Inverse Kinodynamics: Editing and Constraining Kinematic Approximations of Dynamic Motion

Cyrus Rahgoshay[1] Amir Rabbani[1] Karan Singh[2] Paul G. Kry[1]

[1] School of Computer Science, Centre for Intelligent Machines, McGill University
[2] Department of Computer Science, University of Toronto

Abstract

We present inverse kinodynamics (IKD), an animator friendly kinematic workflow that both encapsulates short-lived dynamics and allows precise space-time constraints. Kinodynamics (KD), defines the system state at any given time as the result of a kinematic state in the recent past, physically simulated over a short temporal window to the present. KD is a well suited kinematic approximation to animated characters and other dynamic systems with dominant kinematic motion and short-lived dynamics. Given a dynamic system, we first choose an appropriate kinodynamic window size based on accelerations in the kinematic trajectory and the physical properties of the system. We then present an inverse kinodynamics (IKD) algorithm, where a kinodynamic system can precisely attain a set of animator constraints at specified times. Our approach solves the IKD problem iteratively, and is able to handle full pose or end effector constraints at both position and velocity level, as well as multiple constraints in close temporal proximity. Our approach can also be used to solve position and velocity constraints on passive systems attached to kinematically driven bodies. We show IKD to be a compelling approach to the direct kinematic control of character, with secondary dynamics via examples of skeletal dynamics and facial animation.

Index Terms: I.3.7 [Three-Dimensional Graphics and Realism]: Animation—

1 Introduction

Physical simulation is now a robust and common approach to recreating reality in virtual worlds and is almost universally used in the animation of natural phenomena, ballistic objects, and character accessories like clothing and hair. Despite these strides, the animation of primary characters continues to be dominated by the kinematic techniques of motion capture and above all traditional keyframing. Two aspects of a primary character in particular, skeletal and facial motion, are often laboriously animated using kinematics.

We note from conversations with about half a dozen master animators that there are perhaps three chief reasons for this. First, kinematics, unencumbered by physics, provides the finest level of control necessary for animators to breathe life and personality into their characters. Second, this control is direct and history-free, in that the authored state of the character, set at any point in time, is precisely observed upon playback and its impact on the animation is localized to a neighborhood around that time. Third, animator interaction with the time-line is WYSIWYG (what-you-see-is-what-you-get), allowing them to scrub to various points in time and instantly observe the character state without having to playback the entire animation.

The same animators expressed the utility and importance of secondary dynamics overlaid on primarily kinematic character motion to enhance the visceral feel of their characters. Various approaches to such secondary dynamics have been proposed in research literature [7, 11, 13], some of which are available in commercial animation software. Overlaid dynamics, unfortunately compromise the second and third reasons animators rely on pure kinematic control.

A kinematic solution incorporating secondary dynamics called *kinodynamic* skinning [4] was suggested in the context of volume preserving skin deformations. With this approach, a kinodynamic state at any time is defined as a kinematic state in the recent past, physically simulated forward to the given time. In this paper we develop this idea of kinodynamics (KD) as a history-free kinematic technique that can incorporate short-lived dynamic behavior. Note that the above usage of the term "kinodynamic", while similar in spirit, is distinct from its use in the context of robot motion planning where it addresses planning problems where velocity and acceleration bounds must be satisfied [9].

We begin by formulating an appropriate KD window size for a given kinematic motion and physical parameters: both long enough to ensure a temporally coherent KD trajectory that captures the nuances of system dynamics, and short enough for interactive WYSIWYG computation and temporal localization of the influence of animation edits on system state. Many goal directed actions such as grasping, reaching, stepping, gesticulating, and even speaking, however, involve spatial relationships between the character and its environment, that are best specified directly, as targets states that the character (or parts of the character) must observe at given times. Techniques such as inverse kinematics (IK) and space time optimization algorithmically infer the remaining system states and animation parameters from these animator specified spatio-temporal targets. However, IK does not give the secondary dynamics, and space time optimization is typically computationally expensive. Analogous to these techniques we develop an inverse kinodynamics (IKD) algorithm allowing animators to prescribe position and velocity constraints at specific points in time within a KD setting.

The contribution of this paper is thus the development of a usable kinodynamic framework for interactive character animation, where animators can leverage a direct history-free kinematic workflow, coupled with the benefits of arbitrary physically simulated secondary dynamics. Toward this, we present the first IKD algorithm.

2 Related work

Secondary dynamics provides a significant amount of visual realism in kinematically driven animations and is an important technique for animators. In the case of tissue deformations produced by the motion of an underlying skeleton, various methods can be used to produce this motion through simulation or using precomputation [7, 11, 13]. With respect to secondary dynamics of skeletal motion, it has similarly been demonstrated that tension and relaxation of the skeletal animation can be altered through physically based simulation [17]. These techniques provide an important richness to an animation; while the style of the results are controllable by adjusting elastic parameters or gains of controllers used for tracking, precise control of the motion itself to satisfy given constraints or key frames is typically left to be treated as a separate problem.

The related work can be categorized into two groups. First, there are approaches which try to control a physically based simulation to have it meet some desired constraints. Second, there are approaches which use kinematic editing techniques to produce animations that meet desired constraints and exhibit physically plausibility.

Controlling physically based simulations is a difficult problem. There has been a significant amount of work in this area on controlling rigid bodies [21, 20], fluids [24, 15], and cloth [26, 6]. Other recent successes on controlling physically based animation use gentle forces to guide an animation along a desired trajectory, accurately achieving desired states, but also allowing physical responses to perturbations [5]. Physically based articulated character control has received a vast amount of interest. Building on the seminal work of locomotion control [22], it is now possible to have, for instance, animation of physically based motions that respond naturally to perturbations [28, 27, 29], and editable animations of dynamic manipulation which respects the dynamic interaction between characters and objects [1]. Our work is very different from these approaches, and is instead more closely related to work by Allen et al. [2], which changes PD control parameters to produce skeletal animations that interpolate key-frames at specific times. In our work, however, we keep the control parameters fixed and alter the kinematic trajectory. Jain and Liu [10] show a method for interactively editing interaction between physically based objects and a human. In this work, it is the motion of the dynamic environment which is edited through kinematic changes of a captured human motion. In comparison, we focus on altering and editing a kinodynamic motion with different styles (tension and relaxation) and different constraints. Directly related to the problem of authoring motion, physically correct motion can be achieved by solving optimizations with space-time constraints [25]. Also relevant is work that uses analytic PD control trajectories for compliant interpolation [3].

