Stable Proportional-Derivative Controllers

Jie Tan* Karen Liu† Greg Turk‡
Georgia Institute of Technology

Abstract

In computer animation, a common technique for tracking the motion of characters is the proportional-derivative (PD) controller. In this paper, we introduce a new formulation of the PD controller that allows arbitrarily high gains, even at large time steps. The key to our approach is to determine the joint forces and torques while taking into account the positions and velocities of the character in the next time step. The method is stable even when combined with a physics simulator that uses simple Euler integration. We demonstrate that these new, stable PD controllers can be used in a variety of ways, including motion tracking in a physics simulator, keyframe interpolation with secondary motion, and constraint satisfaction for simulation.

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

Keywords: Character animation, proportional-derivative controller, constrained dynamics, keyframe interpolation.

1 Introduction

Unlike passive dynamic systems, simulating human motion requires active control that models the functionality of our musculoskeletal system. Applying appropriate control forces on the virtual character is critical to simulate motion that accomplishes desired tasks, as well as to appear realistic and humanlike. However, control forces are also a double-edged sword in a discrete-time numerical simulation. When applied appropriately, a control force can stabilize an otherwise unstable system due to numerical errors. On the other hand, when the control force itself is unstable, the simulation becomes unstable no matter what numerical integration scheme is used.

One common approach to active control is to equip the character with a proportional-derivative (PD) servo at each actuated joint. Acting like a spring and a damper, a PD servo provides a simple framework to compute control forces for tracking a kinematic state of a joint trajectory. However, when precise tracking with high gains is desired, PD servos typically produce unstable control forces which require the numerical simulation to take small time steps. Consequently, the animator has to choose between tracking accuracy and simulation efficiency when using the PD framework.

*e-mail: jtan34@gatech.edu
†e-mail: karenliu@cc.gatech.edu
‡e-mail: turk@cc.gatech.edu

We seek an improved PD control that applies stable control forces while maintaining the intuitive PD framework. Our new formulation, termed "Stable proportional-derivative (SPD)" servo, decouples the relation between high gain and small time step, so that accurate tracking can be achieved without sacrificing simulation efficiency. The key idea is to formulate PD control using the state of the character in the next time step, rather than the current state. Because the control force is computed by the deviation between the desired state and the *as yet unknown* state, we approximate the control force at the next time step by solving an implicit equation. Even though the idea is inspired by the fully implicit integrator [Baraff and Witkin 1998], we emphasize that there are differences and that these differences are important. In Baraff and Witkin's paper, cloth is a passive object without active controls and they adopted a fully implicit integrator to stablize the simulation. In contrast, characters can control themselves with active joint torques, which plays an important role in maintaining the stability of the simulation. Thus, we only use the state at the next time step to calculate a stable control force and use the forward Euler integrator for the integration. Compared with the fully implicit integrator, SPD is more computational efficient.

Although the underlying algorithms are different, SPD is conceptually equivalent to PD but provides much needed stability when high gains are applied. We can replace existing PD controllers with SPD in a variety of applications, without impacting the underlying control or simulation mechanisms. We first show that dynamically tracking motion capture data can be stably simulated with large time steps (33 ms) using first-order explicit Euler integration. We also demonstrate that interpolation of keyframes can create natural secondary motion without tedious parameter tuning. Finally, we show that our controller can replace the penalty method for approximating constraints in a dynamic system.

2 Related Work

PD controllers are widely used in computer animation and robotics due to their simplicity in formulation and their efficiency for online computation. Early work in physics-based character animation used PD servos as the fundamental building blocks for complex motor control strategies [Hodgins et al. 1995]. PD controllers also provide an intuitive framework to integrate physical simulation with motion capture data. Zordan and Hodgins simulate a virtual character tracking a reference mocap sequence while responding to external perturbations [Zordan and Hodgins 1999; Zordan and Hodgins 2002].

The concept of PD servo is closely related to the penalty method for enforcing constraints in a forward simulation. The penalty method [Moore and Wilhelms 1988] uses a spring to maintain constraint by calculating the restoring force based on the deviations from the constraints. Similar to PD servos, the spring in the penalty method introduces numerical instability when high gain is used to precisely enforce the constraint [Witkin et al. 1990]. Nevertheless, due to its simplicity, the penalty method is often more preferable than computing exact constraint forces via Lagrangian multipliers. It has been widely used in handling collision and contact, enforcing the joint limitations and combating numerical drift.

Despite their popularity, PD controllers have many drawbacks. For example, it is difficult to apply the PD servo in a complex environment, such as fluid, because it does not take external forces into consideration. Even in a simpler environment, Van de Panne [1996] pointed out that PD controllers have a hard time tracking the desired pose due to gravity and contact forces. Neff and Fiume [2002] discussed the inter-dependency of position control and stiffness control. Wilhelms [1986] reported that tuning the gain for one joint can adversely affect others. Many of these drawbacks stem from the stability problem of the PD.

Several researchers have proposed other techniques to improve or complement PD controllers. Yin et al. [Yin et al. 2003] applies anticipatory feed forward control together with low-gain feedback. The feedforward torques can be computed offline via inverse dynamics or online via feedback error learning. Combining feedforward control, it is possible to track the reference motion precisely using low-gain PD servos. Neff and Fiume [Neff and Fiume 2002] formulate an antagonistic controller that decouples the stiffness control from position control by equipping each DOF with a pair of ideal springs in opposition to each other. Weinstein et al. [2008] also improved the stability of PD controller by using the analytical solution on each joint individually. Different from conventional PD controllers, their method took an impulse-based approach rather than working with forces and accelerations and their method relies on the global post-stablization to control multiple joints simultaneously.

