Inverse Kinodynamics: Editing and Constraining Kinematic Approximations of Dynamic Motion

Paul G. Kry[a], Cyrus Rahgoshay[a], Amir Rabbani[a], Karan Singh[b]

[a]*School of Computer Science, Centre for Intelligent Machines, McGill University*
[b]*Department of Computer Science, University of Toronto*

Abstract

We present inverse kinodynamics (IKD), an animator friendly kinematic work flow 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 time 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 describe both manual and automatic procedures for selecting the kinodynamic window size necessary to approximate the dynamic trajectory to a given accuracy. We demonstrate the convergence properties of our IKD approach, and give details of a typical work flow example that an animator would use to create an animation with our system. 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.

Keywords: inverse kinematics, secondary dynamics, key framing

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 [1, 2, 3], 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 [5].

The KD *window size* determines how far into the recent past we start a physical simulation in order to compute a KD state. We must formulate 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.

Kinodynamics is an interesting approach for interactive character animation, where animators can continue to leverage a direct history-free kinematic work flow, coupled with the benefits of arbitrary physically simulated secondary dynamics. The problem that we are solving is the inverse kinodynamics problem, and our solution allows an animator to easily edit kinodynamic trajectories such that desired constraints can be met. Specifically, the contributions of this paper are

- an iterative inverse kynodynamics (IKD) solver with fast convergence properties;
- details on how to implement our IKD solver for a wide range of scenarios: constraints on full poses or hands and feet in character animation, position and velocity constraints, multiple overlapping constraints, and constraints on passive deformable objects and in facial animation;
- an automatic method for selecting kinodynamic time windows from the physical parameters of the system and acceleration bounds on the kinematic trajectory.

We also discuss limitations, timings, convergence rates, and we describe a typical work flow example that an animator would use to create an animation with our system.

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 [1, 2, 3]. 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 [6]. These techniques provide an important richness to an animation; while the style of the results are controllable by adjusting the 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 as a separate problem.

In constrast to the simulations that provide secondary dynamics, it is the direct local control and WYSIWYG interface provided by forward and inverse kinematics techniques that animators primarily use in the creation of character animation. As a result, there has been a vast amount of research on inverse kinematics over the years, for instance, combining direct and inverse control during editing [7], using nonlinear programming [8], using priority levels to manage conflicting constraints [9], or alternatively, singularity-robust inverse computations [10] or damped least squares [11]. In this work our focus is on a problem similar to inverse kinematics, but in the new setting of kinodynamics. We leave the issue of solving inverse kinodynamics in the presence of conflicting constraints for future work.

The rest of 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 [12, 13], fluids [14, 15], and cloth [16]. 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 [17]. Physically based articulated character control has received a vast amount of interest. Building on the seminal work of locomotion control [18], it is now possible to have, for instance, animation of physically based motions that respond naturally to perturbations [19, 20, 21], maintain balance during locomotion [22, 23], and editable animations of dynamic manipulations that respect the dynamic interactions between characters and objects [24]. Our work is very different from these approaches, and is instead more closely related to work by Allen et al. [25], 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 [26] 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 [27]. Also relevant is work that uses analytic PD control trajectories for compliant interpolation [28], and work on generating physically plausible motion from infeasible reference motion using a dynamics filter [29].

In contrast to the work on controlling fully dynamic simulations, we are addressing a simplified problem due to the finite time window involved in simulating the state at a given time in a kinodynamic trajectory. 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. We use correction curves with a limited temporal width to solve the IKD problem. In a scenario with many constraints at different times, the correction resembles a smooth displacement function, such as the B-splines that are commonly used in solving spacetime constraint problems (for instance, in the motion editing work of Gleicher [30]). An important difference is that our displacements are applied on the reference kinematic trajectory, thus the final motion is the product of a physical simulation as opposed to a displaced motion that satisfies physical constraints. In a different approach, with similar objectives to our own work, Kass and Anderson [31] 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 problems such as skeletal animation.

Specifically with respect to inverse kinematics, Boulic et al. [32] include a control of the center of mass that generates character postures with improved plausibility by satisfying static balance constraints. Also within a purely kinematic setting, Coleman et al. [33] 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.

Mixing kinematics and dynamics to get the best of both has promise for authoring motion in real time. For instance, Nguyen et al. [34] 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 [35], or with biomechanically inspired constraints [36]. 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 $\boldsymbol{x}_K$ 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. Figure 1 shows an example of KD trajectories computed with different time windows, which can also be seen in the supplementary video. Note how the history-free KD trajectories capture the visual behavior of the actual dynamics over a range of window sizes.

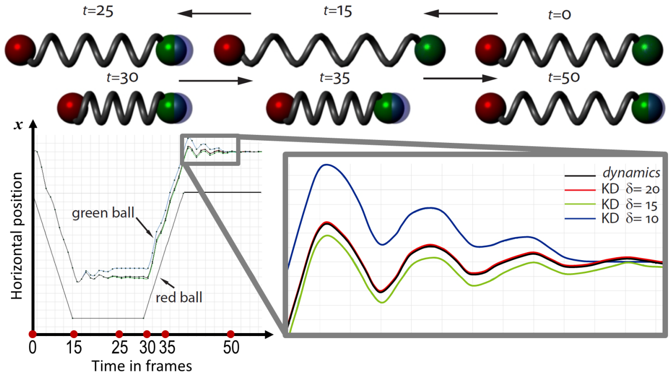


Figure 1: KD trajectories for the green ball: Kinematically the green ball is rigidly connected to the keyframed red ball, with spring dynamics overlaid. A number of frames of the KD trajectory ($\delta = 15$) are shown, with the full dynamics solution for the green ball overlaid in blue (top). KD trajectories with 3 window sizes are shown in relation to a full dynamics solution (bottom).

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. However, there may be specific times in the animation where we need constraints to be met.

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 and an IKD modification are 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 time 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 (for instance, based on screen pixels or a percentage of the object size). 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 shaped correction curve in this work, but a variety of artist designed curves that define a smooth ease-in ease-out correction can be used, as described in Section 6.