In contrast to the work on controlling fully dynamic simulations, we are addressing a simplified problem due to the finite temporal window involved in simulating the state at a given time in a kinodynamic trajectory [4]. This leads to benefits in the context of animation authoring, and allows for a straightforward solution to the inverse kinodynamic problem that we present in this paper. In a different approach, with similar objectives to our own work, Kass and Anderson [12] propose a method for including physically based secondary dynamics in a key frame style editing environment through interactive solutions of space time optimization problems. They focus on linear or linearized space-time constraints problems, while our work, in contrast, looks primarily at non-linear skeletal animation problems. Within a purely kinematic setting, Coleman et al. [8] create handles to edit motion extrema of different joints clustered in time. The visual impact of secondary dynamics is often captured in these temporal relationships. Such an approach can, however, only exaggerate or diminish a dynamic effect already present in the motion and cannot introduce new forces and dynamic behaviors that the mixing of kinematics and dynamics allows. Such mixing to get the best of both worlds also has promise for authoring motion in real time. For instance, Nguyen et al. [18] blend kinematic animation and dynamic animation via a set of forces which act like puppet strings to pull the character back to the kinematic trajectory. Also of note is work on editing kinematic motion through momentum and force [23], or with biomechanically inspired constraints [14]. While these different approaches use dynamic principles to control accelerations and velocity, they deal with dynamic systems which are not necessarily short lived, and these approaches do not share our objective of a scrubbing interface for animation editing which computes states largely in a history free manner.

3 Overview

In this section, we provide an overview of our approach. The animation is principally driven by a kinematic trajectory $\boldsymbol{x}_K(t)$, typically authored and edited using traditional keyframe and motion capture techniques. The kinodynamic trajectory of the system $\boldsymbol{x}_{KD}(t)$ at a time t is the result of a physical simulation run over a time window δ starting from an initial position $\boldsymbol{x}_K(t-\delta)$ and velocity $\dot{\boldsymbol{x}}_K(t-\delta)$. The simulation uses a PD (Proportional-Derivative) controller to follow the kinematic trajectory, so the $\boldsymbol{x}_K(t)$ can be thought of as the target or desired trajectory. The PD controller applies forces to the system that are proportional to the difference between the set point $\boldsymbol{x}\boldsymbol{x}_K$ and the process variable $\boldsymbol{x}$. We also apply viscous damping, thus the forces can be written as $K_p(\boldsymbol{x}_K - \boldsymbol{x}) - K_d\,\dot{\boldsymbol{x}}$, where the gain K_p can be seen as modeling tension and relaxation, while K_d controls damping.

We will have kinodynamic states which deviate from the kinematic trajectory because we are using a simulation with control forces to generate the KD trajectory. This is desirable because we want to include the effects of secondary dynamics in the animation (see Figure 1). However, there may be specific times in the animation where we need constraints to be met.

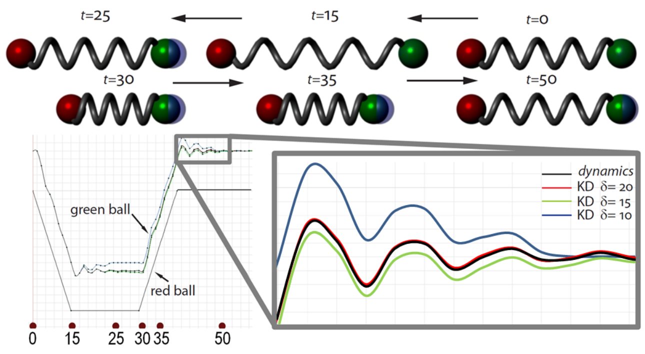

Figure 1: KD trajectories for the green ball: Kinematically the green ball is rigidly connected to the keyframed red ball, with spring dynamics overlaid. A number of frames of the KD trajectory ($\delta = 15$) are shown, with the full dynamics solution for the green ball overlaid in blue (top). KD trajectories with 3 window sizes are shown in relation to a full dynamics solution (bottom). Note how the history-free KD trajectories capture the visual behavior of the actual dynamics over a wide range of window sizes (see video).

Suppose target pose $\boldsymbol{x}_i$ must be produced at time t_i. This target state could be a pose in the original kinematic trajectory, or something different. If the pose belongs to the original kinematic trajectory, a simple solution would be to stiffen the PD control in the vicinity of the desired pose so that it is tracked precisely. Note, however, that stiffness is an inherent attribute of the motion's secondary dynamics under animator control and altering it to interpolate a target pose imbues the animation with a different style. Instead, we iteratively compute a modification to the kinematic trajectory which results in a KD state that satisfies the constraint.

Example trajectories are likewise illustrated in Figure 2, where a red kinodynamic trajectory follows a green kinematic trajectory (suppose it is lower due to gravity). At left we can see an illustration of how the temporal window for computing kinodynamic state must be long enough for any impulse (smaller than a given maximum) to come sufficiently close to rest that it can not be perceived (e.g., based on screen pixels). At right in the figure we can see a dotted green kinematic trajectory with an added bell shape correction, which produces the dotted red kinodynamic trajectory satisfying the constraint at time t_i. This smooth modification of the kinematic trajectory is the approach we use to solve the IKD problem (we use a Gaussian, as described in Section 6).

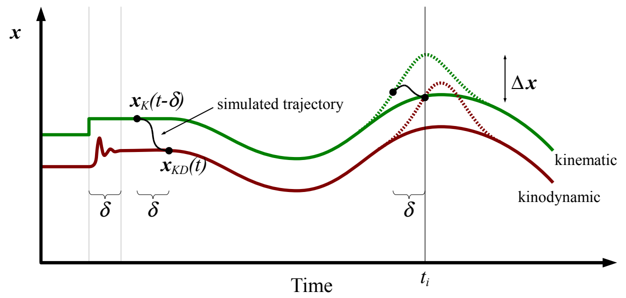

Figure 2: An illustration of how we modify a kinematic trajectory to create a kinodynamic trajectory that satisfies the constraint that the original kinematic state be produced at time t_i.