3 Stable PD Controllers

Before we introduce the formulation of SPD, we first review how conventional PD controllers are used in numerical simulation and their limitations. In a discretized time domain, a PD controller can be expressed as:

$$\tau^n = -k_p(q^n - \bar{q}^n) - k_d\dot{q}^n \tag{1}$$

where q and $\dot{q}$ are the position and velocity of the state at time step n, and where $\bar{q}$ is the desired position. k_p and k_d are the proportional gain and derivative gain respectively. The resulting control force τ^n is then included in a numerical simulation, along with other forces such as gravity. The stability issue arises when the controller needs to quickly reduce the deviation from the desired position. In this situation, the proportional gain k_p must set to a large value, and the control force can become numerically unstable as the simulation progresses. To improve the stability of a high gain PD controller, we have to sacrifice the efficiency of the simulation by reducing the time step significantly. As a result, PD controllers suffer from undesired coupling between the tracking accuracy and simulation efficiency.

3.1 SPD Formulation

Instead of using the current state q^n to compute the control force, we propose a new formulation, SPD, that computes the control forces using the next time step q^{n+1}:

$$\tau^n = -k_p(q^{n+1} - \bar{q}^{n+1}) - k_d\dot{q}^{n+1} \tag{2}$$

Since we do not know the next state, we expand q^{n+1} and $\dot{q}^{n+1}$ using a Taylor series and truncate all but the first two terms.

$$q^{n+1} = q^n + \Delta t\dot{q}^n$$

$$\dot{q}^{n+1} = \dot{q}^n + \Delta t\ddot{q}^n$$

Equation (2) can be expressed as:

$$\tau^n = -k_p(q^n + \Delta t\dot{q}^n - \bar{q}^{n+1}) - k_d(\dot{q}^n + \Delta t\ddot{q}^n) \tag{3}$$

In practice, PD servos with non-zero reference velocity (4) is often used to reduce the lag of tracking.

$$\tau^n = -k_p(q^n - \bar{q}^n) - k_d(\dot{q}^n - \dot{\bar{q}}^n) \tag{4}$$

where the reference velocity $\dot{\bar{q}}^n$ is calculated either analytically or numerically. SPD formulation can also be extended to include the non-zero reference velocity term to eliminate the lag:

$$\tau^n = -k_p(q^n + \Delta t\dot{q}^n - \bar{q}^{n+1}) - k_d(\dot{q}^n + \Delta t\ddot{q}^n - \dot{\bar{q}}^{n+1}) \tag{5}$$

3.2 A Toy Example

Consider a simple dynamic system with only one degree of freedom, q. The goal of the controller in the system is to maintain the position of q at $\bar{q}$. Suppose the initial position and initial velocity are $q^0 = p$ and $\dot{q}^0 = \dot{p}$. The dynamic equation of this 1D system is

$$\begin{aligned} m\ddot{q}^0 &= \tau^0 && (6)\\ &= -k_p(p + \Delta t\dot{p} - \bar{q}) - k_d(\dot{p} + \Delta t\ddot{q}^0) && (7)\end{aligned}$$

Rearranging the equation, we arrive at

$$\ddot{q}^0 = -\frac{k_p(p + \Delta t\dot{p} - \bar{q}) + k_d\dot{p}}{m + k_d\Delta t} \tag{8}$$

To illustrate the stability of SPD, we make k_p approach infinity and $k_d = k_p\Delta t$. In practice, $k_d \geq k_p\Delta t$ should be maintained to ensure the stability. Please see Appendix for further discussion. Plugging k_p and k_d in this extreme example, the acceleration becomes $\ddot{q}^0 = -\frac{p+2\Delta t\dot{p}-\bar{q}}{\Delta t^2}$. After one step of forward Euler integration, the velocity becomes $\dot{q}^1 = -\dot{p} - \frac{p-\bar{q}}{\Delta t}$ and the position becomes $q^1 = p + \Delta t\dot{p}$. After another step of forward Euler integration, the position becomes $q^2 = \bar{q}$. Through this toy example, we find that even though we have chosen very large PD gains (infinity here) and an arbitrary time step, the stability of the controller is still guaranteed and the motion converges to the desired trajectory quickly.

3.3 Controlling Articulated Rigid Bodies

The formulation of SPD can be applied to nonlinear dynamic systems with multiple degrees of freedom. For example, we can apply SPD to an articulated rigid body system expressed in generalized coordinates.