3.1. Inverse kinodynamics

Here we formalize our core approach to solving the IKD problem. Let SimulateKD($\boldsymbol{x}_K$, t_i, $\boldsymbol{\delta}$) be the procedure for computing $\boldsymbol{x}_{KDi}$, the KD state at t_i for kinematic trajectory $\boldsymbol{x}_K$. We

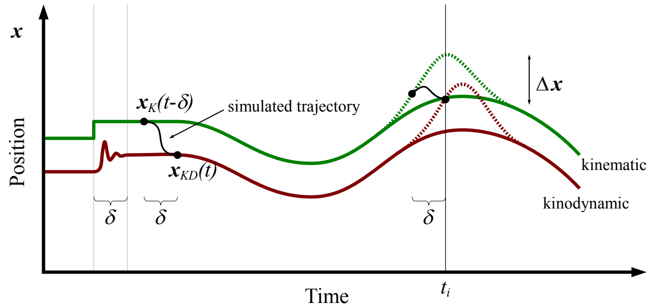


Figure 2: An illustration of how we modify a kinematic trajectory to create a kinodynamic trajectory that satisfies the constraint that the original kinematic state be produced at time t_i.

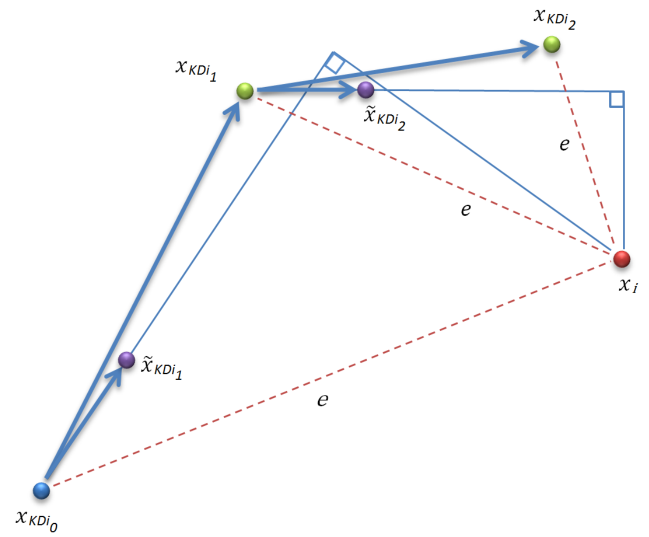


Figure 3: An illustration for a 2D state of two steps of the IKD iteration for a constraint at time t_i. At bottom left, $\boldsymbol{x}_{KDi_0}$ is the initial KD state at time t_i, which is far from target $\boldsymbol{x}_i$. A correction based on $\boldsymbol{e}$ results in the modified KD state $\tilde{\boldsymbol{x}}_{KDi_1}$, 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.

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 (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 $\tilde{\boldsymbol{x}}_{KDi} = \mathrm{SimulateKD}(\boldsymbol{x}_K + \boldsymbol{e}_i\phi_i, t_i, \delta)$ 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. 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 $\Delta\boldsymbol{x}_i = \left(\boldsymbol{e}_i \cdot \boldsymbol{d}_i/||\boldsymbol{d}_i||^2\right)\boldsymbol{e}_i$. Figure 3 shows a 2D illustration of how this scaling encourages good progress toward the target on each iteration. 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 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 [6].

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 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. For instance, we might want an animation of a drummer to hit a set of cymbals at precise times in quick succession (see also Figure 7-b for an example that involves hitting targets on a control panel). Note that two end effectors, for instance two hands, each with a constraint at the *same* time, can be solved with the technique in Section 4.1 by letting the IK problem involve two end effectors; it is only when constraints are at different times that we need a different approach.

Temporally coupled constraints 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 time 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 KD time window size used for the kinodynamic simulation.

While we may typically be able to solve most constraints independently, if the KD time window is large and the temporal width of the correction curve is small then we may need to solve the constraints in order from earliest to latest. Here, we specifically consider the harder case where the temporal width is large enough to cause the constraints to be temporally coupled, requiring the constraints to 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,

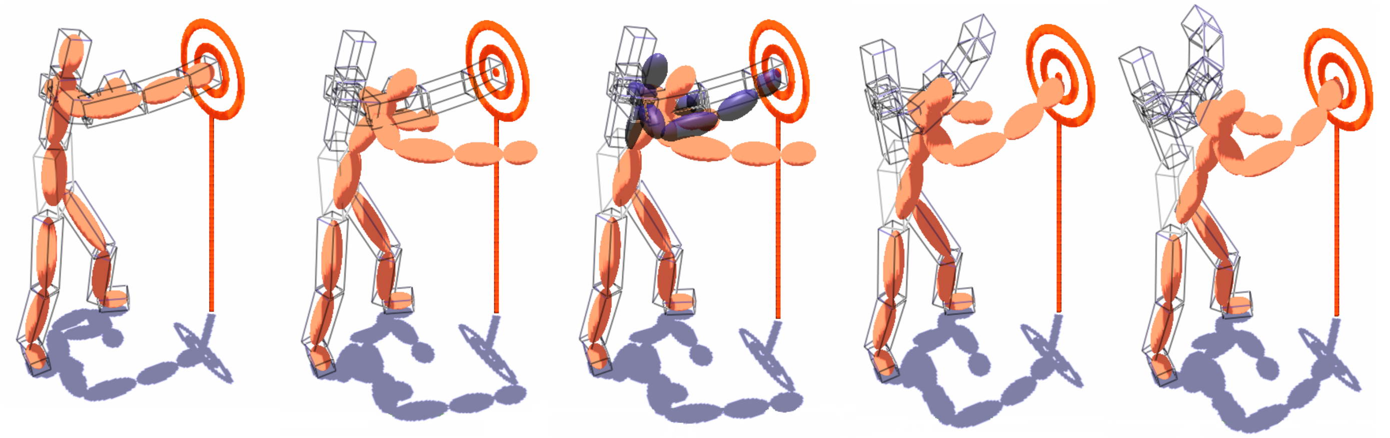
$$\boldsymbol{f}(t_i) = \Delta\boldsymbol{x}_i, \quad \text{for } i = 1..N. \tag{7}$$

Note that the coefficients λ_i are vectors with the same dimension as $\boldsymbol{f}$, that is, 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 an 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.