3.1 Inverse kinodynamics

Let SimulateKD($\boldsymbol{x}_K$, t_i, δ) be the procedure for computing $\boldsymbol{x}_{KDi}$, the KD state at t_i for kinematic trajectory $\boldsymbol{x}_K$. We first compute the IKD error in meeting the target as

$$\boldsymbol{e}_i = \boldsymbol{x}_i - \boldsymbol{x}_{KDi}, \tag{1}$$

and from this we form bell shaped correction curves $\boldsymbol{e}_i\phi_i(t)$ that we add to kinematic trajectory (note that $\boldsymbol{e}_i$ is a vector of same dimension as the state, and each coordinate of the state will have a bell shaped correction of a different magnitude). The bell shaped basis function $\phi_i(t)$ provides a local correction, has its peak value of 1 at t_i, and can be defined as a low degree polynomial or Gaussian. More importantly, it has a local support (i.e., a small temporal width, σ) which is selected by the artist. Conceptually, this IKD error correction introduces an additional spring force proportional to $\boldsymbol{e}_i\phi_i(t)$ in a small temporal neighborhood around t_i. This correction will not be sufficient, however, and our modified KD state $\tilde{\boldsymbol{x}}_{KDi}$ = SimulateKD($\boldsymbol{x}_K + \boldsymbol{e}_i\phi_i, t_i, \delta$) will not meet the constraint. This is because the correction did not take into account the dynamics of the system, but we can fix this by boosting the correction to account for the dynamics, assuming that the system dynamics are approximately locally linear (see Figure 3). Letting $\boldsymbol{d}_i = \tilde{\boldsymbol{x}}_{KDi} - \boldsymbol{x}_{KDi}$, we project the error onto this initial correction result to compute the scaled correction

$$\boldsymbol{f}(t) = \Delta\boldsymbol{x}_i\phi_i(t), \tag{2}$$

where $\Delta\boldsymbol{x}_i = \left(\boldsymbol{e}_i \cdot \boldsymbol{d}_i / ||\boldsymbol{d}_i||^2\right)\boldsymbol{e}_i$. Without this linear prediction step, the convergence is significantly slower.

Using $\boldsymbol{x}_K + \boldsymbol{f}$, the process is repeated, until the system state converges to within a numerical threshold of $\boldsymbol{x}_i$ at t_i. That is, we find the new kinodynamic state at t_i, compute the error $\boldsymbol{e}_i$, the modified kinodynamic state using $\boldsymbol{x}_K + \boldsymbol{f} + \boldsymbol{e}_i\phi_i$, the correction result $\boldsymbol{d}_i$, and finally an update to the correction function

$$\Delta\boldsymbol{x}_i \leftarrow \Delta\boldsymbol{x}_i + \left(\boldsymbol{e}_i \cdot \boldsymbol{d}_i / ||\boldsymbol{d}_i||^2\right)\boldsymbol{e}_i. \tag{3}$$

4 KD animation and IKD scenarios

In this paper, we look at a number of scenarios that can largely be described as either pose constraints (as described above) or end effector constraints (Section 4.1). For instance, we may want a kinodynamic skeletal animation of a dance to produce some key poses, or a kinodynamic skeletal animation of a punch that actually hits

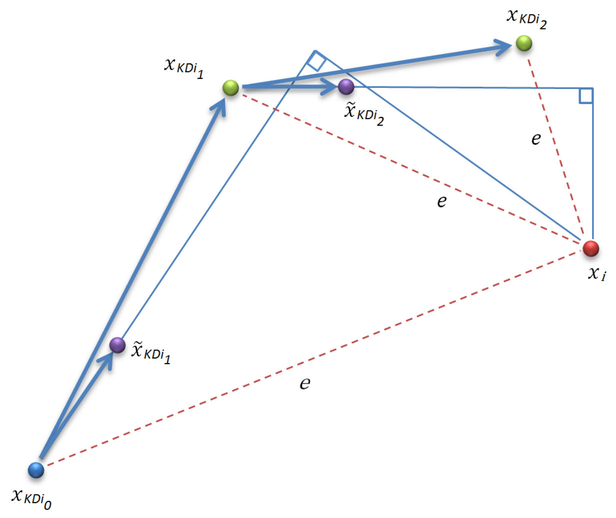

Figure 3: Two steps of the IKD iteration for a constraint at time t_i. At bottom left, $\boldsymbol{x}_{KDi_0}$ is the initial KD state at time t_i, which is far from target $\boldsymbol{x}_i$. A correction based on $\boldsymbol{e}$ results in the modified KD state $\tilde{\boldsymbol{x}}_{KDi_1}$, which does not take into account the system dynamics. We project the error onto $\boldsymbol{d}_i = \tilde{\boldsymbol{x}}_{KDi_1} - \boldsymbol{x}_{KDi_0}$ to determine a scaling of the correction that would produce a KD state as close as possible to $\boldsymbol{x}_i$ assuming linear system dynamics. Using the scaled correction (Equation 3), we produce the new KD state $\boldsymbol{x}_{KDi_1}$, and repeat the process until $||\boldsymbol{e}||$ falls below a threshold.

the desired target at a specific time. Alternatively, another scenario which is important to consider is the case where we drive a deformable mesh animation to follow a target mesh animation. In contrast to joint angles, this case involves a state vector formed by the Cartesian position of vertices in the mesh. In Section 4.5, we show how this approach can be used for facial animation.

We note that the blending of the correction can be done in a number of ways. If we only have one position constraint to satisfy in the entire animation, then it would be possible to naively apply a constant offset to the kinematic trajectory in order to meet the constraint at time t_i. Typically we will have several constraints at different times, so we only make a local edit to the desired trajectory (see Section 4.2). Any of a number of smoothly shaped curves with compact support will serve this purpose, as discussed in Section 6. The shape and width of the correction basis functions are an important artist control, much like setting ease-in ease-out properties in a key frame animation.

4.1 Skeletal animation end effector IKD

In the case of an articulated character, the state $\boldsymbol{x}$ is a set of joint angles, and the simulation uses a PD controller to follow the kinematic trajectory. The gains of the controller set the level of tension or relaxation of the character [17].

When editing a skeletal motion, we may wish to set constraints on the entire pose, as described above, but it is also important that we are able to constrain only part of the state at the time of a contact event, for instance, a point on a hand or foot (we will call such points *end effectors*). Suppose the end effector position of an articulated character is given by $p(\boldsymbol{x})$, and that it must reach position p_i at time t_i. In this case, we have the constraint $p(\boldsymbol{x}_{KDi}) = p_i$, and we use an inverse kinematics solution to map the end effector error to an error in the state.

