ are matrix and vector quantities that can be solved for, and $\boldsymbol{A}^0_{\boldsymbol{y}}$ is known to be full rank. Our assumption that $\boldsymbol{u}_r$ is underactuated prevents direct control of $\ddot{\boldsymbol{h}}_u$ and $\ddot{\boldsymbol{y}}$ simultaneously. However, we can stabilize the motion constraint around $\boldsymbol{y} = 0$ by applying feedback of the form

$$\boldsymbol{u}_a = \left(\boldsymbol{A}^0_{\boldsymbol{y}}\right)^{-1}\left(\ddot{\boldsymbol{y}}^* - \boldsymbol{A}^1_{\boldsymbol{y}}\boldsymbol{u}_r - \boldsymbol{b}_{\boldsymbol{y}}\right) \quad (3)$$

$$\ddot{\boldsymbol{y}}^* = -\frac{1}{\varepsilon}k_s\boldsymbol{y} - \frac{1}{\varepsilon^2}k_d\dot{\boldsymbol{y}},$$

where k_s and k_d are gains and ε controls the exponential rate of convergence of $(\boldsymbol{y}, \dot{\boldsymbol{y}})$ to (0, 0). Although we could directly invert $\boldsymbol{A}^0_{\boldsymbol{y}}$, that's not the most efficient way to solve for $\boldsymbol{u}_a$. We can formulate efficient $O(n)$ algorithms similarly to Roy Featherstone's articulated-body method.[1]

A Reduced-Dimension System

When we enforce the motion constraints through controller feedback (when $\boldsymbol{y} = 0$), we can express the state solely in terms of the uncontrolled coordinates $\boldsymbol{h}_u$:

$$\boldsymbol{q} = \boldsymbol{H}(\boldsymbol{h}_a, \boldsymbol{h}_u) = \boldsymbol{H}(\boldsymbol{c}_w(\theta(\boldsymbol{h}_u)), \boldsymbol{h}_u) = \boldsymbol{q}_w(\boldsymbol{h}_u)$$

$$\dot{\boldsymbol{q}} = \frac{\partial \boldsymbol{q}_w}{\partial \boldsymbol{h}_u}\dot{\boldsymbol{h}}_u,$$

where $\boldsymbol{q}_w$ is a map from the uncontrolled coordinates to the full state, with the subscript $\boldsymbol{w}$ denoting the dependence on the motion constraints' parameters. The subset of states that can be represented by $\boldsymbol{q}_w$ (and its tangent space operator $\mathcal{T}$),

$$\{(\boldsymbol{q},\dot{\boldsymbol{q}}) \| (\boldsymbol{y},\dot{\boldsymbol{y}})=0\} \subset \mathtt{TQ} \ni (\boldsymbol{q},\dot{\boldsymbol{q}}),$$

is the *zero-dynamics set*. It's parameterized by the reduced coordinates $(\boldsymbol{h}_u,\dot{\boldsymbol{h}}_u)$.

By substituting Equation 3 into Equations 1 and 2, we write the resulting closed-loop dynamics as

$$\ddot{\boldsymbol{h}}_u = \boldsymbol{f}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q},\dot{\boldsymbol{q}})\boldsymbol{u}_r$$

$$\ddot{\boldsymbol{y}} = \ddot{\boldsymbol{y}}^*(\boldsymbol{q},\dot{\boldsymbol{q}}),$$

where $\boldsymbol{f}$ and $\boldsymbol{g}$ are computable functions of $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$.

Finally, when the constraints have stabilized to the set $(\boldsymbol{y},\dot{\boldsymbol{y}})=0$, we can write the dynamics in a simplified form,

$$\ddot{\boldsymbol{h}}_u = \boldsymbol{f}(\boldsymbol{h}_u,\dot{\boldsymbol{h}}_u) + \boldsymbol{g}(\boldsymbol{h}_u,\dot{\boldsymbol{h}}_u)\boldsymbol{u}_r \quad (4)$$

$$\ddot{\boldsymbol{y}} = 0,$$

which depends only on $(\boldsymbol{h}_u,\dot{\boldsymbol{h}}_u)$. The dynamics of the system written in this reduced form is called the zero dynamics of $\boldsymbol{y}$.

Motion Constraints

The feedback mechanism we call motion constraints is also called virtual constraints in the robotics and control literature. They're widely used to analyze nonlinear and underactuated control systems[1,2] because they effectively reduce a complex dynamic to a simpler one that's easier to study. Motion constraints have been particularly fruitful in the analysis and control of simplified biped walkers.[2–4]

References

1. C. Byrnes and A. Isidori, "Asymptotic Stabilization of Minimum Phase Nonlinear Systems," *IEEE Trans. Automatic Control*, vol. 36, no. 10, 1991, pp. 1122–1137.
2. A.P. Shiriaev and C. Canudas, "Constructive Tool for Orbital Stabilization of Underactuated Nonlinear Systems: Virtual Constraints Approach," *IEEE Trans. Automatic Control*, vol. 50, no. 8, 2005, pp. 1164–1176.
3. C. Canudas, "On the Concept of Virtual Constraints as a Tool for Walking Robot Control and Balancing," *Ann. Rev. in Control*, vol. 28, no. 2, 2004, pp. 157–166.
4. E.R. Westervelt et al., *Feedback Control of Dynamic Bipedal Robot Locomotion*, CRC Press, 2007.

Motion Constraints for 2D Bipeds

When designing motion constraints for 2D bipeds, we assume a specific form of the coordinate transform $\boldsymbol{H}$:

$$\boldsymbol{h}_a = \boldsymbol{q}_a,\ \boldsymbol{h}_u = \theta(\boldsymbol{q})$$

(see Figure 2). The motion constraints take on the form

$$\boldsymbol{y} = \boldsymbol{q}_a - \boldsymbol{c}_w(\theta), \quad (5)$$

and the zero-dynamics set is parameterized by the reduced coordinates $(\theta,\dot{\theta}) \in \mathbb{S}$ alone:

$$\boldsymbol{q} = \boldsymbol{q}_w(\theta)$$
$$\dot{\boldsymbol{q}} = \frac{\partial \boldsymbol{q}_w}{\partial \theta}\dot{\theta}. \quad (6)$$

Eric Westervelt and his colleagues first introduced this choice of $\boldsymbol{H}$ to model the unactuated ankle of a biped with point feet.[2] Our character models have feet, but the torque that we can apply at the ankle without causing the foot to rotate and slip is limited. So, modeling the ankle as an unactuated joint, at least initially, is still prudent.