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



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

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.

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

$itr \leftarrow 0$
$E \leftarrow \infty$
$\Delta\boldsymbol{x}_i \leftarrow 0$, for $i = 1..N$
$\boldsymbol{f} \leftarrow$ SolveInterpolation($\Delta\boldsymbol{x}$)
while itr++ $< maximum$ and $E > threshold$ **do**
  for $i = 1 \rightarrow N$ **do**
    $\boldsymbol{x}_{KDi} \leftarrow$ SimulateKD($\boldsymbol{x}_K + \boldsymbol{f}$, t_i, δ)
    $\boldsymbol{e}_i \leftarrow$ compute using Equation 1 or 4
    $\tilde{\boldsymbol{x}}_{KDi} \leftarrow$ SimulateKD($\boldsymbol{x}_K + \boldsymbol{f} + \boldsymbol{e}_i\phi_i$, t_i, δ)
    $\Delta\boldsymbol{x}_i \leftarrow$ compute using Equation 3 or 5
    $\boldsymbol{f} \leftarrow$ SolveInterpolation($\Delta\boldsymbol{x}$)
  end for
  $E \leftarrow 0$
  for $i = 1 \rightarrow N$ **do**
    $\boldsymbol{x}_{KDi} \leftarrow$ SimulateKD($\boldsymbol{x}_K + \boldsymbol{f}$, t_i, δ)
    $E \leftarrow E + \|p(\boldsymbol{x}_{KDi}) - p_i\|$ or $\|\boldsymbol{x}_i - \boldsymbol{x}_{KDi}\|$
  end for
end while

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 tap the surface of a stationary object without penetrating the surface. 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 at that time. Alternatively, 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, that is, constraints on both position and velocity at times t_i, for $i = 1 \ldots N$. 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:

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

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

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 (it is not the dynamic simulation velocity that 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 $\Delta\dot{\boldsymbol{x}}_i$, using 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

The previous sections present solutions that deal with constraints on skeletal motion. While this 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. Fortunately, the same techniques can be applied to solve IKD problems on deformable objects.

For example, consider the scenario shown in Figure 7-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. In this example it is 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 kinodynamic position of the tip of the hat to miss the door frame (see the accompanying video).

4.5. Dynamic blend shape IKD

The IKD problem for passive deformable objects discussed in Section 4.4 can be applied to elastic tissue deformation in a variety of scenarios. Particularly in the context of facial animation, deformation is tediously authored by animators by keyframing linearly blend shape targets [37]. 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. [38]. 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.

5. Time window selection

Setting the size of the KD time 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.

5.1. Manual time window selection

We find that it is easy to select a reasonable window by hand. Given 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 or an abrupt change in the kinematic trajectory defined by a motion capture clip or a keyframe animation. We alternate between adjusting the KD time window

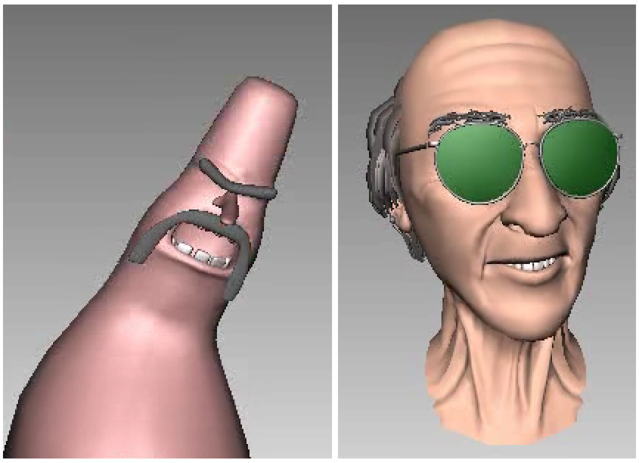


Figure 5: Two facial animation IKD examples (see accompanying video). Left, a temporal pose constraint produces a head tilt. Right, a temporal pose constraint produces a smile.

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.

5.2. *Automatic time window selection for 1D systems*

The KD time window can also be set automatically based on a computation that takes into account both the maximum acceleration of the kinematic animation and the physical properties of the kynodynamic system. Here we first consider the one dimensional example shown in Figure 1 (we will extend this to more complex examples in the next section). The system consists of a particle of mass m attached to a damped spring, the equation of motion is

$$m\ddot{x} + c\dot{x} + kx = 0, \tag{15}$$

and suppose the initial position be zero, $x_0 = 0$, which is likewise the equilibrium position of this system.

Let $J_{\max}$ be the maximum impulse that our particle experiences when following a given kinematic trajectory. We an compute this from the maximum acceleration in the kinematic trajectory as $J_{\max} = \ddot{x}_{\max} hm$ were h is the simulation step size. We want to know how long it will take for the system in Equation 15 to come to rest if activated by this maximum impulse. Here, we define the time at which the system comes to rest as the time, after which the trajectory will always remain within some small distance ε from the equilibrium position. We can solve this by first obtaining an analytic solution for the trajectory [39] using the maximum impulse to define an initial condition on the velocity $\dot{x}_0 = J_{\max}/m$. We are typically interested in mass, stiffness, and damping settings that result in an underdamped system, in which case $c^2 < 4mk$, giving a solution of the form

$$x(t) = Ae^{\gamma t}\sin(\omega t), \tag{16}$$

$$\gamma = -\frac{c}{2m}, \tag{17}$$

$$\omega = \frac{\sqrt{4mk - c^2}}{2m}, \tag{18}$$

$$A = \frac{\dot{x}_0}{\omega}, \tag{19}$$

where γ is the decay rate, ω is the frequency, and A is the amplitude. The function $Ae^{\gamma t}$ provides a bound on the magnitude of the simulation. Given a minimum perceptual magnitude ε, we can choose our time window δ as the time when the system appears to have come to rest, that is, by solving $Ae^{\gamma\delta} = \varepsilon$, obtaining

$$\delta = \frac{1}{\gamma}\ln\left(\frac{\varepsilon}{A}\right). \tag{20}$$

When the system is over damped, the solution can be written as a difference of exponentials,

$$x(t) = iA\left(e^{(\gamma - i\omega)t} - e^{(\gamma + i\omega)t}\right) \le iAe^{(\gamma - i\omega)t}. \tag{21}$$