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) = \tau_{\mathbf{int}} + \tau_{\mathbf{ext}} \tag{9}$$

where $\mathbf{q}$, $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ are vectors of positions, velocities and accelerations of the degrees of freedom respectively. $\mathbf{M}(\mathbf{q})$ is the mass matrix and $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$ is the centrifugal force. $\tau_{\mathbf{ext}}$ indicates other external force such as gravity. The internal force $\tau_{\mathbf{int}}$ can be computed via our SPD formulation:

$$\tau_{\mathbf{int}} = -\mathbf{K}_p(\mathbf{q}^n + \dot{\mathbf{q}}^n\Delta t - \bar{\mathbf{q}}^{n+1}) - \mathbf{K}_d(\dot{\mathbf{q}}^n + \ddot{\mathbf{q}}^n\Delta t) \tag{10}$$

where both $\mathbf{K}_p$ and $\mathbf{K}_d$ are diagonal matrices that indicate the gains and damping coefficients. The acceleration can be written as:

$$\ddot{\mathbf{q}}^n = (\mathbf{M} + \mathbf{K}_d\Delta t)^{-1}(-\mathbf{C} - \mathbf{K}_p(\mathbf{q}^n + \dot{\mathbf{q}}^n\Delta t - \bar{\mathbf{q}}^{n+1}) - \mathbf{K}_d\dot{\mathbf{q}}^n + \tau_{\mathbf{ext}}) \tag{11}$$

We use the explicit Euler method to integrate to the next time step.

We compare our SPD formulation with the fully implicit integrator when simulating articulated rigid bodies. Since $\mathbf{M}$ is nonlinearly dependent on $\mathbf{q}$ and $\mathbf{C}$ is nonlinearly dependent on $\mathbf{q}$ and $\dot{\mathbf{q}}$, in an implicit integrator, both $\mathbf{M}$ and $\mathbf{C}$ need to be linearized. However, the linearization of $\mathbf{M}$ and $\mathbf{C}$ is non-trivial because this involves

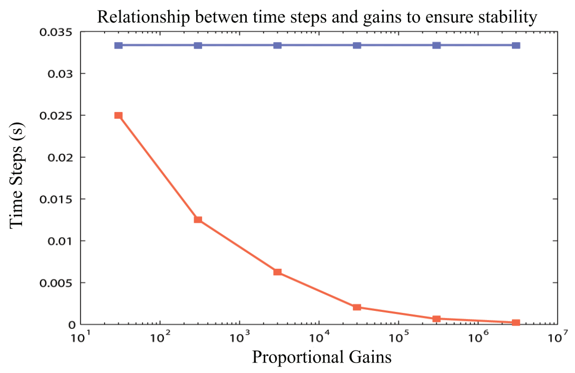

Figure 4: *For each chosen gain k_p, we simulate PD (red) and SPD (blue) with the largest time step that maintains stability.*

computing another order of derivatives and careful treatments of the singularities of the rotation representations. In SPD, no linearization is needed, which not only greatly simplifies the derivation and implementation, but also saves in computational time. Another difference between implicit integrator and SPD lies in that they have different stability characteristics. An implicit integrator is unconditionally stable while SPD is stable under the condition that $\mathbf{K}_d \geq \mathbf{K}_p \Delta t$ (Refer to the Appendix for a detailed stability analysis). This condition makes SPD less stable, but in return, SPD is more computationally efficient. Moreover, implicit integrator usually overly damps the motion while SPD does not suffer from this issue. For character animation, overly damped motion can produce undesirable artifacts for timing-critical motions, such as catching a basketball (Figure 1).

3.4 Experiments

To compare our method with a conventional PD controller, consider a two-link pendulum with a hinge joint at each link. The reference trajectory for the two hinge joints are $q_1(t) = cos(t)$ and $q_2(t) = cos(t) + 1.0$ respectively. For each chosen gain, we simulate PD and SPD with the largest time step that maintains stability. Figure 4 shows the relation between the gain and the largest time step allowed. We sample k_p at 3×10^i, $i = 1, 2, \cdots, 6$ and set $k_d = k_p \Delta t$. Note that the largest time step for SPD is bound by the reference motion that is sampled at 30 frames per second, not by the stability condition.

We compare the tracking errors between PD and SPD with different gains at $\Delta t = \frac{1}{60} s$. For clarity, we still choose the simple example of a two-link pendulum with two hinge joints. Figure 2 and 3 show the tracking error at the two DOFs respectively. When small or medium gain are chosen, i.e. $k_p = 30$ or 300, both PD and SPD controllers are fairly stable, but SPD controller exhibits smaller oscillatory motion around the reference trajectory. When we apply larger gains, i.e. $k_p = 3000$ or 30000, SPD controller tracks the reference motion closely while PD controller fails to maintain stability and eventually crashes the simulation. Table 1 and 2 summarize the tracking errors $||e||_\infty$ and $||e||_2$. The measurement for PD controllers is not available at $k_p = 3 \times 10^3$ and 3×10^4, because the simulation quickly diverges due to the unstable control force. We also conduct experiments on tracking the velocity of the reference trajectory (instead of tracking zero velocity). The results are shown in Figure 2, Figure 3, Table 1, Table 2, as well as the accompanying video. We observe small but noticeable reduction of tracking delay in both PD and SPD controllers. In particular, when SPD tracks the reference velocity, the tracking error decreases approximately linearly with respect to the gain increase. This error reduction is not observed when SPD tracks zero velocity. We hypothesize this error is due to the tracking delay between the simulated motion and reference motion.

4 Applications

Because SPD can replace PD without changing underlying control or simulation mechanisms, our method can potentially improve all the existing applications that depend on PD controllers, such as tracking a motion trajectory, maintaining a kinematic state, or enforcing penalty constraints. Although the main contribution of this paper is not building a specific application, we introduce three possible ways to leverage SPD as a demonstration of the wide applicability of this method.