Designing Motion Constraints for 2D Bipeds

Motion constraints aim to mimic an input motion's joint configurations. We do this by finding the parameters $\boldsymbol{w}$ that best fit the motion. First, we divide the motion into different stages depending on the contact configuration between the character and the ground. For example, we divide a walk cycle into a single-support and a double-support stage.

A different set of motion constraints is active during each stage. In general, between stages, impulsive collisions change the character's velocity state discontinuously. We must ensure that the motion constraints at the end of one stage are consistent with the motion constraint at the beginning of the next stage. We call this the *consistency condition*.

At contact events, we assume a standard inelastic impulse between the character and environment:

$$\mathtt{L}\dot{\boldsymbol{q}}^- = \dot{\boldsymbol{q}}^+, \quad (7)$$

where $\dot{\boldsymbol{q}}^-$ and $\dot{\boldsymbol{q}}^+$ are the joint velocities before and after the impulse and $\boldsymbol{L}$ is a linear map that depends only on the configuration of $\boldsymbol{q}$ at the impulse event. By substituting Equation 6 into Equation 7, we see that the condition for consistency is

$$\boldsymbol{L}\frac{\partial \boldsymbol{q}_w}{\partial \theta}\dot{\theta}^- = \frac{\partial \boldsymbol{q}_{w_1}}{\partial \theta}\dot{\theta}^+ = \frac{\partial \boldsymbol{q}_{w_1}}{\partial \theta}\frac{\partial \theta}{\partial \boldsymbol{q}}\boldsymbol{L}\frac{\partial \boldsymbol{q}_{w_0}}{\partial \theta}\dot{\theta}^-$$
$$\Rightarrow 0 = \boldsymbol{C} = \boldsymbol{L}\frac{\partial \boldsymbol{q}_{w_0}}{\partial \theta} - \frac{\partial \boldsymbol{q}_{w_1}}{\partial \theta}\frac{\partial \theta}{\partial \boldsymbol{q}}\boldsymbol{L}\frac{\partial \boldsymbol{q}_{w_0}}{\partial \theta}, \quad (8)$$

where w_0 and w_1 are the constraint parameters in the adjacent phases.

Finally, we solve for the parameters $\boldsymbol{w}$ that satisfy

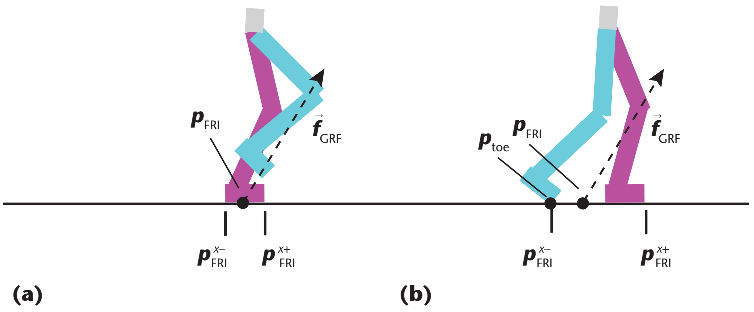

Figure 3. The foot rotation indicator point $\boldsymbol{p}_{\text{FRI}}$ and ground reaction force $\vec{\boldsymbol{f}}_{\text{GRF}}$ during (a) single and (b) double support. The FRI's horizontal component, $\boldsymbol{p}^{x}_{\text{FRI}}$, must remain within the bounds of the support $\left[\boldsymbol{p}^{x+}_{\text{FRI}}, \boldsymbol{p}^{x-}_{\text{FRI}}\right]$ to prevent foot rotation. The swing toe, $\boldsymbol{p}_{\text{toe}}$, is constrained to remain in contact with the ground during double support.

the consistency condition (see Equation 8) and that minimize $\boldsymbol{y}$ along the input motion's trajectory. The optimization takes the form

$$\min_{w} \sum_{i} \left\| \boldsymbol{y}_i \right\|_2 + k \sum_{\theta} \left\| \frac{\partial^2 \boldsymbol{c}_w}{\partial \theta^2} \right\|_2$$
$$\text{subject to } \mathbf{C}_j = 0,$$

where $\boldsymbol{y}_i$ is the value of the motion constraints at each time index i along the walk cycle, $\mathbf{C}_j$ is the consistency condition at stage transition j, and k is a scalar regularization term that helps avoid large accelerations in the motion constraints. The consistency condition is nonlinear in $\boldsymbol{w}$ because $\boldsymbol{L}$ depends on the character's configuration at the transition between stages. We solve the system using nonlinear optimization with finite differencing of the gradients. Generally, we can obtain solutions in minutes on a desktop computer.

Robust Contact and Double Support

The feedback on the motion constraints assumes a fixed connection between the swing foot and the ground. However, in simulation, the character's foot isn't actually attached to the ground. When the input motion is tracked closely, ground reaction forces produced by the character remain in a friction cone, thus acting identically to the fixed foot. However, when large perturbations occur, the feedback from the motion constraints might generate ground reaction forces outside the friction cone. When this occurs, the simulated character's foot will rotate or slip on the ground, often resulting in the character falling. To prevent this, we must temporarily violate the motion constraints in favor of keeping the foot steady.

The Foot Rotation Indicator Policy and Contact Preservation

To prevent the foot from slipping, we compute the resulting *ground reaction force* (GRF) $\vec{\boldsymbol{f}}_{\text{GRF}}$ and ensure that it's within a Coulomb friction cone. To prevent foot rotation, we compute the *foot rotation indicator* (FRI) point $\boldsymbol{p}_{\text{FRI}}$, a point that must remain in the support of the stance foot to prevent rotation[3] (see Figure 3). Formally, $\boldsymbol{p}_{\text{FRI}}$ is defined as the point on the ground plane such that

$$\boldsymbol{u}_r = \boldsymbol{p}_{\text{FRI}} \times \vec{\boldsymbol{f}}_{\text{GRF}}\,, \tag{9}$$

where $\boldsymbol{u}_r$ is the torque on the ankle (assuming a massless foot and the origin at the ankle joint). It is worthwhile noting that the FRI is identical to the similar-purposed zero-moment point (ZMP) for an equivalent mechanism with a fixed foot.[3]