The function is always positive because the velocity initial condition is positive, and thus it can be bounded by the exponential with positive coefficient (which also has the slower decay due to a smaller exponent). The time window can again be computed easily with a logarithm

$$\delta = \frac{1}{\gamma - i\omega}\ln\left(\frac{\varepsilon}{iA}\right). \tag{22}$$

However, if the system is critically damped, then the trajectory has the form $x(t) = \dot{x}_0 e^{\gamma t} t$, and the time window should be selected as the larger root of $\dot{x}_0 e^{\gamma t} t - \varepsilon = 0$. Alternatively, we can create an approximate exponential bound by avoiding the critically damped case though a small adjustment of the system parameters.

5.3. *Case study: time windows for multidimensional systems*

While the one dimensional example above is useful for understanding how we can choose a KD time window, we are interested in more complicated systems with multiple degrees of freedom, for instance, our floppy hat example described in Section 4.4. This system can be analyzed in a similar way, provided that a linearized version of the system yields a reasonable prediction of the behavior. This is typical in many elastic systems when deformations are small due to small forces or large stiffness. Using linearized forces at the equilibrium pose of the system, the idea is to diagonalize the system to break the problem up into a set of independent oscillators [40, 2]. While we specifically look at elastic deformable objects as a case study, the same approach can be used for our examples of PD control driven skeletal animation by computing the vibration modes of the skeletal system [41, 42].

Consider, for instance, a 3D multidimensional deformable elastic system such as a finite element model, written in terms of N nodal displacements $u \in \mathbb{R}^{3N}$, and recall that some nodes of this system are rigidly attached to a frame, or a "bone", that is driven by a kinematic animation. Assuming for now that the bone is stationary, the equations of motion can be written as

$$M\ddot{u} + C\dot{u} + Ku = 0 \tag{23}$$

where M is the mass matrix, K is the stiffness computed as the force gradient at the equilibrium pose, and $C = (\alpha M + \beta K)$ is the damping matrix that we assume is Rayleigh. The eigenvalue decomposition of $M^{-1}K$ gives the modal vibration basis matrix U, which provides a change of coordinates $u = Uq$, and allows for the system to be written as a set of independent oscillators,

$$m_i\ddot{q}_i + c_i\dot{q}_i + k_i q_i = 0. \quad i = 1..n, \tag{24}$$

where $c_i = (\alpha m_i + \beta k_i)$. With initial conditions $q(0) = 0$ and $\dot{q}(0)$ nonzero, the solution in the under-damped case is

$$q_i(t) = A_i e^{\gamma_i t}\sin(\omega_i t), \tag{25}$$

$$\gamma_i = -\frac{c_i}{2m_i}, \tag{26}$$

$$\omega_i = \frac{\sqrt{4m_i k_i - c_i^2}}{2m_i}, \tag{27}$$

$$A_i = \frac{\dot{q}(0)}{\omega_i}. \tag{28}$$

The exponential term decays at a minimum rate given by α, and higher frequency modes will have faster decays when β is nonzero. Because of this, we focus on a maximum activation (through an acceleration of the bone) of the lowest frequency mode, $i = 1$, in order to compute the time window. That is, instead of solving for the smallest δ such that $\sqrt{u^T u} < \varepsilon$ for all $t > \delta$, we assume that $u(t)$ will approximately equal $U_1 q_1(t)$ near our solution because the contribution of all the higher frequency modes will have long since become insignificant due to their much faster decay rates. Note that $\sqrt{q_1^T U_1^T U_1 q_1} = q_1$ because the eigenvector U_1 is unit length, thus we solve $A_1 e^{\delta\gamma_1} = \varepsilon$ for the time window size δ. Because the problem has been reduced to one dimension, we can use the same technique as in the previous section to solve for the time window in over-damped and critically damped cases too.

This leaves two points to clarify. First, the selection of the perceptual threshold ε, and second, the largest activation of this lowest mode that will let us specify the initial condition $\dot{q}_1(0)$.

We use the Euclidean norm to measure displacement from the rest pose, and it can be difficult to set the perceptual threshold because a model can be discretized with different numbers of nodes. We account for this by setting the threshold as the value where all nodes are displaced 0.1% of the object diameter d, or $\varepsilon = 0.001d\sqrt{N}$.

To define the largest activation of the first modal vibration, we follow the work of James and Pai [2] in specifying the effect of a 6 dimensional spatial velocity $\phi = (\omega^T, v^T)^T$ on the nodal velocities. With node i at position r_i, the velocity of the nodal

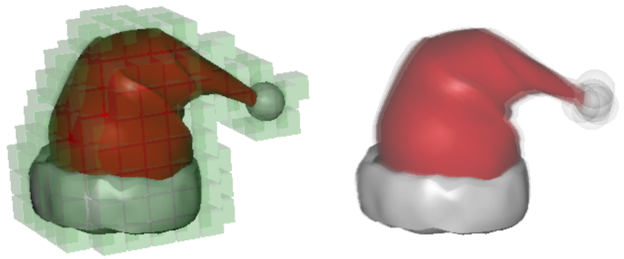


Figure 6: Left, the finite element model cubes in which a christmas hat is embedded. Right, a number of superimposed frames from the simulation showing the vibration.