Figure 4 shows an example of how we solve the IKD problem of punching a target. While the motion in Figure 4(a) hits the target at the desired time, we change the motion style by adjusting the tracking gains of the physically simulated character shown in orange, to

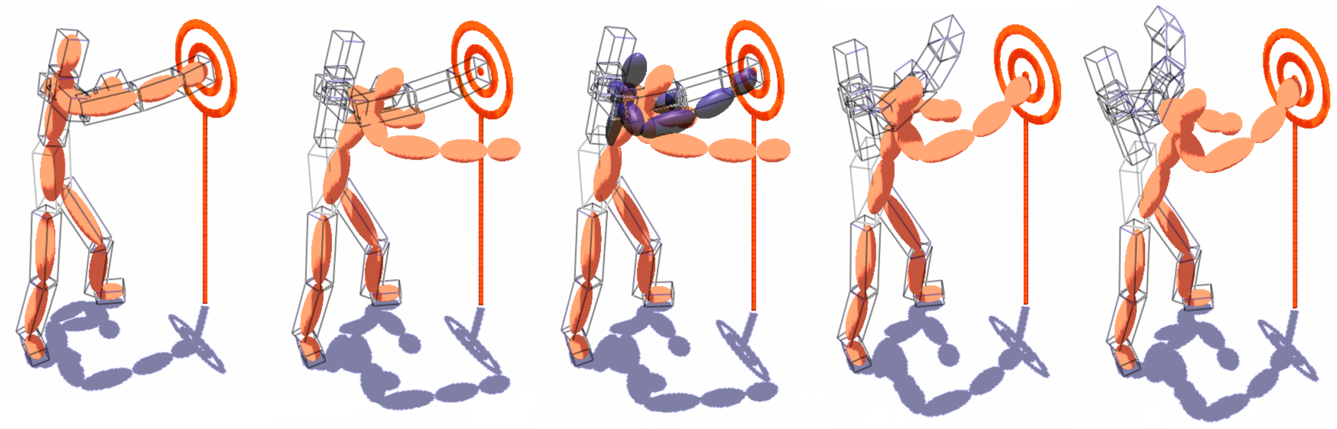

Figure 4: Illustration of how IKD is used to produce an animation of a relaxed character that punches a target. (a) shows the motion capture at the time of contact in both wire-frame and solid orange. (b) the solid orange character shows the KD state of the relaxed character, which fails to reach the target at the time of contact. (c) inverse kinematics produces the pose of the character in dark blue. (d) iteratively computing the error and modifying the kinematic trajectory produces a KD state which hits the target (orange). Here the modified motion capture pose is shown in wire-frame. (e) shows the result of using a smaller temporal width for the bell shaped correction curve, which results in more of an upper cut.

produce the more relaxed KD motion show in Figure 4(b). This relaxed motion fails to hit the target, but we can solve an inverse kinematics problem to adjust the joints of our relaxed character so that the end effector does hit the target. This IK solution pose is shown in dark blue in Figure 4(c). We could make a purely kinematic fix to our KD trajectory by simply layering this IK solution on top our KD trajectory, using a bell shaped curve to slowly ease the correction in and out. However, this does not respect the relaxed dynamics of the character (see the accompanying video). Instead, we modify the kinematic trajectory used to produce the kinodynamic animation. By editing the kinematic trajectory, we produce a natural looking motion that exhibits a relaxed style with a follow through motion. This modification is shown in Figure 4(d) and (e) for two bell shaped correction curves of different widths.

The algorithm iterates as described in the previous section, using an update to the correction curve that is based on an inverse kinematics solution,

$$\boldsymbol{e}_i = \text{SolveIK}(\boldsymbol{x}_{KDi}, p_i) \tag{4}$$

where SolveIK computes a state displacement $\boldsymbol{e}_i$ such that $p(\boldsymbol{x}_{KDi} + \boldsymbol{e}_i) = p_i$. Note that the correction function update must be modified to use the end effector error, $\Delta p_i = p_i - p(\boldsymbol{x}_{KDi})$, instead of the state displacement $\boldsymbol{e}_i$. The update becomes

$$\Delta \boldsymbol{x}_i \leftarrow \Delta \boldsymbol{x}_i + \left(\Delta p_i \cdot d_i / ||d_i||^2 \right) \boldsymbol{e}_i. \tag{5}$$

where $d_i = p(\tilde{\boldsymbol{x}}_{KDi}) - p(\boldsymbol{x}_{KDi})$.

4.2 Multiple constraints

The process of designing an animation typically involves setting multiple constraints at different times throughout the animation. If these events are sufficiently far apart, we can treat each as an independent IKD problem. However, constraints in close temporal proximity may need to be solved simultaneously. This can happen in a variety of ways. If the bell shaped correction curve necessary to satisfy one constraint modifies kinematic states that fall within the temporal window used to simulate the KD state at a another constraint, then the solution of the latter constraint will depend on the solution of the former. This dependence can be one way, or both ways, depending on the temporal width of the bell shaped curves used for each constraint, and the temporal window size used for the kinodynamic simulation. While we may be able to solve some constraints independently, or in a specific order, for simplicity we will make the assumption in our examples that all constraints are temporally coupled and must be solved simultaneously. In our multi-constraint examples, we typically choose correction function widths that provide an ease-in trajectory with a duration of approximately one or two seconds, so constraints that fall within one or two seconds of one another will need to be addressed simultaneously.

The correction $\boldsymbol{f}$ that we must add to the kinematic state to satisfy a number of constraints can now be seen as an interpolation function. That is, $\boldsymbol{f}$ interpolates a set of corrections $\Delta \boldsymbol{x}_i$ at t_i, for $i = 1..N$ where N is the number of constraints. We implement this interpolated correction function using a sum of basis functions,

$$\boldsymbol{f}(t) = \sum_i^N \lambda_i \phi_i(t), \tag{6}$$

where the basis function coefficients λ_i are computed by solving a linear system of equations,

$$\boldsymbol{f}(t_i) = \Delta \boldsymbol{x}_i, \quad \text{for } i = 1..N. \tag{7}$$

Note that the coefficients λ_i are vectors with the same dimension as $\boldsymbol{f}$, i.e., the dimension of the state.