Under the motion constraints' feedback, the choice of $\boldsymbol{u}_r$ fully determines the system's instantaneous acceleration (compare this with Equation 4). Because $\vec{\boldsymbol{f}}_{\text{GRF}}$ is solely a function of the center of mass's acceleration, we can compute this linear relationship:

$$\vec{\boldsymbol{f}}_{\text{GRF}} = \mathrm{A}_{\text{GRF}}\left(\theta, \dot{\theta}\right) \boldsymbol{u}_r + \boldsymbol{b}_{\text{GRF}}\,. \tag{10}$$

In the 2D case, Equation 9 reduces to

$$\boldsymbol{u}_r = \left(\vec{\boldsymbol{f}}^{y}_{\text{GRF}} \boldsymbol{p}^{x}_{\text{FRI}} - \vec{\boldsymbol{f}}^{x}_{\text{GRF}} \boldsymbol{p}^{y}_{\text{FRI}}\right). \tag{11}$$

By substituting Equation 10 into Equation 11 and solving for $\boldsymbol{u}_r$, we can see that to prevent the foot from rotating, we must limit $\boldsymbol{u}_r$ to the set

$$\left\{ \boldsymbol{u}_r = \frac{\boldsymbol{p}^{y}_{\text{FRI}} \boldsymbol{b}^{x}_{\text{GRF}} - \boldsymbol{b}^{y}_{\text{GRF}} \boldsymbol{p}^{x}_{\text{FRI}}}{\mathrm{A}^{y}_{\text{GRF}} \boldsymbol{p}^{x}_{\text{FRI}} - 1.0 - \boldsymbol{p}^{y}_{\text{FRI}} \mathrm{A}^{x}_{\text{GRF}}} \middle| \boldsymbol{p}^{x-}_{\text{FRI}} < \boldsymbol{p}^{x}_{\text{FRI}} < \boldsymbol{p}^{x+}_{\text{FRI}} \right\}, \tag{12}$$

where $\boldsymbol{p}^{x+}_{\text{FRI}}$ and $\boldsymbol{p}^{x-}_{\text{FRI}}$ are the upper and lower bounds of the support foot's contact with the ground. Rather than choosing $\boldsymbol{u}_r$ directly, we define an FRI policy,

$$\Pi\left(\theta, \dot{\theta}\right) : \mathbb{S} \rightarrow \left[\boldsymbol{p}^{x+}_{\text{FRI}}, \boldsymbol{p}^{x-}_{\text{FRI}}\right] \in \mathbb{R}\,,$$

which maps from the reduced state $\left(\theta, \dot{\theta}\right) \in \mathbb{S}$ to a value of $\boldsymbol{p}_{\text{FRI}}$ in the valid range. Given a value of $\boldsymbol{p}_{\text{FRI}}$ from the policy, we compute $\boldsymbol{u}_r$ using Equation 12 and then $\boldsymbol{u}_a$ using Equation 3. This completes the calculation of the joint torques.

In extreme cases, the computed torque still produces a GRF that's outside the allowable friction cone, which would cause the character to slip. To prevent this, we project the resulting GRF back onto the friction cone. However, because the GRF is no longer consistent with the motion constraints, a least-squares minimization is necessary to compute the final joint accelerations:

$$\begin{aligned} &\min_{\ddot{\boldsymbol{q}}} \left\| \ddot{\boldsymbol{y}}(\ddot{\boldsymbol{q}}) \right\|_2 \\ &\text{subject to } \vec{\boldsymbol{f}}_{\mathrm{GRF}} = \vec{\boldsymbol{f}}^{*}_{\mathrm{GRF}} \\ &\qquad \boldsymbol{p}_{\mathrm{FRI}} = \Pi\left(\theta, \dot{\theta}\right), \end{aligned} \tag{13}$$

where $\vec{\boldsymbol{f}}^{*}_{\mathrm{GRF}}$ is the projected version of $\vec{\boldsymbol{f}}_{\mathrm{GRF}}$. We formulate this optimization as a linear-equality constrained least-squares problem, which Lapack's `dgglse` function solves efficiently at runtime. Once `dgglse` determines $\ddot{\boldsymbol{q}}$, an $O(n)$ inverse-dynamics algorithm computes $\boldsymbol{u}$.

Double Support

Previous analyses of gaits with motion constraints have treated double-support stages as instantaneous impulse events. This avoids dealing with closed-loop kinematic configurations but limits motions to ones that don't often occur in nature. The human walking gait, for example, involves a noninstantaneous double-support stage called *toe-off*, in which the back foot pushes off the ground and injects forward momentum. This phase is critical to walking up steep inclines and makes the gait more robust to force perturbations.

In double support, we assume that the back foot's toes contact the ground. To ensure this even when perturbations occur, we modify the motion constraints (see Equation 5) to include the toe's acceleration $\ddot{\boldsymbol{p}}_{\mathrm{toe}}$:

$$\boldsymbol{y} = \begin{bmatrix} \hat{\boldsymbol{q}} \\ \ddot{\boldsymbol{p}}_{\mathrm{toe}} \end{bmatrix} - \begin{bmatrix} \hat{\boldsymbol{c}}_{W}(\theta) \\ 0 \end{bmatrix}. \tag{14}$$

The hat symbol represents an operation that removes an appropriate number of rows from the original vector such that the total number of constraints is still equal to the total number of actuated joints. The removed rows correspond to the back leg's heel and knee. This adapts those joints' motion to the ground contact requirements during the simulation. We apply the same FRI control strategy as in single support, except we expand the bounds on the FRI to include the inscribing polygon of both feet (see Figure 3).

In the final computation of joint torques, an ambiguity still exists owing to redundancy in the actuation due to having both feet on the ground. This ambiguity corresponds to the choice of where to place the aggregate linear force on each foot. Previous control strategies for characters have either ignored this redundancy or resolved it though minimum-joint-torque criteria. However, a better way to resolve the redundancy is to choose an aggregate force near each foot's center. This strategy will be more robust to model discrepancies and force disturbances that might otherwise cause the foot to rotate on edge.

To compute the final joint torques, we use this efficient algorithm:

1. Calculate a valid set of accelerations $\ddot{\boldsymbol{q}}$ under the motion constraints (see Equation 14) and the FRI policy.
2. Use an $O(N)$ recursive algorithm to calculate the joint torques for an open-loop skeleton that's rooted at the front foot.
3. Determine the location of the aggregate linear force on each foot (near the foot's center if possible).
4. Project the aggregate linear force on the back foot onto the joint space through the Jacobian transpose.