4.1 Tracking and Simulation

In computer animation, it is often useful to make a virtual character perform a set of desired behaviors while responding to dynamic changes and stimuli in the environment. One simple and effective technique to do this is to use PD controllers to track a predefined input sequence. Since the character's motion results from a physical simulation, it can deviate from the input motion under external forces. In this technique, the PD gains play an important role in the visual fidelity of the resulting motion, especially when external forces disturb the character. Large gains result in stiff reactions while small gains cannot follow the desired behavior well. Zordan and Hodgins [Zordan and Hodgins 2002] proposed a schedule to modulate the gains according to the perturbation and achieved pliable reactions. The main problem they encounter with this approach is that the simulation is unstable when tracking with high gains. They circumvented this by using small time steps (0.67ms). With SPD controllers, we do not have such problems. We can use arbitrarily high gains while still using large time steps 33ms. With 50 times larger time steps, SPD is so efficient that the gain tuning becomes an interactive process.

To demonstrate that SPD produces responsive motion using much larger time steps, we first use the same schedule as [Zordan and Hodgins 2002] to modulate the gains for the examples shown in Figure 1. Different from Zordan and Hodgins [2002], which applied a controller to actively keep balance, we anchor the root of the character using SPD servo with large gains. We demonstrate three examples: soccer kicking, volleyball hitting and basketball catching. Each input sequence was motion captured when a subject pretended to kick, hit or catch without a ball. The motion is smooth because no momentum transfer from the ball occurred in the mocap stage. After the simulation with SPD controllers, the virtual character reacts to the perturbation caused by the ball. In the first example, when the character kicks a very heavy ball, the speed of the swinging leg slows down suddenly the moment it collides with the ball. Then the leg gradually accelerates as if the character is putting more effort towards making the ball move. In the volleyball example, when the volleyball hits the character's forearms with high speed, its arms suddenly accelerate downwards due to the momentum transfer. We imposed joint limits at the elbows in this case because otherwise, the forearms would bend backwards when hit by the volleyball. We enforce the joint limits by using SPD controllers with large proportional gains $k_p = 10^9$. When the character catches a fast moving basketball, its elbows bend and the hands move toward the chest to slow the ball down. When the ball comes closer to the chest, the character leans backward to avoid to be hit. The parameters we used for the examples are included in Table3, where k_t is the gain for tracking, k_r is the gain for reaction, t_e is the reaction duration and t_f is the recover duration. Please re-

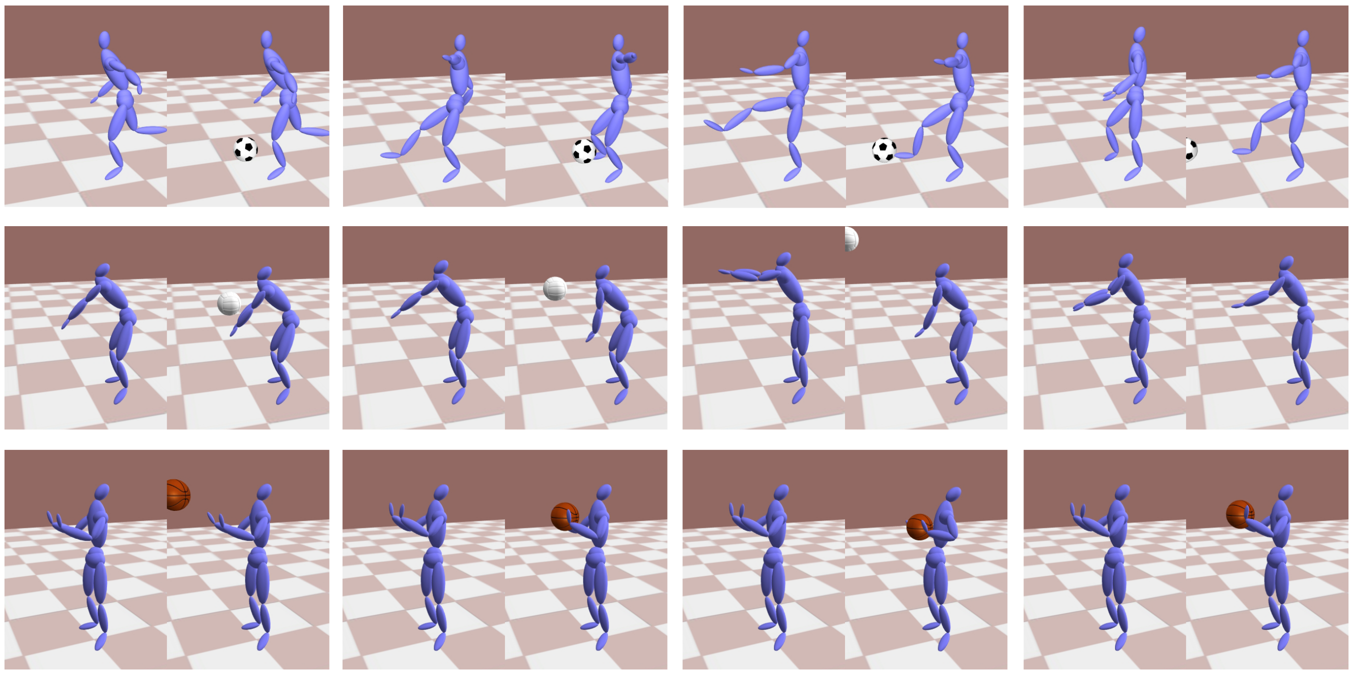

Figure 1: *Tracking and simulation examples. Top row: four separate frames simulating kicking a heavy soccer ball. Middle row: four frames simulating hitting a fast volleyball. Bottom row: four frames simulating catching a fast basketball. The left portion of each frame shows the input sequence while the right portion shows the simulated motion with SPD controllers.*