points can be written $\dot{u} = \Gamma\phi$ where

$$\Gamma = \begin{pmatrix} -[r_1] & I \\ \vdots & \\ -[r_N] & I \end{pmatrix}, \tag{29}$$

and $[r_i]$ denotes the skew symmetric matrix that computes the cross product $r_i\times$. The inverse mode matrix lets us compute the modal velocities due to the spatial velocity, $\dot{q} = U^{-1}\Gamma\phi$, and we are specifically interested in the first mode. Notice that the six values in the top row of $U^{-1}\Gamma$ allow us to determine how large $\dot{q}_1$ can be for a given ϕ. Let $h_{\omega max}$ be the maximum among the absolute values of the first three components, and similarly h_{vmax} for the last three components. If we bound the magnitude of the largest angular and linear acceleration of the kinematic bone, then we can define scalars ω_{max} and v_{max} as the magnitude of largest angular and linear velocity that can be obtained in a single time step when starting from zero. Finally, we can set the initial condition of the first mode velocity as

$$\dot{q}_1(0) = h_{\omega max}\omega_{max} + h_{vmax}v_{max}. \tag{30}$$

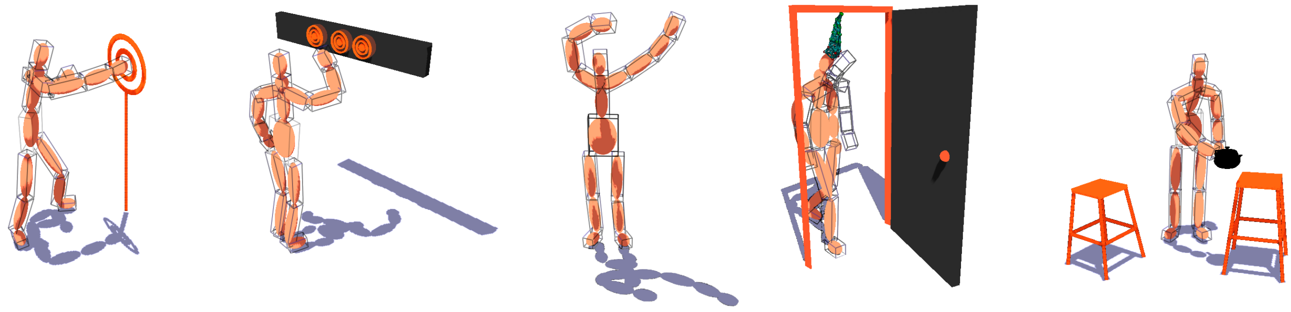
This gives a conservative setting of the initial velocity of the first mode that corresponds to the maximum accelerations we are willing to have in our kinematic animation.

Figure 6 shows a finite element model hat, for which we compute a time window $\delta = 0.568$ seconds with the approach described above where we let $\omega_{max} = 4$ radians per second and $v_{max} = 2$ meters per second.

6. Results and discussion

In this section we discuss a number of aspects of our IKD solution. We discuss convergence of the IKD solver in Section 6.1, the choice of correction curves in Section 6.2, the limitations of artist designed contact constraints in Section 6.3, issues in setting KD time windows in Section 6.4, and the specifics of our prototype implementation and timings in Section 6.5.

Figure 7 shows snapshots of examples that highlight the different scenarios demonstrating the IKD techniques presented in Sections 4 and 5. Please refer to the accompanying video to view the full animations. The video also includes a work flow example demonstrating the interactivity of our system for skeletal animation IKD, which is also described in Section 6.6.



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

Figure 7: From left to right, (a) punch, (b) control panel, (c) YMCA's "C" Pose , (d) passive deformable hat, (e) position constraint for grasp

6.1. Convergence

It is important to discuss 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) to allow us to use a linear approximation of the dynamics in our IKD solver.

While the convergence rate of our IKD algorithm depends on the actual scenario, Figure 8 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 8).

We find that IKD converges quickly under a wide variety of constraints. Figure 9 shows the error in cm after 6 iterations for varying target positions in the punch example. The error is small in all cases, except for target constraints which are physically out of reach of the character.

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. Correction curves

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

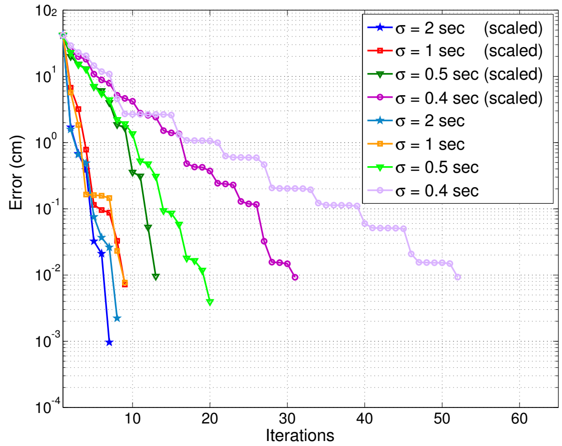


Figure 8: IKD convergence rates for the punch scenario using a bell shaped correction function with big temporal width (1 sec), and a small temporal width (0.5 sec), with error measured in cm, and error threshold 10^{-2} cm. IKD convergence can be slower when a small temporal width is used, but this can be improved slightly by damping the correction adjustment by a small amount, for instance, 0.8 in the examples labeled *scaled* in the legend.

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.