We apply the resulting torque directly to the simulated character.

Policy Optimization on the Reduced Dynamics

Robust controllers must incorporate prediction so that they can take the best action to accomplish future goals. Prediction must occur at many time scales. Consider a walking controller. At the instantaneous time scale, it must ensure that the foot doesn't slip. At the time scale of an individual step, it must guide the joint angles along a prescribed path while rejecting unanticipated disturbances. At the time scale of multiple steps, it must ensure that the character reaches its final destination while avoiding obstacles. A character's high dimensionality makes achieving reliable prediction difficult.

A key advantage of motion constraints is that they restrict the character's state to the zero-dynamics set. Owing to this set's low dimensionality, it's computationally feasible to directly sample and tabulate actions' outcomes to provide reliable predictions of future states. These predictions are in the form of transition functions that map a reduced state in $\mathbb{R}$ and an action in $\mathbb{A}$ to a future reduced state:

$$\mathbf{T} = \mathbb{R} \times \mathbb{A} \rightarrow \mathbb{R}. \tag{15}$$

On the basis of these transition functions, we can construct policies that optimally accomplish goals using vanilla reinforcement-learning techniques. (For further information about reinforcement learning, see the related sidebar). In particular, we used *value iteration*[4] to design two

Reinforcement Learning

Reinforcement learning is a way to design control policies that's well established in robotics and other fields.[1] In biped control, researchers have applied it mainly to simple models[2–6] because computations become intractable in higher dimensions. In the main article, we apply a form of reinforcement learning called *value iteration* to more complicated bipeds, using motion constraints to reduce the dimension of the state space. Our research is closely related to using value iteration to actuate a passive walker[7] and to the research of Stelian Coros and his colleagues, who also used reinforcement learning to control a biped.[8]

References

1. D.P. Bertsekas and J.N. Tsitsiklis, *Neuro-dynamic Programming*, Athena Scientific, 1996.
2. H. Benbrahim and J. Franklin, "Biped Dynamic Walking Using Reinforcement Learning," *Robotics and Autonomous Systems*, vol. 22, nos. 3–4, 1997, pp. 283–302.
3. K. Byl and R. Tedrake, "Approximate Optimal Control of the Compass Gait on Rough Terrain," *Proc. 2008 IEEE Int'l Conf. Robotics and Automation* (ICRA 08), IEEE CS Press, 2008, pp. 1258–1263.
4. C. Chew and G.A. Pratt, "Dynamic Bipedal Walking Assisted by Learning," *Robotica*, vol. 20, no. 5, 2002, pp. 477–491.
5. G. Endo et al., "Learning CPG-Based Biped Locomotion with a Policy Gradient Method: Application to a Humanoid Robot," *Int'l J. Robotics Research*, vol. 27, no. 2, 2008, pp. 213–228.
6. J. Morimoto et al., "Poincaré-Map-Based Reinforcement Learning for Biped Walking," *Proc. 2005 IEEE Int'l Conf. Robotics and Automation* (ICRA 05), IEEE CS Press, 2005, pp. 2381–2386.
7. R. Tedrake and H.S. Seung, "Improved Dynamic Stability Using Reinforcement Learning," *Proc. 5th Int'l Conf. Climbing and Walking Robots and the Support Technologies for Mobile Machines* (Clawar 02), Professional Engineering Publishing, 2002, pp. 341–348.
8. S. Coros, P. Beaudoin, and M. van de Panne, "Robust Task-Based Control Policies for Physics-Based Characters," *ACM Trans. Graphics*, vol. 28, no. 5, 2009, article 170.

types of optimal policies in the reduced space: a continuous FRI policy for broad jumping and a discrete policy for walking. We discuss these policies in more detail in the next section.

In practice, the character will act optimally under the learned policies only if the motion constraints are satisfied ($\mathbf{y} = 0$). However, because the feedback causes the constraints to exponentially converge toward zero, the controller can recover from unanticipated disturbances in two stages. First, the motion constraints converge toward zero. Once the character is on the zero-dynamics set, it behaves optimally according to the learned policy.

Results

We experimented with designing phase-indexed tracking controllers and optimizing policies on the reduced dynamics.

Walking

We constructed a robust walking controller that can handle significant force perturbations and unanticipated terrain. We started with a recorded motion and applied spline smoothing to produce a cyclical walk. Next, we identified a monotonic variable θ. As we mentioned before, for walking, a convenient choice is the angle of the vector between the stance ankle and the pelvis. We partitioned the cycle into single- and double-support stages, and we fit the motion constraints' parameters to the motion using the process we described earlier.

The final step in our controller design was to define the FRI policy Π. The simplest policy keeps the FRI constant throughout the entire gait cycle. With this policy, the controller achieves a constant walking speed. Moving the FRI forward or back results in slower or faster walks. If the FRI is too far forward, the controller stops or, in some cases, steps backward, depending on the motion constraints' parameters. For a summary of our walking control algorithm, see Figure 4.

A constant FRI policy won't robustly combat unexpected perturbation such as deviation from flat ground. A simple way to regulate the forward walking speed is to incorporate a PID (proportional-integral-derivative) controller. We found through experimentation that using the integral term alone works best:

$$\Pi\left(\theta,\dot{\theta}\right) = \min\left(fri^{+}, \max\left(fri^{-}, fri_0 + \int_t e\left(t\right)dt\right)\right),$$

where $e\left(t\right) = \dot{\theta} - \dot{\theta}_d$ is an error between the current and desired velocity of θ.

The controller can consistently withstand a forward or backward push of up to 350 Newtons for 0.1 second at all points along the walking cycle. These results are comparable to the ones described for the Simbicon controller[5] after accounting for our character's smaller weight (51 versus 90 kg).

The walking controller is robust to terrain variations and to unanticipated pushes applied to the body. The same controller designed to walk over flat ground also makes forward progress over a sloped ground between –18 and 10 degrees. Other terrain adaptations are possible, including walking over stairs, spongy ground, and a moving-link bridge (see Figure 1). The main failure mode for our controller is when the toe unexpectedly stubs the ground. One possible solution might be to check for ground clearance and take a higher step, but our controller currently doesn't incorporate any higher-level decision-making of this kind.

