

Computer Puppetry: An Importance-Based Approach

To appear in ACM Transactions on Graphics journal

Hyun Joon Shin[†] Jehee Lee[†] Michael Gleicher[‡] Sung Yong Shin[†]

[†]Division of Computer Science, Department of Electrical Engineering & Computer Sciences

Korea Advanced Institute of Science & Technology, Taejon, Korea

{joony, jehee, syshin}@jupiter.kaist.ac.kr

[‡]Department of Computer Sciences, University of Wisconsin-Madison

Madison, WI, USA

gleicher@cs.wisc.edu

April 29, 2001

Abstract

Computer puppetry maps the movements of a performer to an animated character in real time. In this paper, we provide a comprehensive solution to the problem of transferring the observations of the motion capture sensors to an animated character whose size and proportion may be different than the performer. Our goal is to map as much of the *important* aspects of the motion to the target character as possible, while meeting the on-line, real-time demands of computer puppetry. We adopt a Kalman filter scheme that addresses motion capture noise issues in this

setting. We provide the notion of dynamic importance of an end-effector that allows us to determine what aspects of the performance must be kept in the resulting motion. We introduce a novel inverse kinematics solver that realizes these important aspects within tight real-time constraints. Our approach is demonstrated by its application to broadcast television performances.

1 Introduction

Computer puppetry [24] transforms the movements of a performer to an animated character in real time. The immediacy of computer puppetry makes it useful for providing live performances and as a visualization tool for traditional cinematic animation. However, this immediacy creates a number of challenges, as solutions to animation issues must be handled in an on-line, real-time manner. A computer puppetry system must capture the movements of the performer, interpret the important aspects of this motion, and determine the movements required to make the character reproduce these important aspects of the performance.

The challenges of mapping a motion from the performer to the target character become more difficult when the target character is of a different size and proportion than the performer [3, 5, 7, 12]. In such cases, the resulting motion of the character cannot exactly duplicate the original performer's. For example, we cannot simultaneously match the original joint angles and end-effector positions. Generally, to preserve the important aspects of the original motion we must alter the unimportant aspects of the motion. This process of adapting a motion for a new character is called *retargeting* [12, 17].

To date, solutions to computer puppetry issues have been limited to restricting the range of puppets that can be used or providing restrictive notions of what is important in motions. The latter implicitly limits the range of puppets since artifacts are introduced as the puppet's differences from the performer are increased.

In this paper we provide techniques that address the challenges of computer puppetry when the target character is different than the performer. Three major animation issues are addressed in a manner that fits within the real-time, on-line nature of computer puppetry:

1. The sensors used to capture the performer's motion are often noisy. Therefore, we provide a filtering technique that operates on-line with the efficiency required to process whole body motions in real time. We apply a Kalman Filter to rotation vectors, providing an orientation smoothing technique that is more efficient than previous methods.
2. The important aspects of the original performance must be determined such that these details can be reproduced in the resulting motion. We provide the notion of a dynamic importance measure that allows us to account for changing situations even when the future is unknown.
3. The resulting pose of the target character must be computed in a way that recreates the important aspects of the original. We provide a fast inverse kinematics solver that provides the necessary real-time performance and predictability.

Our solutions have been used to realize a computer puppetry system that has been used successfully to create animated television broadcasts.

We begin our discussion of computer puppetry by providing an overview of our approach. We examine previous solutions with respect to the issues raised in the overview. The components of our approach are then detailed in Sections 3, 4, and 5. An analysis in Section 6 reviews why our approach avoids introducing unwanted artifacts such as temporal discontinuities. Our experimental results are provided to support our approach. We conclude with a summary and discussion of future directions.