6.3. Contact constraints

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 KD time 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

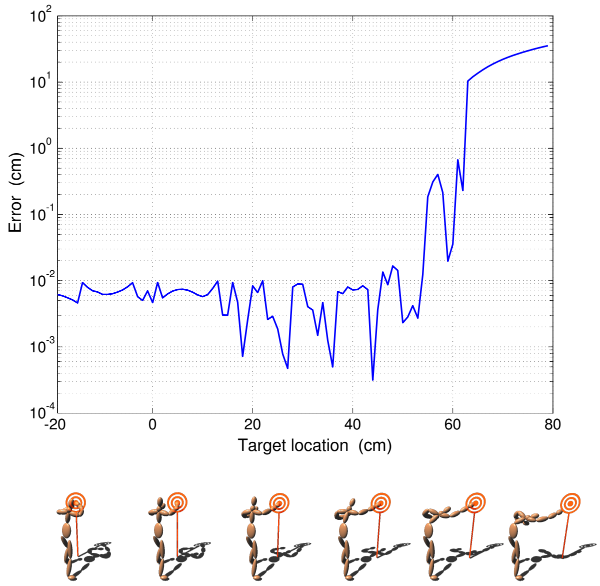


Figure 9: IKD error in cm after 6 iterations for varying target locations (cm) in the punch scenario. The error consistently falls to less than 0.1 mm after 6 iterations, except when the target is nearly out of reach.

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.

6.4. KD time windows

In the discussion of time window selection in Section 5, we noted that the maximum acceleration in the kinematic trajectory will influence the size of the time windows (large accelerations will require longer time 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 time 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 time window does not need to be the same for the entire time line and can be smaller when there are only small accelerations in the kinematic trajectory. Figure 10 shows measurements we have made of the Euclidean error of the KD trajectory compared to a normal simulation, for a variety of time windows. The system shown here consists of a floppy elastic hat attached to the head of an articulated character driven by motion capture as shown in Figure 7-d. The stiffness and damping parameters were set quite low, requiring a relatively large time window of 4 seconds before the KD trajectory to be imperceptably different from simulation for the accelerations in this example kinematic trajectory. However, the plot shows that there are moments during the animation where a much smaller KD time window can be used and still result in an imperceptible amount of error. This example illustrate the possible benefits from modulating the KD windows size across an animation, and is a problem we leave for future work.

6.5. Implementation and timings

Here we present timing details of the different implementations we have tested. Note that the skeletal motion examples make use of motion capture that was recorded at 100 Hz, and we also produce KD states and IKD solutions based on this same frame rate as opposed to resampling and simulating at the final video rate. In contrast, the facial animation examples have a 25 Hz simulation and frame rate.

The IKD Skeletal animation examples were generated with our Java implementation which uses ODE (Open Dynamics Engine) to simulate the forward dynamics. On an Intel Core i7 3.2 GHz processor, the KD state takes roughly 0.01 s to generate at each frame with a time window of 0.3 s, which allows for interactive scrubbing of the time line. Because the main cost of the IKD solution is the computation of KD states, an IKD solution for the punch scenario in Figure 4 can be computed in under half a second (that is, less than 50 iterations are required as shown in Figure 8).

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 at left in Figure 5 (approximately 1200 vertices) can be scrubbed at about 8 frames per second when using a KD time window of 1 s. We note that a large portion of the computation time is external to the KD algorithm, and an better optimized implementation would be possible by making use of the Maya plug-in application programming interface.

6.6. Interface and work flow example

A work flow example helps explain how animators can use IKD to create a desired animation and the types of controls they have for editing the animation. Figure 11 shows a snap shot of our Java implementation interface as it is being used during the example of creating an animation of a character punching a target.

The interface has a time frame slider that the animators can scrub back and forth to get to a desired frame of the animation, as well as three buttons for playing the animation forward, backward, and stopping playback. With these controls, the animator can play or scrub the animation to observe the character with secondary dynamics that exist in the kinodynamic trajectory. The interface allows various parameters to be adjusted, such as the physical parameters of the character, the kinodynamic window size, as well as the temporal width of the IKD correction function. Finally, there is a button that can be pressed to compute the IKD solution for any constraints that are set in the time line.

In our work flow example, the animator starts by loading an animation of a character throwing a punch. In this case, the

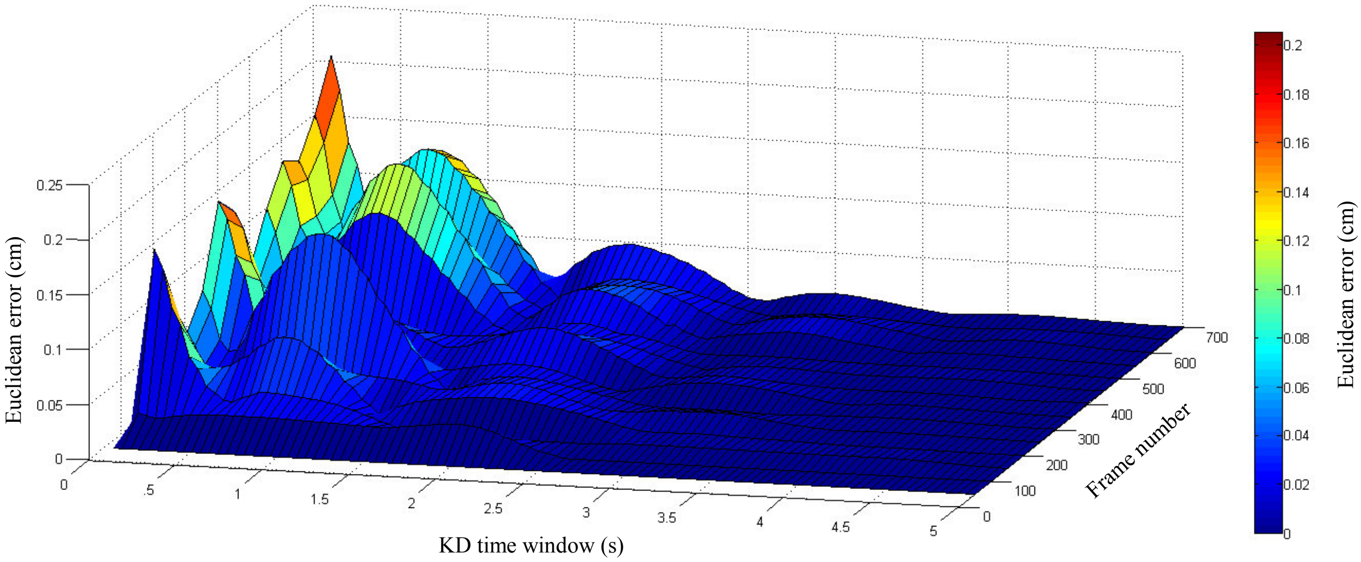


Figure 10: Euclidean error for the KD state of the hat example shown in Figure 7-d. The plots shows the error, a comparison between the KD state and the ground truth simulation, for different KD time window settings and across different frames of the animation. Notice that the KD time window necessary to meet a given level of error varies across the 7 seconds of animation (700 frames at 100 Hz).