Comparison to NQR

The linear quadratic regulator and nonlinear variants[6,7] have shown promising results for tracking motions with robustness. They represent the state-of-the-art in time-indexed trajectory tracking. To compare our controller with this indexed-tracking controller, we implemented both our controller and an NQR on a simplified five-link model with point feet.

To produce a trajectory for the NQR to track, we ran our controller for six steps. Designing the NQR involves tuning five different parameters per model DOF. To simplify tuning, we used the same five parameters for all DOFs, as is common practice. We tuned both controllers to be as robust as possible while retaining some compliancy. We clamped torques to 300 Newton meters to prevent use of large forces. To test the robustness, we applied a force to the torso midway through the second step.

After we tuned the NQR gains for maximum stability, the character could withstand forces from –50 to 10 N (in the horizontal direction for 100 milliseconds) without falling during the remaining steps. In contrast, our controller could sustain forces between –500 and 400 N. Tables 1 and 2 summarize the two controllers' parameter sensitivities and shows the viable range of parameters in which they could recover from the push.

Qualitatively, our controller responds considerably differently to the –50 Nm push than the NQR controller does. The NQR controller flails a leg outward to catch up with the motion's original timing. The motion will often diverge significantly from the original trajectory, which can create exciting, dynamic recoveries. However, these recoveries aren't always natural or graceful. The exact response depends heavily on setting the parameters in an unintuitive manner. For example, increasing the penalty for deviating from the joint angles' reference value had the opposite effect in some cases.

```
stage ← DOUBLE
loop
  $\theta, \dot{\theta}$ ← ComputeTheta($\boldsymbol{q}, \dot{\boldsymbol{q}}$)
  if stage = DOUBLE && $\theta > \theta_s$ then
    stage ← SINGLE
  else if stage = SINGLE && footContact() then
    stage ← DOUBLE
  end if
  $\boldsymbol{p}_{\mathrm{FRI}} = \Pi(\theta, \dot{\theta})$
  ($\boldsymbol{u}, \vec{\boldsymbol{f}}_{\mathrm{GRF}}$) = ComputeTorque($\boldsymbol{p}_{\mathrm{FRI}}$) ▷ Equations 3 and 12
  if $\vec{\boldsymbol{f}}^{y}_{\mathrm{GRF}} < 0$ then
    $\vec{\boldsymbol{f}}_{\mathrm{GRF}} = 0$
    $\boldsymbol{u}$ = ComputeLeastSquaresTorque($\boldsymbol{p}_{\mathrm{FRI}}, \vec{\boldsymbol{f}}_{\mathrm{GRF}}$) ▷ Equation 13
  else if $\left\| \vec{\boldsymbol{f}}^{x}_{\mathrm{GRF}} / \vec{\boldsymbol{f}}^{y}_{\mathrm{GRF}} \right\|$ > friction coefficient then
    $\vec{\boldsymbol{f}}^{x}_{\mathrm{GRF}}$ = (friction coefficient) $*$ sign($\vec{\boldsymbol{f}}^{x}_{\mathrm{GRF}}$) $*$ $\vec{\boldsymbol{f}}^{y}_{\mathrm{GRF}}$
    $u$ = ComputeLeastSquaresTorque($\boldsymbol{p}_{\mathrm{FRI}}, \vec{\boldsymbol{f}}_{\mathrm{GRF}}$) ▷ Equation 13
  end if
  ($\boldsymbol{q}, \dot{\boldsymbol{q}}$) = ForwardSimulate($\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}$)
end loop
```

Figure 4. A walking controller algorithm. The walking controller algorithm detects transitions between double and single support and switches between exact tracking of the motion constraints and a least-squares solver that prevents loss of foot stability.

Table 1. Our controller: the parameter range for a stable response to a –50-Newton push.

Parameter	Description	Value	Min.	Max.
ε	Exponential convergence factor	0.05	0.001	0.49
K_s	Position gain	10	0.01	10,000

**We compute the velocity gain as* $K_d = 2\sqrt{K_s}$.

Our controller recovers more predictably and uses smaller torques to do so (see Figure 5).

Broad Jumping with Dynamic Balance

Our broad-jump controller performs a sequence of forward jumps. Because this motion involves dynamic balance, in which the center of mass isn't over the base of support, performing it is tricky. The key control challenge is regulating the landing speed so that the character is prepared for the next jump.

Table 2. The nonlinear quadratic regulator: the parameter range for a stable response to a –50-Newton push.

Parameter	Description	Value	Min.	Max.
Q	Position cost	1,000,000	0	10,000,000
$\dot{Q}$	Velocity cost	100	60	60,000
Q_{end}	Final position cost	100,000,000	10,000,000	1,000,000,000
$\dot{Q}_{end}$	Final velocity cost	100,000,000	100	100,000,000
R	Actuation cost	10	1	10,000