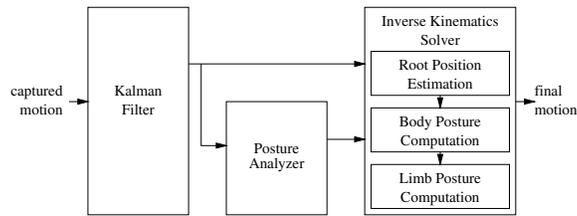


Figure 1: Overall structure

2 Overview

Computer puppetry requires the captured movements of the performer to be mapped to the target character in real time. As shown in Figure 1, our approach for on-line motion retargetting divides the task into three phases. First, a filtering phase “cleans” the sensor data to remove artifacts of the motion capture device. A second phase examines this filtered motion and determines the importance value of every end-effector in relation to its environment. A final phase computes a pose for the target character that achieves as much of the important aspects as possible. In this section, we provide an overview of these components and survey their relationship to previous work.

2.1 On-Line Filtering of Orientations

In general, captured motion data are noisy. The real-time sensors required for computer puppetry are particularly problematic in this regard. However, because of the dense sampling rates and signal characteristics of motion capture data, low-pass filtering is an effective tool to suppress noise in the captured data. This is challenging for three reasons:

1. Because computer puppetry is an on-line application, standard off-line filters cannot be employed.
2. Because the orientation space is highly non-linear, standard signal processing methods cannot be applied directly.

3. Because of the real-time demands, filtering should be performed on the entire body very efficiently.

A Kalman filter uses prediction of future values to create a filtering scheme that operates in an on-line manner. The technique is common in on-line applications, and was first introduced to the graphics community by Friedman et. al. [9]. Such a filter cannot be directly applied to rotation data without accounting for the non-linearity of the rotation space. To address this problem, Welch and Bishop [28] linearized the orientation space by locally parameterizing the incremental orientation change with Euler angles, based on the result in [1, 6]. Because they were interested only in tracking the head motion, they were less concerned with efficiency than we are and therefore addressed only issues 1 and 2 above. In Section 3 we provide a modified Kalman filter. To achieve real-time performance, we locally parameterize the incremental orientation with rotation vectors instead of the Euler angles used in Welch and Bishop [28].

2.2 Importance Determination

The goal of computer puppetry is to create the movement of a target character based on the performer's movements. If the target character is quite different from the performer, there may not be a direct mapping. Indirect mappings are common in traditional puppetry, for example, a marionette is controlled by strings that pull on its end-effectors. Computer equivalents may create arbitrary mappings from sensor input to character parameters. For example, the Alive system from Protozoa [22] allows arbitrary Scheme functions to be written to perform mapping.

Our interest is in recreating characters with human form, so the target character has equivalent degrees of freedom as the simplified model of human being. In this paper, we consider characters that are articulated figures with identical connectivity, so that it is possible to transfer the captured joint angles directly to the target character. Despite this structural equivalence, the resulting motion will not match the performer's

unless the character has identical size and proportion. There will be some level of mismatching even for characters which have the same size and proportion as the performer, since we simplify the real human by a hierarchy of rigid bodies. One approach to performance animation, described by Molet et al. [18, 19], models the character to be as similar to the performer as possible. Bodenheimer et al. [5] presented how to determine the segment lengths of a character that best fit the captured motion data while discarding outliers in the captured motion data by a robust estimation technique. If the segment proportions of the character are kept the same as those of the performer, a limited motion adaptation, such as keeping the foot position stationary during a contact, can be achieved by scaling the position data according to the size difference. Restricting the proportions of the character precludes the use of stylized cartoon characters, unless we can find similarly proportioned performers.

When the virtual character and performer have different sizes and proportions, not all aspects of the motions can be preserved during mapping. At the lowest level, it is simply not possible to mimic both the locations of the end-effectors and the joint angles. A system must make choices as to which aspects of the motion should be preserved and which should be allowed to change. We call an approach to motion retargetting that makes this choice explicitly an *importance-based* approach.

Non-importance-based approaches make implicit choices as to what should be preserved during retargetting. For example, the most naive implementation of retargetting simply transfers the parameter (joint angles and root position) values from performer to character. Such a scheme implicitly selects the values of the parameters to be important and, therefore, the positions of the end-effectors to be unimportant. This is a poor choice when the character must interact with other objects in the world, such as the floor.