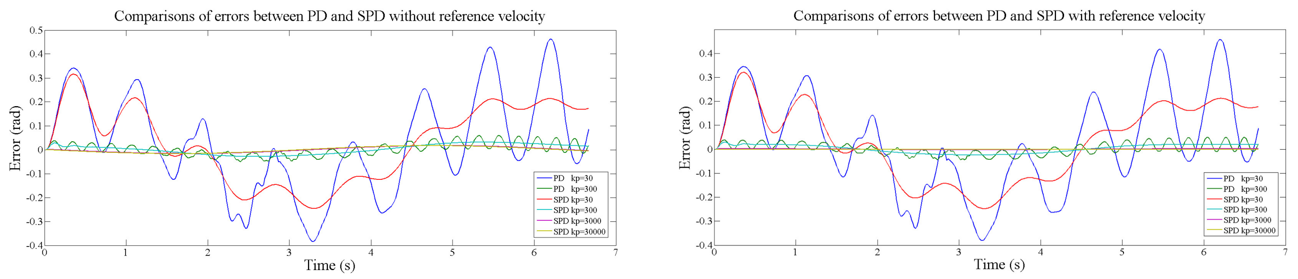

Figure 2: *Comparisons of tracking errors at first degree of freedom between PD and SPD with various gains. Left: PD/SPD formulations does not include a reference velocity. Right: PD/SPD formulations include a reference velocity.*

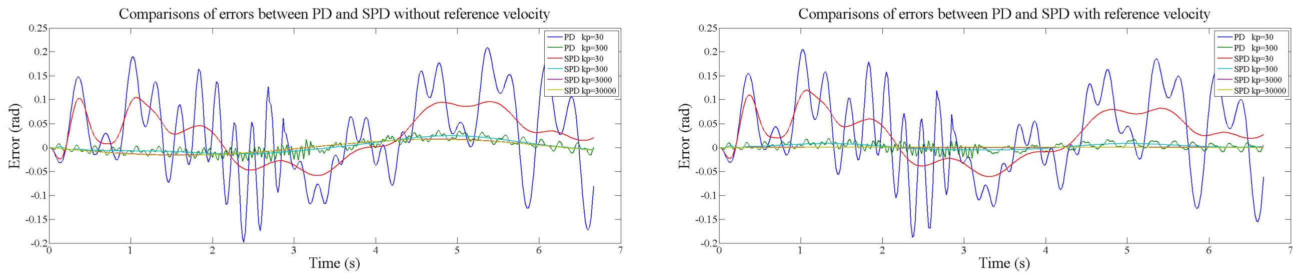

Figure 3: *Comparisons of tracking errors at second degree of freedom between PD and SPD with various gains. Left: PD/SPD formulations does not include a reference velocity. Right: PD/SPD formulations include a reference velocity.*

fer to [Zordan and Hodgins 2002] for a more detailed explanation of each parameter.

Starting with a sequence of motion data, a user can tune the SPD gains to simulate different reactions under different circumstances. We want to emphasize that the parameter tuning is a simple task because 1) the simulation is so efficient with large time steps that we can tune the parameters and preview the result interactively. 2) We can use and tune a global gain for all the degrees of freedom without stability problems. In the previous PD formulation, gains are usually chosen separately for each degree of freedom. One important reason is that a single gain might be too loose for the root while too stiff for the hand. The result is that the root does not track

k_p	PD		PD with $\bar{\dot{q}}$		SPD		SPD with $\bar{\dot{q}}$	
	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$
3×10^1	0.4630	4.0542	0.4585	4.0317	0.3164	3.1939	0.3210	3.1955
3×10^2	0.0598	0.5188	0.0505	0.4658	0.0312	0.4047	0.0297	0.3451
3×10^3	NA	NA	NA	NA	0.0174	0.2301	0.0023	0.0330
3×10^4	NA	NA	NA	NA	0.0167	0.2293	0.0001	0.0016

Table 1: *Comparisons of error for the first degree of freedom when tracking a reference trajectory using different PD formulations.*

k_p	PD		PD with $\bar{\dot{q}}$		SPD		SPD with $\bar{\dot{q}}$	
	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$
3×10^1	0.2094	1.8177	0.2048	1.7778	0.1049	1.0817	0.1198	1.0652
3×10^2	0.0364	0.2826	0.0225	0.1601	0.0244	0.2466	0.0082	0.0964
3×10^3	NA	NA	NA	NA	0.0174	0.2294	0.0008	0.0087
3×10^4	NA	NA	NA	NA	0.0167	0.2295	0.0003	0.0017

Table 2: *Comparisons of error for the second degree of freedom when tracking a reference trajectory using different PD formulations.*

Animation	k_t	k_r	$k_d/(k_p\Delta t)$	t_e	t_f
soccer(light)	25000	5	8	0.1	0.5
soccer(heavy)	25000	5	96	0.1	0.5
volleyball(slow)	25000	5	8	0.1	0.5
volleyball(fast)	25000	5	16	0.1	0.5
basketball(slow)	25000	5	16	0.6	50
basketball(fast)	25000	1	16	0.6	50

Table 3: *Parameters used in the tracking and simulation examples.*

well but the hand starts to move unstably. Tuning a 42-degree-of-freedom character with 42 gains is tedious and frustrating. With our formulation, we only need to tune one global gain.