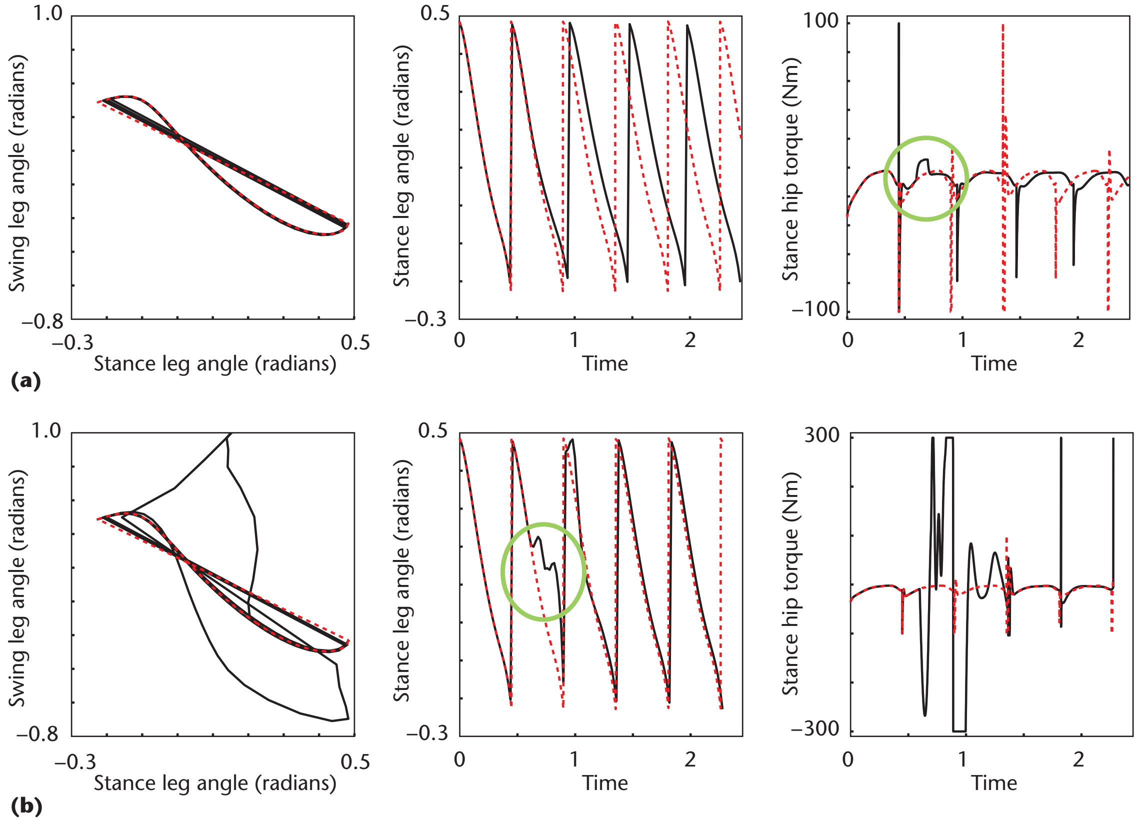

Figure 5. Graphs comparing (a) our controller's response to (b) that of the nonlinear quadratic regulator (NQR) controller, for a push of –50 Newtons for 100 ms. Black lines indicate the response; red lines are the unperturbed reference trajectory. Our controller stays close to the original trajectory (the left graph) by deviating from the original timing (the middle graph). The NQR controller closely tracks the original timing (the middle graph) but uses larger torques (the right graph) to recover. Compared to the NQR controller, ours can recover from pushes that are an order of magnitude larger.

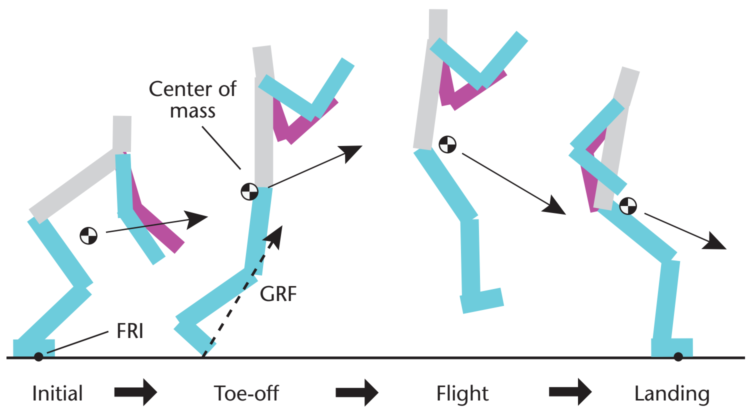

Figure 6. In the initial stage of jumping, the character's feet are flat on the ground. Briefly before flight, the character pushes off with its toes. In the flight stage, there's no contact between the feet and the ground. The character enters the landing stage when the feet are flat on the ground again. Finally, the character transitions back to the initial stage.

For this motion, θ is the horizontal position of the center of mass relative to the feet. The jumping controller involves four stages corresponding to different contact configurations between the character's feet and the ground: *initial*, *toe-off*, *flight*, and *landing* (see Figure 6). Switching between stages occurs at designated values of θ for all transitions except the transition between flight and landing, which occurs when the feet are flat on the ground.

During the flight stage, the initial angular and linear momentums fully determine the character's deterministic trajectory. So, the controller is highly sensitive to the ground reaction force through the toe-off stage. To better control these forces, we replaced the motion constraints on the left knee and ankle with direct control over the ground reaction forces. When designing the controller, we fit a spline function $\vec{f}_{GRF}(\theta)$ to the desired values of these forces, as a function of θ. We treat motion constraints in the initial and landing stages the same way as the double-support stage of walking.

The jump controller is particularly sensitive to the FRI's location throughout the initial and landing stages. Depending on the FRI's value, the jump will either speed up or slow down, which results in the character falling down after several cycles. Simple FRI policies, such as the integral controller