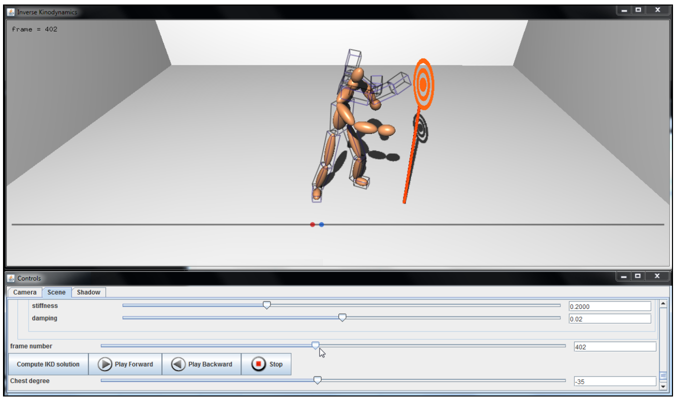
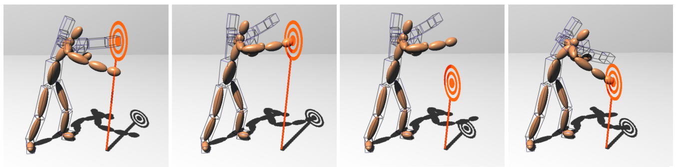


Figure 11: Work flow framework showing tools for animators to scrub back and forth in animation and change animation parameters.



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

Figure 12: (a) shows a KD state (orange) failing to punch a target. (b) is a modified motion capture pose (wire-frame) that produces a KD state which hits the target. The previous solution fails to punch a modified target position (c). A satisfied punch can be achieved by recomputing the IKD solution for the new target (d).

animation is a motion capture clip, though it could likewise be a kinematic trajectory produced with key frames. In Figure 11, the main window shows the kinematic pose with a wire cube style, while the KD state of the character is rendered with orange ellipsoids. The animator can interactively adjust the stiffness of the joints and the kinodynamic window size and observe the influence of these changes at the current pose. The animator can also immediately see the influence of these changes on the entire animation by scrubbing back and forth in the time line. In our example, the animator lowers the stiffness of the character which gives a drunken boxing style to the animation. Modifications to the kinematic trajectory can also be made, and the results viewed immediately. Our work flow example also includes a modification of the bend in the back to have the drunken boxer stand straight, or stoop further. Again, the results of these modifications can be seen immediately at the current pose, or viewed in the KD animation by scrubbing back and forth. This highlights the WYSIWYG editing interface that we provide to the animator.

Once the animator has adjusted tension, relaxation, or other aspects of the style of the motion, there will still be other constraints that need to be satisfied. In our work flow example, the animator would like to have the character punch a target at the desired time. Suppose the motion capture involves a punch at a target p_i at time t_i. The KD trajectory with the tension and relaxation parameters set to produce a drunken boxing style does not satisfy the target as shown in Figure 12-a. Setting the time line to the specific frame (t_i) and observing the error in the punch, the animator can use IKD to satisfy the constraint, producing the solution shown in Figure 12-b. If a different target is desired, for instance, a lower target as shown in Figure 12-c, then the animator can change the constraint and use IKD again to satisfy the constraint as show in in Figure 12-d.

The ability to see the effect of edits in real-time is what makes the interface a powerful tool for animators. Besides character animation, we also include deformable objects in our simulation and control their pose, as discussed in Section 4.4.

7. Conclusions

While the IKD interface itself is not a contribution of our work, a demonstration of the Maya implementation to a few keyframe animators was positively received. From a work flow 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.

Acknowledgments

This work was supported by NSERC, and GRAND NCE Project MOTION, and originated from discussions at the Bellairs Research Institute during the 2011 Workshop on Computer Animation.

References

[1] Capell S, Green S, Curless B, Duchamp T, Popović Z. Interactive skeleton-driven dynamic deformations. In: Proceedings of the 29th annual conference on Computer graphics and interactive techniques. SIGGRAPH '02. ISBN 1-58113-521-1; 2002, p. 586–93.

[2] James DL, Pai DK. Dyrt: dynamic response textures for real time deformation simulation with graphics hardware. In: SIGGRAPH. 2002, p. 582–5.

[3] Kim J, Pollard NS. Fast simulation of skeleton-driven deformable body characters. ACM Trans Graph 2011;30:121:1–121:19.

[4] Angelidis A, Singh K. Kinodynamic skinning using volume-preserving deformations. In: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation. SCA '07; Eurographics Association. ISBN 978-1-59593-624-4; 2007, p. 129–40.

[5] Donald B, Xavier P, Canny J, Reif J. Kinodynamic motion planning. J ACM 1993;40:1048–66.

[6] Neff M, Eugene F. Modeling tension and relaxation for computer animation. In: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation. SCA '02; New York, NY, USA: ACM. ISBN 1-58113-573-4; 2002, p. 81–8.

[7] Boulic R, Thalmann D. Combined direct and inverse kinematic control for articulated figure motion editing. Computer Graphics Forum 1992;11(4):189–202. doi:10.1111/1467-8659.1140189. URL `http://dx.doi.org/10.1111/1467-8659.1140189`.

[8] Zhao J, Badler NI. Inverse kinematics positioning using nonlinear programming for highly articulated figures. ACM Trans Graph 1994;13(4):313–36.

[9] Baerlocher P, Boulic R. An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. Vis Comput 2004;20(6):402–17.