4.2 Keyframe Interpolation

Keyframe animation is one of the most fundamental techniques to create character animation. Many artists appreciate the full controllablility it provides, but very few would advocate for its ease of use in practice. In particular, keyframes for complex dynamic motion can be tedious to create, and the results seldom look realistic.

We propose a new keyframe interpolation technique to simplify the process of creating dynamic motion. Our method works particularly well when passive, secondary motion is evident in the scene. We simulate the articulated system using two SPD controllers with time-varying gains to track the adjacent keyframes. The time-varying gain profile determines the timing characteristics of the motion and can be designed by the user. We illustrate two possible profile of the gains in Figure 5. Linearly blending two gains (Left) results in stiff motions. The second profile uses higher order polynomials. The sharp contrast between the gain at the keyframe and the gain at inbetween frames ensures that the keyframes are well satisfied while the motion remains compliant. The new interpolation method has three advantages: 1) the interpolation is physics-based; 2) the interpolation is local, which only requires the two adjacent keyframes and 3) the timing of the interpolation can be controlled by the user.

We verify our new interpolation method using a “toy mouse” example. Figure 6 compares our results with an interpolated sequence using cubic splines. Because we only specify the root position and orientation of the mouse at each keyframe, the tail appears rigid in the interpolated motion. In contrast, our method generates realistic secondary motion on the tail as the mouse accelerates and makes turns. We choose $k_t = 130m_{sup}$, $k_d = 10k_p\Delta t$ for each degree of freedom, where m_{sup} is the total mass supported by the joint. We use the gain profile $k_{pi}(t) = (1.0-\alpha)^{24}k_t$ and $k_{pi+1}(t) = (\alpha^{24}+0.005)k_t$ to cross fade the gains, where α is the normalized distance to the ith keyframe. Our method is suited for motions with sudden changes in acceleration and less effective when the motion is inherently smooth.

We also compare SPD with PD controllers. The stability issue with PD prevents us from using large gains (simulation explodes quickly with $k_p = 10m_{sup}$). Smaller gains ($k_p = 0.005m_{sup}$) produce more stable motion, but the interpolated motion does not meet the keyframes at the specified time. The stability of PD controllers is also sensitive to the physical properties of the model. Because the tail is represented by a long chain of light links, the mass matrix is often ill-conditioned, which exacerbates the stability problem of the PD controller.

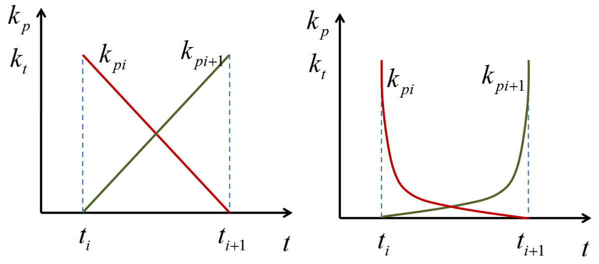

Figure 5: *Two examples of the profile of PD gains.*

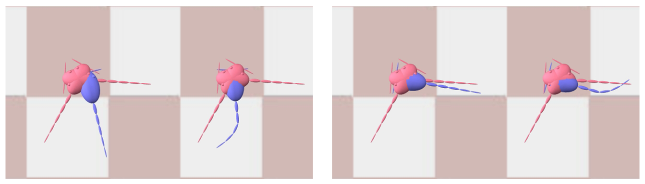

Figure 6: *Two interpolated frames from the “toy mouse” example. The red mice are the keyframes while the blue one calculated from the interpolation. The left portion of each image is generated using the cubic spline interpolation while the right portion is generated using our method.*

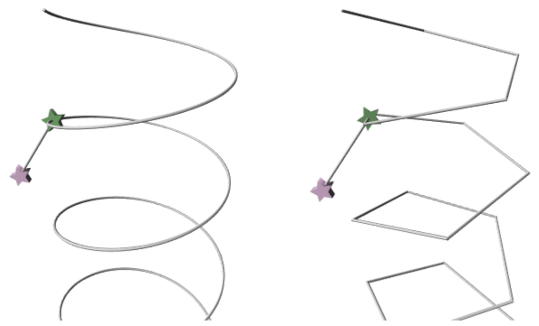

Figure 7: *3D tinkertoy examples. Left: one particle connected with the other one sliding down along a smooth helix curve. Right: one particle connected with the other one sliding down along a piecewise linear helix curve.*

4.3 Extension to Constrained Simulation

One simple yet widely used implementation of constrained dynamics is to use the penalty method. Since the penalty method is analogues to using a PD controller to track the constraint, these methods share the same problem. To ensure that the constraints are always satisfied, the spring constant must be large enough to overpower all competing forces, and this means that very small deviations will induce large restoring forces. However, when a large spring constant is chosen, the penalty method is numerically unstable unless sufficiently small time steps are used. To overcome this numerical difficulty, Witkin [1997] used Lagrangian multipliers to directly compute the constraint forces. Their method favors a differentiable parameterization of the constraint, which makes it difficult to apply to more general cases.

Our method offers an elegant way to stabilize the penalty method, so that both stiff parameters and large time steps can be achieved simultaneously. Instead of computing the penalty force using the current state, we predict the deviation from the constraint in the next time step and plan the constraint force accordingly. In other words, we use a stable PD controller with large gains to closely track the constraints.