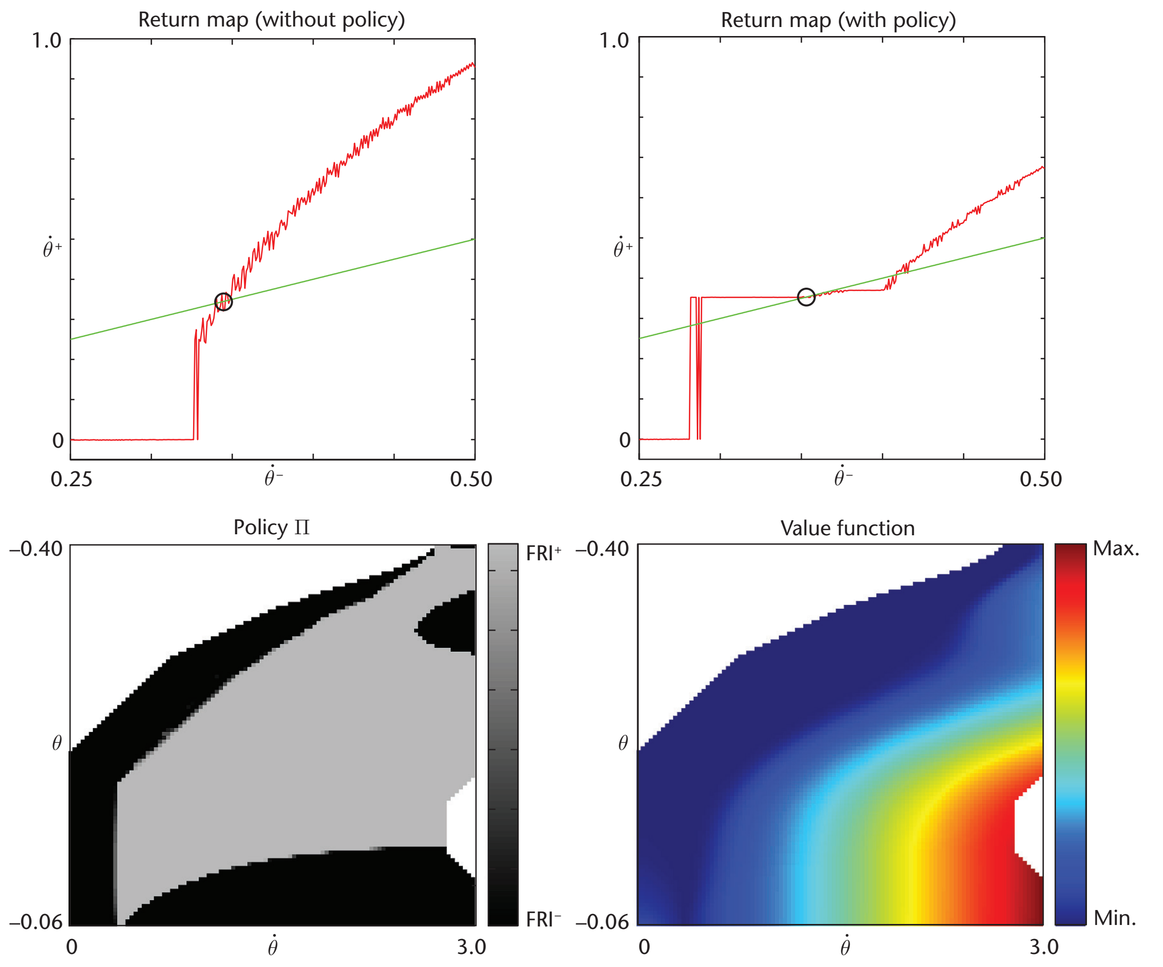

Figure 7. Return maps indicate stability. They map the value of $\dot{\theta}$, which is the horizontal velocity of the center of mass, just before the initial stage of one jump to the value of $\dot{\theta}$ at the same point in the next jump in the sequence. If the return map's slope (the red line) is less than 1 (the green line) at the point of intersection (the circle), successive jumps will converge toward the intersection, and the controller will be stable. Otherwise, the controller will speed up or slow down until failure. We learn an optimal FRI policy Π (bottom left) that results in a stable return map (top right). Although we allow for a range of FRI values, the controller chooses to rapidly switch between extreme values of the FRI. In the policy and value function graphs (bottom left and right), white regions correspond to states from which the controller will fail. The value function predicts, given the current state $\left(\theta,\dot{\theta}\right)$, how close the controller will be to the desired value of $\dot{\theta}$ at the beginning of the next jump.

we described for walking, don't work. Instead, we designed an optimal FRI policy using value iteration.

We learned the FRI policy in the 2D reduced state space $\mathbb{S}$ indexed by the tuple $\left(\theta,\dot{\theta}\right)$. We discretized the state into a 100 × 100 grid, for a total of 10,000 discrete states. The action space $\mathbb{A}$, corresponding to the position of the FRI, was also divided into 20 values in the range $\left[\boldsymbol{p}_{\text{FRI}}^{x-},\boldsymbol{p}_{\text{FRI}}^{x+}\right]$. Next, we sampled a discrete transition function $\mathbf{T}$ (see Equation 15) by initializing the system at each discrete state in $\mathbb{S}$ and simulating with each discrete value of the FRI from $\mathbb{A}$. We defined transitions to an end state whenever the controller switched back to the initial stage. To each end state, we assigned a value $\left\|\dot{\theta}-\dot{\theta}_d\right\|$, where $\dot{\theta}_d$ is the desired horizontal speed at the beginning of the initial stage. Finally, we propagated the value function back to all prior states using value iteration, which also defines an optimal policy Π (see Figure 7).

With a constant FRI policy, the jump controller either speeds up or slows down until failure. You can see this by examining the return map of $\dot{\theta}$ in the top right of Figure 7. The FRI policy reshapes the return map and indicates the jump cycle's stability. By using the learned policy, the jump controller can jump indefinitely on flat ground and even up slight inclines.

Constrained Stepping

We can use policy optimization to sequence phase-indexed controllers, which is the goal of our constrained stepping controller. This controller uses a

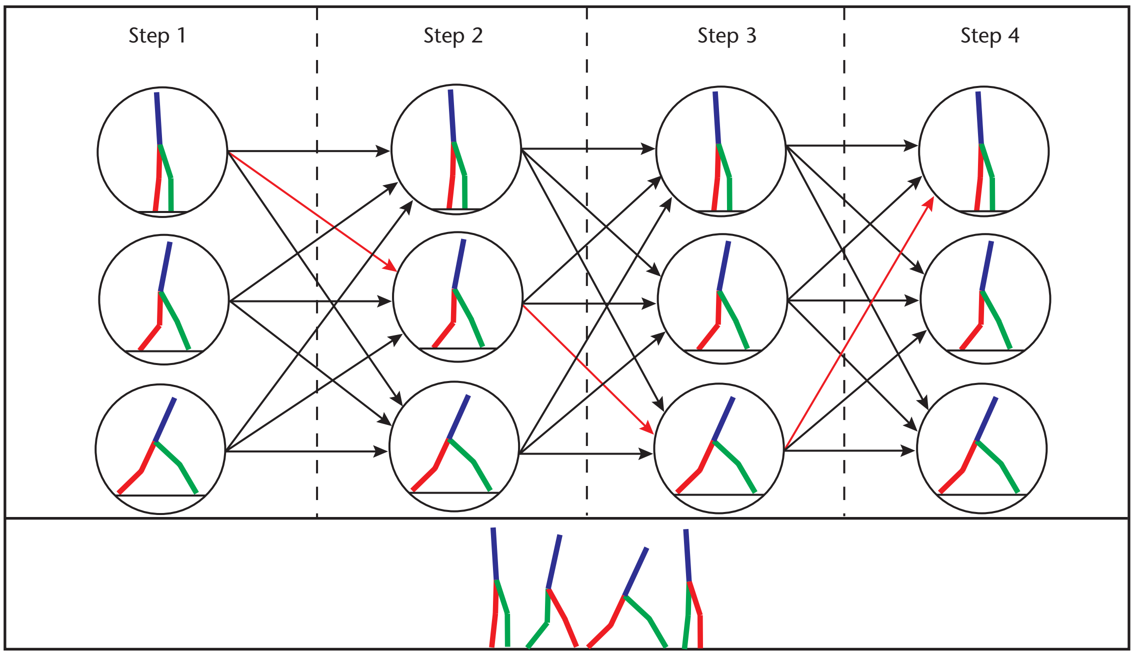

Figure 8. Operation of the constrained stepping controller for a walk of four steps (at the bottom). Arrows represent a fixed choice of motion constraints over the next step's duration. Circles are transition regions between steps. An optimal stepping policy chooses the best sequence of motion constraints (the red arrows) to navigate a constrained terrain. The actual controller has 100 transition regions and 100 possible motion constraints exiting each region.

stepping policy to choose optimal transitions in a connected graph of phase-indexed controllers. The policy is used to guide a character over a terrain to a goal. The stepping policy is discrete in that it's evaluated only at each step's start (as opposed to the FRI policy of the previous section, which is evaluated at every simulation step). An action of the policy corresponds to a choice of a phase-indexed controller for the next step's duration.