A common approach to motion retargetting matches the end-effector positions of the character to those of the performer. Such an approach has the advantage that it

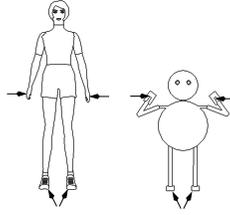


Figure 2: Artifacts of position-based approach

preserves the interactions between the character and its environment. Badler et al. [3] used only the position data of hands and feet to adopt them to a virtual character with an inverse kinematics technique. Residual degrees of freedom are fixed by exploiting bio-mechanical knowledge. Choi et al. [7] adopted the idea of inverse rate control [29] to compute the changes in joint angles corresponding to those in end-effector positions while imitating the captured joint angles by exploiting the kinematic redundancy.

Implicit in end-effector schemes is the notion that end-effector positions are more important than joint angles, that is, joint angles should be changed to achieve end-effector positioning goals. While this prioritization is often preferable to the reverse, it is not without its flaws. Consider the example of Figure 2. In this example, the importance of the foot positions are properly reflected, while that of the hand positions are overstated.

The central observation of an importance-based approach is that what is important can only be determined by the context of the motion. At each instant, a system must somehow select among the many possible things which are important, so it can change the aspects that are not important.

Constraint-based approaches to motion explicitly represent details of the motion that are important as geometric constraints. Gleicher's space-time motion editing [11] and retargetting system [12] proposed the notion of preserving the important qualities of the motion by changing unimportant ones, where the important qualities were defined by constraints. Lee and Shin's hierarchical motion editing [17] provided similar

results using a different underlying implementation. Popovic and Witkin demonstrated results that made the kinetic aspects of the original motion important to preserve [21].

The methods mentioned in the previous paragraph are all off-line in that they examine the entire motion simultaneously in processing. This off-line nature is also implicit in the problem formulation, as well as in the solution method. All of the methods require the constraints to be identified before the motion can be processed. The decisions as to what is important in a motion must be known before processing can occur in these previous constraint-based approaches. This is infeasible in on-line applications. Bindiganavale and Badler [4] introduced a constraint determination scheme to generate constraints automatically. However, their motion adaptation is done in an off-line manner.

For computer puppetry, we must decide what is important in a given motion in an on-line manner. We analyze the importance of each end-effector position based on several factors discussed in Section 4. For example, the proximity of an end-effector position to its surrounding environment can be used as a predictor of its importance. The importance of an end-effector is inversely proportional to its distance to the nearest object in the environment. A key notion of this work is that the power of an importance-based approach, already demonstrated in off-line constraint-based systems, can be brought to the on-line domain of computer puppetry.

2.3 Inverse Kinematics

We employ an inverse kinematics (IK) solver to compute the pose of the target character. IK has become a standard technique in animation systems to control the pose of a character based on the positions of its end-effectors.

IK solvers can be divided into two categories: analytic and numerical solvers. Most industrial robot manipulators are designed to have analytic solutions for efficient and robust control. Paden [20] divided an IK problem into a series of simpler subprob-

lems each of which has closed-form solutions. Korein and Badler [16] showed that the IK problem of a human limb allows an analytic solution, and Tolani and Badler [26] derived their actual solutions. A numerical method relies on an iterative process to obtain a solution. Girard and Maciejewski [10] generated the locomotion of a legged figure using a pseudo inverse of a Jacobian matrix. Based on neurophysiology, Koga et al. [15] produced an experimentally good initial guess for a numerical procedure. Gullapalli et al. [13] reduced the dimensionality of the redundant control system using synergies as a basis control set. Zhao and Badler [30] formulated the IK problem as a constrained non-linear optimization problem. Rose et al. [23] extended this formulation to cover constraints that hold over an interval. To prevent the figure from doing non-natural motions and reduce the redundancy of the IK problem, Badler et al. [3] incorporated biomechanical information.

For computer puppetry, we make a number of demands on IK that required the development of a novel solver. First, we must achieve real-time performance on the entire body of the character. Secondly, we need the solver to provide predictably consistent solutions: small changes to the problems should provide similar answers. Finally, the solver must be able to account for the importances that are determined dynamically in our system.