We demonstrate our method with a simple 3D Tinkertoy example. We implemented two different types of constraints: particle-on-curve and distance constraints. For the former constraint, we represent the curve as a large number of piecewise line segments. In the simulation, we first predict the position of the particle $\mathbf{q}^{n+1}$ at the next step using Equation (3). We then find out the nearest position on the curve $\bar{\mathbf{q}}^{n+1}$. As long as the curve is smooth, the direction $\bar{\mathbf{q}}^{n+1} - \mathbf{q}^{n+1}$ is orthogonal to the curve and agrees with the direction $\mathbf{d}$ of the control force. We calculate the constraint force using our stable PD controller.

$$\tau = -k_p(\mathbf{q}^{n+1} - \bar{\mathbf{q}}^{n+1}) - k_d \mathbf{d}\mathbf{d}^T \dot{\mathbf{q}}^{n+1} \tag{12}$$

Note that the second term on the right hand side of Equation(12) is different from that of Equation (2) because only the velocity along the direction $\mathbf{d}$ violates the constraint and needs damped out. After plugging Equation (12) into the Newton's second law, we get the modified equation of motion for the on-curve particle:

$$(\mathbf{M} + k_d \Delta t \mathbf{d}\mathbf{d}^T)\ddot{\mathbf{q}} = -k_p(\mathbf{q}^n + \dot{\mathbf{q}}^n \Delta t - \bar{\mathbf{q}}^{n+1}) - k_d \mathbf{d}\mathbf{d}^T \dot{\mathbf{q}}^n \tag{13}$$

where $\mathbf{M}$ is a diagonal mass matrix for the particle.

For the distance constraint, two particles should always stay r_0 apart as if a massless rod connects them. Similar to the particle-on-curve constraint, we first predict the distance between the two particles $|\mathbf{q}_1^{n+1} - \mathbf{q}_2^{n+1}|$ at the next time step. The constraint force is exerted along the line between the two particles $\mathbf{d} = (\mathbf{q}_1^{n+1} - \mathbf{q}_2^{n+1})/|\mathbf{q}_1^{n+1} - \mathbf{q}_2^{n+1}|$. Similarly, the constraint force should only damp out the velocity along the direction $\mathbf{d}$. Using the stable PD controller formulation, we get

$$\tau_1 = -k_p(\mathbf{q}_1^{n+1} - \mathbf{q}_2^{n+1} - r_0\mathbf{d}) - k_d \mathbf{d}\mathbf{d}^T(\dot{\mathbf{q}}_1^{n+1} - \dot{\mathbf{q}}_2^{n+1}) \tag{14}$$

and

$$\tau_2 = -\tau_1 \tag{15}$$

Plugging these into the Newton's second law and rearranging the terms, we get the modified equation of motion for the distance-constrained pair of particles:

$$\begin{aligned} & (\mathbf{M} + k_d \Delta t \mathbf{d}\mathbf{d}^T \mathbf{D})\ddot{\mathbf{q}}^n \\ = \; & -k_p(\mathbf{D}(\mathbf{q}^n + \Delta t \dot{\mathbf{q}}^n) - r_0\mathbf{d}) - k_d \mathbf{d}\mathbf{d}^T \mathbf{D}\dot{\mathbf{q}}^n \end{aligned} \tag{16}$$

where

$$\mathbf{D} = \begin{bmatrix} \mathbf{I} & -\mathbf{I} \\ -\mathbf{I} & \mathbf{I} \end{bmatrix}$$

In the first experiment, we set up a smooth helix using 2400 short line segments. A green particle is constrained to be on a helix (using a particle-on-curve constraint) and a purple particle is connected to the green one by a massless rigid rod (using a distance constraint). The only forces throughout the simulation are the gravity and the constraint forces. When the simulation starts, the green particle slides down the helix while the purple one acts as a swinging pendulum.

In the second experiment, we test the same scene except that we use only 24 instead of 2400 line segments to approximate the helix. As a result, the curve is composed of many non-differentiable sharp corners. This setting imposes extreme difficulties to the simulation because 1) any methods that requires analytical derivatives is not applicable and 2) the numerical derivative near the discontinuities introduces large errors, which can easily drive the simulation out of control. We set $k_p = 10^{16}$ and $k_d = 1.5 k_p \Delta t$ for both constraints. Even though we notice some oscillations when the green particle passes the sharp turns in the second setting, the oscillations damp out quickly and never accumulate. The simulation is stable in both experiments.

With a simple modification of the penalty method, we can use high gains and large time steps simultaneously without raising any numerical stability issue. We believe this idea can be further extended to handle other types of constraints, such as collision, contact and more.

5 Limitations

Although we have successfully used SPD in serveral settings, it does have some limitations. First, despite the fact that SPD overcomes stability issues, it is still a PD-like control that cannot be used for sophisticated control strategies. For example, it does not explicitly solve the problem of balance or long-horizon planning. Consequently, it is difficult to create motion using SPD that is considerably different from the reference motion.

Second, SPD can require extra computational resources in some settings. Note that the computation of control forces is intertwined with the forward simulation step. For applications that explicitly process both control and forward simulation, SPD does not require additional computation because the effect of the control force can be computed at the same time as simulation. Some applications handle control and simulation separately, however, such as using a black-box simulator or controlling a robot (no simulation needed). For such applications, additional computational resources must be used for SPD to compute control forces that are independent of the simulation.