At each iteration, the multi-constraint IKD solver must produce an appropriate update to each $\Delta \boldsymbol{x}_i$. In Section 3.1, we effectively computed numerical partial derivatives with respect to the bell shaped basis magnitudes, and found a least squares solution for the desired update using a projection (computed with a dot product). In the case of two constraints i and j we have d_i influenced by a magnitude adjustment for constraint j, but we avoid the expense of computing these relationships by fixing only one constraint at a time. Thus we have an inner loop that consists of computing the KD state at t_i, the error $\boldsymbol{e}_i$, the updated KD state for trajectory $\boldsymbol{x}_K + f + \boldsymbol{e}_i \phi_i$, the update for $\Delta \boldsymbol{x}_i$ (using Equation 3 or Equation 5), and then finally we recompute the interpolation function weights. This approach, similar to Gauss Seidel iteration, works well because the effect of ϕ_i on $\boldsymbol{x}_{KDi}$ is typically much larger than at $\boldsymbol{x}_{KDj}$.

The technique we use to solve the multi-constraint IKD problem is summarized in Algorithm 1, and consists of a nested loop of adjusting the correction $\boldsymbol{f}$ to fix each of the violated constraints, until all constraints are sufficiently satisfied or a maximum number of iterations is reached.

Algorithm 1 Inverse Kinodynamics Multi-Constraint Solve

Input: constraints $\boldsymbol{x}_i$ or p_i at t_i, for $i = 1..N$, δ
Output: state correction curve f

```
1: $itr \leftarrow 0$
2: $E \leftarrow \infty$
3: $\Delta\boldsymbol{x}_i \leftarrow 0$, for $i = 1..N$
4: $\boldsymbol{f} \leftarrow$ SolveInterpolation( $\Delta\boldsymbol{x}$ )
5: while $itr{+}{+} < maximum$ and $E > threshold$ do
6:     for $i = 1 \rightarrow N$ do
7:         $\boldsymbol{x}_{KDi} \leftarrow$ SimulateKD( $\boldsymbol{x}_K + \boldsymbol{f}$, $t_i$, $\delta$ )
8:         $\boldsymbol{e}_i \leftarrow$ compute using Equation 1 or 4
9:         $\tilde{\boldsymbol{x}}_{KDi} \leftarrow$ SimulateKD( $\boldsymbol{x}_K + \boldsymbol{f} + \boldsymbol{e}_i\phi_i$, $t_i$, $\delta$ )
10:        $\Delta\boldsymbol{x}_i \leftarrow$ compute using Equation 3 or 5
11:        $\boldsymbol{f} \leftarrow$ SolveInterpolation( $\Delta\boldsymbol{x}$ )
12:    end for
13:    $E \leftarrow 0$
14:    for $i = 1 \rightarrow N$ do
15:        $\boldsymbol{x}_{KDi} \leftarrow$ SimulateKD( $\boldsymbol{x}_K + \boldsymbol{f}$, $t_i$, $\delta$ )
16:        $E \leftarrow E + \|p(\boldsymbol{x}_{KDi}) - p_i\|$ or $\|\boldsymbol{x}_i - \boldsymbol{x}_{KDi}\|$
17:    end for
18: end while
```

Solving for the basis function coefficients λ_i is fast. To solve the interpolation function, we can compute an LU decomposition, from which we can find λ_i using a back solve. Repeated solves of the interpolation function can be done quickly because we can reuse the same decomposition (the basis functions and their centers do not change).

Note that the inner loop update could skip an update for a given constraint if its contribution to the error was known to be small. However, the size of this error can only be verified by recomputing the KD state as it is influenced by other changes to $\boldsymbol{f}$. The computation of x_{KDi} is the bulk of the cost.

4.3 Constraining velocities

When constraining a pose or an end effector position, we might also want to set constraints on velocities. For instance, we may want the hand of a character to touch the surface of a stationary object. The hand end effector must satisfy both position and velocity constraints, meaning it must reach the target at the time of contact and have zero velocity. The desired velocity can follow the original velocity of the animation or can be set to achieve a different velocity at the time of the constraint.

We can solve the IKD problem for constrained velocities in a similar manner to the position problem, and likewise solve for simultaneously constrained position and velocity. Again, the IKD solution comes from layering a correction overtop of the kinematic trajectory.

Suppose that at time t_i we have desired state velocity $\dot{\boldsymbol{x}}_i$ (or alternatively, a desired end effector velocity $\dot{p}_i$). Instead of adding a bell shaped curve to change the velocity, we will add a wiggle to change the velocity $\dot{\boldsymbol{x}}_K(t_i)$ without changing $\boldsymbol{x}_K(t_i)$. We use the derivative of the bell shaped position correction basis function as a basis function for setting the derivative,

$$\psi_i(t) = \frac{\partial}{\partial t}\phi_i(t), \tag{8}$$

though this function could likewise be selected by the animator.

For simplicity, suppose we are dealing with a set of N pairs of constraints, i.e., constraints on both position and velocity at times t_i, for $i = 1...N$. To deal with position and velocity constraints in close proximity we use an interpolation of the corrections $\Delta\boldsymbol{x}_i$ with velocities $\Delta\dot{\boldsymbol{x}}_i$ necessary to correct the kinematic trajectory. Thus, the interpolation function has the form

$$\boldsymbol{f}(t) = \sum_i^N(\lambda_i\phi_i(t) + \beta_i\psi_i(t)). \tag{9}$$

Again, the basis function coefficients λ and β can be found by solving the system of $2N$ linear equations for each dimension of the state, given by the required corrections and correction velocities:

$$\boldsymbol{f}(t_i) = \Delta\boldsymbol{x}_i, \quad \text{for } i = 1..N, \tag{10}$$

$$\frac{\partial \boldsymbol{f}(t_i)}{\partial t} = \Delta\dot{\boldsymbol{x}}_i, \quad \text{for } i = 1..N. \tag{11}$$

It is important to observe that we update the desired velocity correction $\Delta\dot{\boldsymbol{x}}_j$ by comparing the desired velocity $\dot{\boldsymbol{x}}$ with the velocity of the KD trajectory. The velocity of the dynamic simulation which produces $\boldsymbol{x}_{KD}(t)$ does *not* give us this KD velocity (i.e., it is not the dynamic simulation velocity which we want to control). Instead, we must approximate this KD state velocity from successive frames of the KD state,

$$\dot{\boldsymbol{x}}_{KD}(t_i) \approx \frac{1}{h}(\boldsymbol{x}_{KD}(t_i) - \boldsymbol{x}_{KD}(t_i - h)). \tag{12}$$

We measure the difference to set the velocity error $\dot{\boldsymbol{e}}_i$, with which we compute a new KD state, and ultimately find an update to the required velocity correction $\Delta\dot{\boldsymbol{x}}_i$ (with a computation similar to Equation 3).