[10] Yamane K, Nakamura Y. Natural motion animation through constraining and deconstraining at will. IEEE Transactions on Visualization and Computer Graphics 2003;9(3):352–60.

[11] Buss SR, Kim JS. Selectively damped least squares for inverse kinematics. Journal of Graphics Tools 2004;10:37–49.

[12] Popović J, Seitz SM, Erdmann M, Popović Z, Witkin A. Interactive manipulation of rigid body simulations. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques. SIGGRAPH '00. ISBN 1-58113-208-5; 2000, p. 209–17.

[13] Popović J, Seitz SM, Erdmann M. Motion sketching for control of rigid-body simulations. ACM Trans Graph 2003;22:1034–54.

[14] Treuille A, McNamara A, Popović Z, Stam J. Keyframe control of smoke simulations. ACM Trans Graph 2003;22:716–23.

[15] McNamara A, Treuille A, Popović Z, Stam J. Fluid control using the adjoint method. ACM Trans Graph 2004;23:449–56.

[16] Wojtan C, Mucha PJ, Turk G. Keyframe control of complex particle systems using the adjoint method. In: SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation. Eurographics Association. ISBN 3-905673-34-7; 2006, p. 15–23.

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

[18] Raibert MH, Hodgins JK. Animation of dynamic legged locomotion. SIGGRAPH Comput Graph 1991;25:349–58.

[19] Zordan VB, Hodgins JK. Motion capture-driven simulations that hit and react. In: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation. SCA '02; New York, NY, USA: ACM. ISBN 1-58113-573-4; 2002, p. 89–96.

[20] Yin K, Cline MB, Pai DK. Motion perturbation based on simple neuromotor control models. In: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications. PG '03; Washington, DC, USA: IEEE Computer Society. ISBN 0-7695-2028-6; 2003, p. 445.

[21] Zordan VB, Majkowska A, Chiu B, Fast M. Dynamic response for motion capture animation. ACM Trans Graph 2005;24:697–701.

[22] Yin K, Loken K, van de Panne M. Simbicon: simple biped locomotion control. ACM Transactions on Graphics 2007;26(3).

[23] Coros S, Beaudoin P, van de Panne M. Generalized biped walking control. In: ACM SIGGRAPH 2010 papers. SIGGRAPH '10; New York, NY, USA: ACM; 2010, p. 1–9.

[24] Abe Y, Popović J. Interactive animation of dynamic manipulation. In: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation. SCA '06; Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. ISBN 3-905673-34-7; 2006, p. 195–204.

[25] Allen BF, Chu D, Shapiro A, Faloutsos P. On the beat!: Timing and tension for dynamic characters. In: SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation. Eurographics Association. ISBN 978-1-59593-624-4; 2007, p. 239–47.

[26] Jain S, Liu CK. Interactive synthesis of human-object interaction. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA). ACM. ISBN 978-1-60558-610-6; 2009, p. 47–53.

[27] Witkin A, Kass M. Spacetime constraints. SIGGRAPH Comput Graph 1988;22:159–68.

[28] Allen BF, Neff M, Faloutsos P. Analytic proportional-derivative control for precise and compliant motion. In: Robotics and Automation (ICRA), 2011 IEEE International Conference on. 2011, p. 6039 –44.

[29] Yamane K, Nakamura Y. Dynamics filter - concept and implementation of online motion generator for human figures. IEEE Transactions on Robotics and Automation 2003;19(3):421–32.

[30] Gleicher M. Motion editing with spacetime constraints. In: Proceedings of the 1997 symposium on Interactive 3D graphics. I3D '97; New York, NY, USA: ACM; 1997, p. 139–49.

[31] Kass M, Anderson J. Animating oscillatory motion with overlap: wiggly splines. ACM Trans Graph 2008;27:1–8.

[32] Boulic R, Mas R, Thalmann D. A robust approach for the control of the center of mass with inverse kinetics. Computers & graphics 1996;20(5):693–701.

[33] Coleman P, Bibliowicz J, Singh K, Gleicher M. 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; Eurographics Association. ISBN 978-3-905674-10-1; 2008, p. 137–46.

[34] Nguyen N, Wheatland N, Brown D, Parise B, Liu CK, Zordan V. Performance capture with physical interaction. In: Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.

SCA '10; Eurographics Association; 2010, p. 189–95.
[35] Sok KW, Yamane K, Lee J, Hodgins J. Editing dynamic human motions via momentum and force. In: Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. SCA '10; Aire-la-Ville, Switzerland, Switzerland: Eurographics Association; 2010, p. 11–20.
[36] Lockwood N, Singh K. Biomechanically-inspired motion path editing. In: Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. SCA '11; New York, NY, USA: ACM. ISBN 978-1-4503-0923-3; 2011, p. 267–76.
[37] Pighin F, Hecker J, Lischinski D, Szeliski R, Salesin DH. Synthesizing realistic facial expressions from photographs. In: Proceedings of the 25th annual conference on Computer graphics and interactive techniques. SIGGRAPH '98. ISBN 0-89791-999-8; 1998, p. 75–84.
[38] Müller M, Heidelberger B, Teschner M, Gross M. Meshless deformations based on shape matching. ACM Trans Graph 2005;24:471–8.
[39] Meirovitch L. Analytical methods in vibrations. Macmillan series in advanced mathematics and theoretical physics; Macmillan; 1967.
[40] Bathe KJ. Finite Element Procedures in Engineering Analysis. Prentice-Hall, Inc.; 1982.
[41] Kry P, Revéret L, Faure F, Cani MP. Modal locomotion: animating virtual characters with natural vibrations. Computer Graphics Forum 2009;28(2):289–98. Special Issue: Eurographics 2009.
[42] Nunes RF, Cavalcante-Neto JB, Vidal CA, Kry PG, Zordan VB. Using natural vibrations to guide control for locomotion. In: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. I3D '12; New York, NY, USA: ACM; 2012, p. 87–94.