We constructed an initial set of 100 phase-indexed stepping controllers by sampling 10 different stride lengths and 10 different speeds at even intervals. We also constructed transition controllers between each pair of stepping controllers. This ensured that the controller maintains the motion constraints' consistency condition between steps of different lengths and speeds. The result was a connected graph of phase-indexed stepping controllers with 100 possible transitions at each node (see Figure 8).

We optimized our sequencing policy to choose an appropriate sequence of steps such that the character reaches the goal while avoiding gaps in the ground. In reinforcement-learning language, this is equivalent to maximizing a value function that rewards steps that reach the goal and penalizes steps resulting in the character stepping in a gap or failing to make forward progress. To optimize the policy, we formulated a value iteration in a 3D state space. The first dimension is the index of the node in the graph (that is, a transition region); the second dimension is the value of $\dot{\theta}$ at the step's start. The remaining dimension is the horizontal position on the terrain.

We partitioned the state space into a 100 × 20 × 500 grid. We sampled the phase-indexed stepping controllers at each discrete state in the grid to tabulate a transition function. Transitions to states in which the character's foot is on a gap map to a failure state; transitions to the goal map to a success state. Discrete states in which the character fails to make forward progress ($\dot{\theta} < 0$) also map to a failure state. Because we assume forward progress, the transition map is acyclic, and the value function is guaranteed to converge in fewer than 500 iterations.

The main cost in performing the value iteration is in tabulating the transition function, which involves performing a number of simulations equal to the number of states times the number of actions (100 × 20 × 500 × 100). Because the steps are identical modulo horizontal translation on the ground plane, the number of necessary simulations decreases by a factor of 500. Once we tabulated the transition function, computing an optimal policy was relatively easy. We can adapt the policy for a different sequence of gaps without resampling the transition function. Additionally, our action set is rich enough that we can include in the value function a weighting for a secondary criterion. We demonstrated this by designing a value function that assigns higher value to taking either longer or shorter steps.

Implementation Details

We solved nonlinear optimization problems using Matlab's `fmincon` function. To efficiently solve the linear-equality constrained, least-squares problems that are necessary for the controller at runtime, we used Lapack's `dgglse` function.

To perform forward simulation of articulated rigid bodies, we used a projected Gauss-Seidel iterative solver.[8] We used a fixed size ($\delta t = 1/240$), velocity-based integration step. We stabilized contact constraints by interleaving a projection step between simulation steps. The projection step updates the configuration by using the generalized pseudoinverse of the contact Jacobian, which minimizes the "inertial norm" ($\delta \mathbf{q}^T \mathbf{M} \delta \mathbf{q}$). Even with a relatively large step size, this procedure closely matched the inelastic impulse model assumed by the consistency condition (see Equation 8). Other procedures for handling contacts, such as Baumgarte stabilization, resulted in less consistent behavior during impulsive collision, which led to rougher motion. All simulations ran in real time or faster.

We hope to develop phase-indexed tracking controllers for 3D characters. Motion constraints are applicable in 3D, and researchers have applied them to control simplified walkers with point feet.[9] However, our initial experiments suggest that directly applying our approach for the 2D case can't generate robust gaits in 3D. Additional feedback on the foot placement and swing-hip angle will likely be necessary. Despite these obstacles, we believe we can apply the basic principle of phase-indexed tracking to 3D characters, with some modification.

References

1. R. Featherstone, *Rigid Body Dynamics Algorithms*, Springer, 2008.
2. E.R. Westervelt et al., *Feedback Control of Dynamic Bipedal Robot Locomotion*, CRC Press, 2007.
3. A. Goswami, "Foot Rotation Indicator (FRI) Point: A New Gait Planning Tool to Evaluate Postural Stability of Biped Robots," *Proc. 1999 IEEE Int'l Conf. Robotics and Automation* (ICRA 99), IEEE CS Press, 1999, pp. 47–52.
4. D.P. Bertsekas and J.N. Tsitsiklis, *Neuro-dynamic Programming*, Athena Scientific, 1996.
5. K. Yin, K. Loken, and M. van de Panne, "Simbicon: Simple Biped Locomotion Control," *ACM Trans. Graphics*, vol. 26, no. 3, 2007, article 105.
6. M. da Silva, Y. Abe, and J. Popović, "Interactive Simulation of Stylized Human Locomotion," *ACM Trans. Graphics*, vol. 27, no. 3, 2008, article 82.
7. U. Muico et al., "Contact-Aware Nonlinear Control of Dynamic Characters," *ACM Trans. Graphics*, vol. 28, no. 3, 2009, pp. 1–9.
8. K. Erleben, "Velocity-Based Shock Propagation for Multi-body Dynamics Animation," *ACM Trans. Graphics*, vol. 26, no. 2, 2007, article 12.
9. C. Chevallereau, J.W. Grizzle, and C.L. Shih, "Asymptotically Stable Walking of a Five-Link Underactuated 3D Bipedal Robot," *IEEE Trans. Robotics*, vol. 25, no. 1, 2008, pp. 37–50.

Yeuhi Abe *is a PhD candidate and member of the Computer Graphics Group at MIT's Computer Science and Artificial Intelligence Laboratory. His research focuses on developing novel methods for simulating and controlling animated characters. Abe has a master's in computer science from MIT. Contact him at yeuhi@csail.mit.edu.*

Jovan Popović *is principal scientist in Adobe Systems' Advanced Technology Labs. His research interests are computer animation and geometric modeling. Popović has a PhD in computer science from Carnegie Mellon University. Contact him at jovan@adobe.com.*