In the above example, we are considering a target velocity on the entire state. If instead our constraint is only on the end effector of a skeleton, then the approach is slightly different. In this case, we compute the approximate KD end effector velocity,

$$\dot{p}_{KD}(t_i) \approx \frac{1}{h}(p(\boldsymbol{x}_{KD}(t_i)) - p(\boldsymbol{x}_{KD}(t_i - h))). \tag{13}$$

The difference between this velocity and the artist requested end effector velocity $\dot{p}_i$ is then mapped to a state error,

$$\dot{\boldsymbol{e}}_i = J^{+}(\dot{p}_i - \dot{p}_{KD}(t_i)). \tag{14}$$

where J^{+} is a pseudoinverse of the end effector Jacobian $J = \partial p/\partial \boldsymbol{x}$ evaluated at pose $\boldsymbol{x}_{KD}(t_i)$. Again, this error is used to update the required velocity correction $\Delta\dot{\boldsymbol{x}}_i$, and the process is repeated until our IKD algorithm has converged or we reach a maximum number of iterations.

4.4 Passive deformable object IKD

While skeletal motion plays an important role in character animation, animators may wish to have more control over passive deformable objects attached to kinematically driven bodies. Controlling secondary dynamics of such deformations can be tricky since the motion can only be indirectly edited by changing the kinematic motion that drives the secondary dynamics.

For example, consider the scenario shown in Figure 6 (d), where a character with a floppy hat must walk through a door without the hat hitting any part of the door frame. In this case, the hat would hit the top of door frame at time t_i. We can use IKD to set a constraint that the tip of the hat must be at a position just below the door frame at time t_i. In our example, the hat is a passive elastic system rigidly attached to the head of the character, and the character motion is purely kinematic (and it must be altered to change the trajectory of the tip of the hat).

IKD can be used to control the tip of the hat using the skeletal IKD technique described in Section 4.1, with a small adjustment. At each iteration of the IKD algorithm, we fix the end effector position, i.e., the tip of the hat, based on its kinodynamic position at time t_i (even though it is not fixed with respect to the character's head). The IKD solution involves a change in the character's posture, allowing the tip of the hat to miss the door frame (see the accompanying video).

4.5 Dynamic blend shape IKD

While the previous subsections focus on skeletal animation, the same ideas are applicable to elastic tissue deformation. Particularly in the context of facial animation, this articulated deformation is tediously authored by animators by keyframing linearly blend shape targets [19]. Overlaying secondary jiggle and other dynamic nuance currently comes at the cost of letting dynamics have the "final word" on the animation, with no guarantees of hitting certain expressions. IKD allows one to overlay this desired secondary dynamics in a kinematic setting and further specify critical poses as target shapes to be precisely interpolated, independent of the kinematically authored blend shape animation. A loosened facial animation can also be kept in sync with the environment (like taking a puff from a cigarette or sip from a glass) or an audio track by adding checkpoints from the kinematic trajectory as IKD targets, so the final facial trajectory has a limited deviation from the kinematic input. We implement IKD as a deformation that tracks control points on a shape using springs and dampers as in work by Müller et al. [16]. Figure 5 shows examples of pose constraints applied to a kinodynamic trajectory for two different characters.

The inverse kinodynamic solution follows the same algorithm presented above, with the correction update following Equation 3, which is very easy to compute as we simply need the difference between the kinodynamic state and the target. The techniques for dealing with multiple constraints and velocity constraints are likewise similar to those describe above.

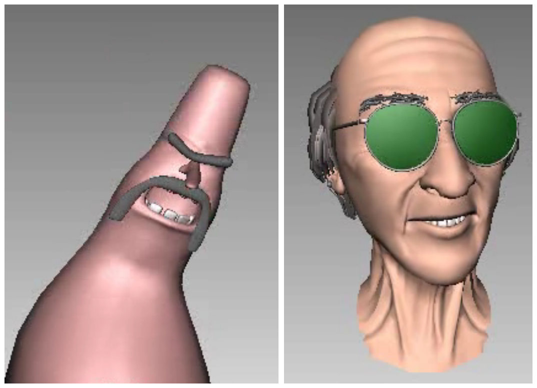

Figure 5: Two facial animation IKD examples (see accompanying video). Left, a temporal pose constraint produces a head tilt. Right, a temporal pose constraint produces a smile.

5 Temporal window selection

Setting the size of the temporal window δ has an important influence on the quality and cost of the kinodynamic trajectory. We want a small window to make it cost effective to simulate kinodynamic states on the fly, but the window also needs to be long enough to produce the desired secondary dynamics effects.

We find that it is easy to select a reasonable window by hand. Given some fixed gains for the PD controller, this can be done by simulating the physical system in response to an impulse and visually selecting the time at which vibrations are no longer visible. Instead of an isolated impulse, we typically focus on a maximum acceleration, i.e., an abrupt change, in a kinematic trajectory defined by a motion capture clip or a keyframe animation. We alternate between adjusting the temporal window size, and scrubbing back and forth on the time line to observe the results. We stop adjusting the window size when we are satisfied that we have selected the smallest window that does not prematurely truncate damped vibrations caused by the maximum acceleration in the kinematic trajectory. This is the technique we use for all of our examples.

6 Results and discussion

Please refer to the accompanying video for examples of our results. Figure 6 shows snapshots from the video, highlighting different scenarios that demonstrate the IKD techniques presented in Section 4. The video also includes a work-flow example demonstrating the interactivity of our system for skeletal animation IKD.

In our discussion of time window selection, we noted that the maximum acceleration in the kinematic trajectory will influence the size of the temporal windows (large accelerations will require longer temporal windows in order for oscillations produced by these accelerations to become imperceptible). This is also true for the altered trajectory which includes the correction to solve a given IKD problem. We are using smooth bell shaped curves to add this displacement, so generally the accelerations due to the correction will be small. But if we set the temporal width of this curve to be small, then the IKD solution will need to involve a very large displacement to the kinematic trajectory to force the dynamic trajectory to the desired target, thus requiring larger temporal windows for computing a KD state. While we impose no explicit restrictions on the physical simulation of characters, our approach is largely suited to well-conditioned and continuous simulations.