Third, since we predict next state using the first order Taylor expansion, we have no mathematical proof that SPD works with any integration schemes. We have implemented and tested that SPD is compatible with Explicit Euler, Midpoint, RK4 and implicit integration schemes.

Finally, although we demonstrate that SPD can replace penalty methods to enforce most constraints, our current implementation is ill-suited for unilateral constraints such as foot-ground contact because it can generate pulling forces towards the ground. Adapting SPD to handle such constraints is a fruitful area for future work. Using SPD for unilateral constraints could potentially improve the performance of a simulation drastically, as the time step does not need to be reduced for handling collision.

6 Conclusion

In this paper, we presented a new formulation of a PD servo that decouples the dependency of high gain control and small time steps. Our approach uses the state of the character in the next time step to calculate stable control forces. Since the PD controller serves as a fundamental building block for many sophisticated control algorithms, the improvement of PD can potentially have a wide impact on practical applications. We have demonstrated the applicability of our method in various examples, including tracking and simulation, keyframe interpolation and constrained dynamics.

For future research, we would like to explore automatic methods for tuning the gains for the SPD controllers. It is difficult and inefficient to tune PD controllers using search algorithms or optimization techniques because the stability of the motion is highly sensitive to the combinations of gains, resulting in a very “jagged” search space. On the other hand, SPD is stable with both high gains and large time steps. Using SPD, such a search procedure can be conducted more efficiently and will be more likely to converge to an optimal solution.

A Appendix: Stability Analysis

We choose $k_d \geq k_p \Delta t$ for the sake of stability. In the one dimensional case, $k_d = k_p \Delta t$ gives the fastest convergence to the desired trajectory. We have shown in Section 3.2 that q converges to $\bar{q}$ in two time steps [1]. We will analyze the stability of SPD in two cases: $k_d \geq k_p \Delta t$ and $k_d \leq k_p \Delta t$. Without loss of generality, we can write $k_d = \alpha k_p \Delta t$ where $\alpha \geq 0$. From Equation (8), it is easy to verify that the acceleration towards the desired trajectory is monotonically increasing with respect to k_p and decreasing with respect to α when the other parameters are fixed. We already know that q does not overshoot the desired trajectory when $\alpha = 1$ and $k_p = \infty$. Thus q cannot overshoot the target trajectory with less stringent case $\alpha \geq 1$ (more damping) or $k_p \leq \infty$ (less stiffness). We conclude that the controller is stable with arbitrarily gains in the case of $k_d \geq k_p \Delta t$. When $\alpha < 1$, however, stability cannot be guaranteed.

In a multi-dimensional dynamic system, the interdependency of DOFs becomes very complex and the above one-dimensional analysis does not directly apply. However, we can argue that the lower bound, $k_d \geq k_p \Delta t$, still plays a crucial role in stability for multi-dimensional systems. Suppose we have a highly stiff system with a very large $\mathbf{K}_p$, $\mathbf{K}_d$ will also be very large due to the lower bound, $\mathbf{K_d} \geq \mathbf{K_p} \Delta t$. Because SPD adds $\mathbf{K_d}$ to the mass matrix (refer to Equation(11)), $\mathbf{M} + \mathbf{K}_d \Delta t$ becomes a near-diagonal matrix when $\mathbf{K}_d$ dominates $\mathbf{M}$. In this case, the results of the stability analysis for one-dimensional case hold well. When $\mathbf{K}_p$ is small, rigorous analysis is difficult to achieve but small gains usually do not induce instability in practice. From our empirical data, we have never experienced any stability problem under the condition: $\mathbf{K_d} \geq \mathbf{K_p} \Delta t$.

[1] Two steps are the minimum time for q to converge using explicit Euler because the integration from the acceleration $\ddot{q}$ to the displacement q needs two time steps.

References

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 43–54.

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.

MOORE, M., AND WILHELMS, J. 1988. Collision detection and response for computer animation. In *Computer Graphics*, 289–298.

NEFF, M., AND FIUME, E. 2002. Modeling tension and relaxation for computer animation. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM, New York, NY, USA, 81–88.

VAN DE PANNE, M. 1996. Parameterized gait synthesis. *IEEE Computer Graphics and Applications 16*, 40–49.

WEINSTEIN, R., GUENDELMAN, E., AND FEDKIW, R. 2008. Impulse-based control of joints and muscles. *IEEE Transactions on Visualization and Computer Graphics 14*, 1, 37–46.

WILHELMS, J. 1986. Virya—a motion control editor for kinematic and dynamic animation. In *Proceedings on Graphics Interface '86/Vision Interface '86*, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 141–146.

WITKIN, A., GLEICHER, M., AND WELCH, W. 1990. Interactive dynamics. In *COMPUTER GRAPHICS*, 11–21.

WITKIN, A. 1997. Physically based modeling: Principles and practice – constrained dynamics. In *COMPUTER GRAPHICS*, 11–21.

YIN, K., CLINE, M. B., AND PAI, D. K. 2003. Motion perturbation based on simple neuromotor control models. In *Pacific Graphics*.

ZORDAN, V. B., AND HODGINS, J. K. 1999. Tracking and modifying upper-body human motion data with dynamic simulation. In *In Computer Animation and Simulation 99*, 13–22.

ZORDAN, V. B., AND HODGINS, J. K. 2002. Motion capture-driven simulations that hit and react. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM, New York, NY, USA, 89–96.