The shape of the correction curve we use to modify the kinematic motion directly affects the motion which is produced. We use Gaussian shaped curves in our examples because they are simple and smooth. We effectively treat them as if they have compact support, and could easily use any other ease-in-ease-out curve of a desired shape and support, and we leave the selection of this curve to the animator. That is, the width of the Gaussian is selected by the animator; a wide curve will produce a smooth anticipatory motion, while a short curve will produce a motion that abruptly moves to meet the constraint with a larger acceleration (and in turn, produces a larger follow through). While we only look at symmetric curves, any smooth artist created ease-in ease-out curve can be used. For instance, a non-symmetric correction curve can be designed to create a quick reaction followed by a slow return to the unmodified trajectory.

While we are adding constraints to deal with contact, we require the artist to specify these constraints. Although contacts may naturally happen in the dynamic simulations that produce our kinodynamic states, we will only have a "memory" of contacts that happen in the temporal window. For instance, we cannot correctly handle a braid of hair which is normally at rest down the back of a character but flips over a shoulder with the turn of a head. As such, these cases are comfortably handled as pure skeletally driven dynamics, but we hope to address such scenarios in the future by analysing collision events to adaptively vary the KD window size.

6.1 Convergence

It is important to discuss issues with the convergence of our IKD algorithm. If there are many abrupt motions in the kinematic trajectory, then the resulting simulation could be chaotic. As such, we might not expect a small change in the joint angles to produce a predictable result, even if we smoothly and slowly blended in and out of this desired trajectory. While we do not assume linear dynamics, we do assume that the function mapping $\boldsymbol{x}_K$ to $\boldsymbol{x}_{KD}$ is smooth "enough", as is the case for all of our example systems.

While the convergence rate of our IKD algorithm depends on the actual scenario, Figure 7 compares the convergence rates achieved using different temporal widths of the correction function, for the target punching example from Figure 4. While convergence can be slower when very small temporal widths are used, the number of iterations can be reduced by damping the correction adjustment computed in Equation 3 or 5 (see curves marked *scaled* in Figure 7).

We find that IKD converges quickly under a wide variety of constraints. Figure 8 shows the error in cm after 6 iterations for varying target positions in the punch example. The error is small in all

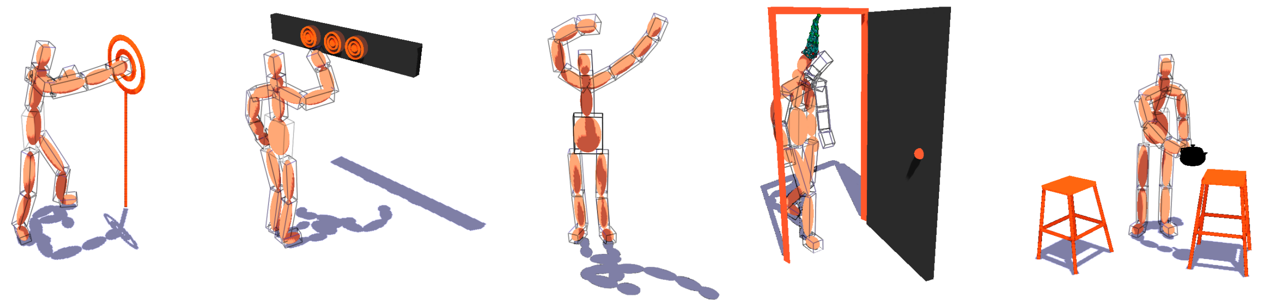
(a) (b) (c) (d) (e)

Figure 6: From left to right, (a) punch, (b) control panel, (c) YMCA's "C" Pose , (d) passive deformable hat, (e) position constraint for grasp

cases, except for target constraints which are physically out of reach of the character.

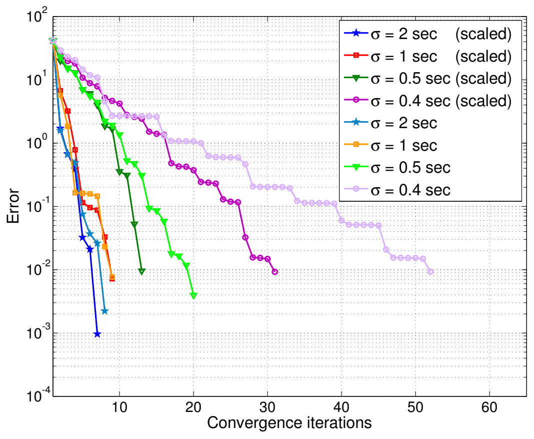

Figure 7: IKD convergence rates for the punch scenario using a bell shaped correction function with big temporal width (1 sec), and a small temporal width (0.5 sec), with error measured in cm, and error threshold 10^{-2} cm. IKD convergence can be slower when a small temporal width is used, but this can be improved slightly by damping the correction adjustment by 0.8 (i.e., scaled).

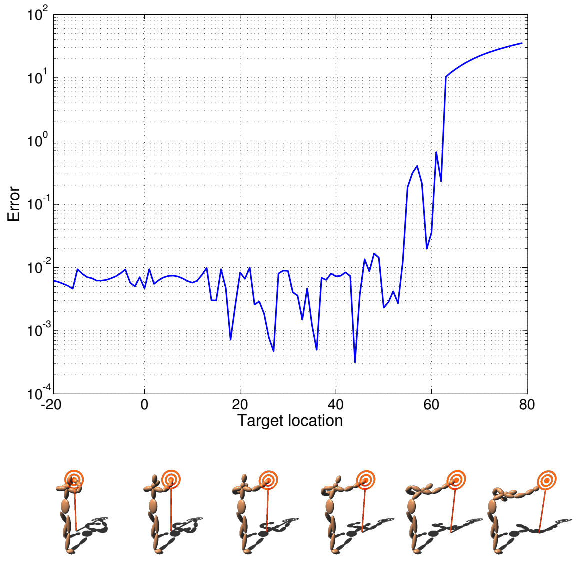

Figure 8: IKD error in cm after 6 iterations for varying target locations in the punch scenario. The error consistently falls to less than 0.1 mm after 6 iterations, except when the target is nearly out of reach.

We have noticed that our IKD algorithm can fail to make further progress once the error falls below 10^{-2} cm. We believe this is because we are using the Open Dynamics Engine (ODE) to compute the simulations that produce our KD states. While repeating simulations using the same initial conditions should produce the same results, aggressive optimizations within ODE make use of randomization. This does not present a problem as the error in end effector placement is significantly smaller than the overall size of the articulated character.

6.2 Implementation and Timings

IKD Skeletal animation examples were generated with our Java implementation which uses ODE (Open Dynamics Engine) to simulate the forward dynamics. On an Intel(R) Core i7, 3.2 GHz processor, the KD takes roughly 0.01 s to generate the resulting frame for a time window of 0.3 s (30 frames), which allows for interactive scrubbing of the time line.

IKD has also been implemented as a Maya 2011 deformer for control point shapes that track a kinematic trajectory using a spring and damper simulation. On an Intel i7, 1.87 GHz processor, the model in Figure 5 (approximately 1200 vertices) takes 13.56 s (of which 7.34 s is external to the KD algorithm) to update 50 frames with a KD window of 1 s (25 frames) resulting in a reasonable interactivity of 8.03 fps.

7 Conclusions

A demonstration of the Maya implementation to a few keyframe animators was positively received. From a workflow standpoint, the animators felt they would have to consciously omit keyframing dynamic nuances but this would be a welcome change allowing them focus on the primary motion. For the approach to be used in practice they expressed a need for interface tools that make the addition and management of IKD targets user friendly. Our current implementation, while interactive for skeletal animation, is only interactive for face blend shapes with around 1000 control vertices. The vectorizable nature of our algorithm, however, makes it a good candidate for a faster GPU implementation. In future work we would like to address the coupling of kinodynamic trajectories with fully dynamic environments via adaptive kinodynamic window sizes that are aware of collision events and other discontinuities in a full physical simulation. In summary, we propose the concept of Inverse Kinodynamics and present a first algorithm which opens up new possibilities for editing traditional keyframe animations that are augmented with secondary dynamics.

References

[1] Y. Abe and J. Popović. Interactive animation of dynamic manipulation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '06, pages 195–204, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[2] B. F. Allen, D. Chu, A. Shapiro, and P. Faloutsos. On the beat!: Timing and tension for dynamic characters. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 239–247. Eurographics Association, 2007.

[3] B. F. Allen, M. Neff, and P. Faloutsos. Analytic proportional-derivative control for precise and compliant motion. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 6039 –6044, may 2011.

[4] A. Angelidis and K. Singh. Kinodynamic skinning using volume-preserving deformations. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '07, pages 129–140. Eurographics Association, 2007.

[5] J. Barbič and J. Popović. Real-time control of physically based simulations using gentle forces. *ACM Trans. on Graphics (SIGGRAPH Asia 2008)*, 27(5):163:1–163:10, 2008.

[6] M. Bergou, S. Mathur, M. Wardetzky, and E. Grinspun. TRACKS: Toward Directable Thin Shells. *SIGGRAPH (ACM Transactions on Graphics)*, 26(3):50, Jul 2007.

[7] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popović. Interactive skeleton-driven dynamic deformations. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 586–593, 2002.

[8] P. Coleman, J. Bibliowicz, K. Singh, and M. Gleicher. Staggered poses: a character motion representation for detail-preserving editing of pose and coordinated timing. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 137–146. Eurographics Association, 2008.

[9] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *J. ACM*, 40:1048–1066, November 1993.

[10] S. Jain and C. K. Liu. Interactive synthesis of human-object interaction. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pages 47–53. ACM, 2009.

[11] D. L. James and D. K. Pai. Dyrt: dynamic response textures for real time deformation simulation with graphics hardware. In *SIGGRAPH*, pages 582–585, 2002.

[12] M. Kass and J. Anderson. Animating oscillatory motion with overlap: wiggly splines. *ACM Trans. Graph.*, 27:28:1–28:8, August 2008.

[13] J. Kim and N. S. Pollard. Fast simulation of skeleton-driven deformable body characters. *ACM Trans. Graph.*, 30:121:1–121:19, October 2011.

[14] N. Lockwood and K. Singh. Biomechanically-inspired motion path editing. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, pages 267–276, New York, NY, USA, 2011. ACM.

[15] A. McNamara, A. Treuille, Z. Popović, and J. Stam. Fluid control using the adjoint method. *ACM Trans. Graph.*, 23:449–456, August 2004.

[16] M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24:471–478, July 2005.

[17] M. Neff and F. Eugene. Modeling tension and relaxation for computer animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '02, pages 81–88, New York, NY, USA, 2002. ACM.

[18] N. Nguyen, N. Wheatland, D. Brown, B. Parise, C. K. Liu, and V. Zordan. Performance capture with physical interaction. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 189–195. Eurographics Association, 2010.

[19] F. Pighin, J. Hecker, D. Lischinski, R. Szeliski, and D. H. Salesin. Synthesizing realistic facial expressions from photographs. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 75–84, 1998.

[20] J. Popović, S. M. Seitz, and M. Erdmann. Motion sketching for control of rigid-body simulations. *ACM Trans. Graph.*, 22:1034–1054, October 2003.

[21] J. Popović, S. M. Seitz, M. Erdmann, Z. Popović, and A. Witkin. Interactive manipulation of rigid body simulations. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 209–217, 2000.

[22] M. H. Raibert and J. K. Hodgins. Animation of dynamic legged locomotion. *SIGGRAPH Comput. Graph.*, 25:349–358, July 1991.

[23] K. W. Sok, K. Yamane, J. Lee, and J. Hodgins. Editing dynamic human motions via momentum and force. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 11–20, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[24] A. Treuille, A. McNamara, Z. Popović, and J. Stam. Keyframe control of smoke simulations. *ACM Trans. Graph.*, 22:716–723, July 2003.

[25] A. Witkin and M. Kass. Spacetime constraints. *SIGGRAPH Comput. Graph.*, 22:159–168, June 1988.

[26] C. Wojtan, P. J. Mucha, and G. Turk. Keyframe control of complex particle systems using the adjoint method. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 15–23. Eurographics Association, 2006.

[27] K. Yin, M. B. Cline, and D. K. Pai. Motion perturbation based on simple neuromotor control models. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, PG '03, pages 445–, Washington, DC, USA, 2003. IEEE Computer Society.

[28] V. B. Zordan and J. K. Hodgins. Motion capture-driven simulations that hit and react. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '02, pages 89–96, New York, NY, USA, 2002. ACM.

[29] V. B. Zordan, A. Majkowska, B. Chiu, and M. Fast. Dynamic response for motion capture animation. *ACM Trans. Graph.*, 24:697–701, July 2005.