Our IK solver is discussed in Section 5. To solve an IK problem in real time, we divide it into three subproblems: root position estimation, body posture computation, and limb posture computation. First, the root position of a virtual character is computed to provide a good initial guess for the body posture computation. If needed, we then adopt numerical optimization to refine the body posture, which consists of the root position and the orientations of the pelvis and the upper body. Finally, we use an analytical IK solver to compute the limb postures and blend them with the captured limb postures. Our solution for each of these subproblems is designed to incorporate the importance values of the end-effectors so that it tries to preserve end-effector positions when their

importance values are high, while trying to preserve the captured joint angles of the corresponding limb, otherwise.

3 Motion Filtering

In general, motion capture devices capable of providing real-time performance are particularly susceptible to noise. Magnetic motion capture systems, which are widely used for real-time motion capture, suffer from the interference of low-frequency current-generating devices such as a CRT-type display. Thus, there always exists some level of jitter, that is, rapid random changes in reported positions and orientations that do not correspond to actual movements [8]. Since on-line motion retargetting requires a high quality input motion as the reference of an output motion, filtering is an essential part. In the context of computer puppetry, filtering must be real-time, on-line, and performed on orientations.

For on-line filtering, Kalman filters [2, 9, 28] are often employed because of their capability of prediction and correction, that is, predicting future input data from their history and correcting them by incorporating actual input data. Moreover, Kalman filters can remove the random gaussian noises together with high frequency noises. Because the noise included in motion capture data is not strictly a high frequency, Kalman filters are an effective choice for denoising motion capture data in an on-line manner.

In a standard (extended) Kalman filter, its state would completely describe the position of a sensor and its orientation. However, because of the non-linearity of the orientation space, this scheme can hardly be applied directly to orientation data. Adopting the results in [1, 6], Welch and Bishop [28] parameterized an incremental orientation change with Euler angles which were regarded as a 3-vector to filter. The filtered Euler angles were transformed back to an incremental orientation change in the non-linear space to update the target orientation at each time step. However, the conversion be-

tween an incremental orientation change and its equivalent Euler angles is inefficient. Moreover, recent motion capture devices measure orientations directly in unit quaternions. Therefore, differently from Welch and Bishop, we parameterize incremental orientation changes with rotation vectors.

To facilitate our scheme, we maintain the target orientation \mathbf{q}_e externally to the Kalman filter together with the internal state vector \mathbf{x} . In particular, \mathbf{q}_e is represented by an unit quaternion:

$$\mathbf{q}_e = (w, (x, y, z)),$$

where $w^2 + x^2 + y^2 + z^2 = 1$. The internal state \mathbf{x} consists of the position \mathbf{p} , the rotation vector \mathbf{r} , and their derivatives $\dot{\mathbf{p}}$ and $\dot{\mathbf{r}}$:

$$\mathbf{x} = (\mathbf{p}^T \dot{\mathbf{p}}^T \mathbf{r}^T \dot{\mathbf{r}}^T)^T. \quad (1)$$

Here the rotation vector \mathbf{r} parameterizes the incremental orientation change of the actual sensor input $\mathbf{q}(t)$ at the current frame with respect to the target orientation $\mathbf{q}_e(t - \Delta t)$ at its previous frame. Therefore, we first compute difference between these two orientations, that is, $\mathbf{q}_e^{-1}(t - \Delta t)\mathbf{q}(t)$. Then, we convert this unit quaternion into a rotation vector through the logarithmic map [14] to measure $\mathbf{r}(t)$:

$$\mathbf{r}(t) = \ln(\mathbf{q}_e^{-1}(t - \Delta t)\mathbf{q}(t)). \quad (2)$$

At each filter update step, $\mathbf{r}(t)$ in the state is converted into its incremental orientation change equivalent $e^{\mathbf{r}(t)}$ through the exponential map to update the external target orientation \mathbf{q}_e and then reset to be zero. Therefore, incremental orientations are linearized for our (extended) Kalman filter, centered about zero.

Our dynamic model predicts the current position and the rotation by first-order ap-

proximations. Therefore, the prediction $\hat{\mathbf{x}}^-(t)$ of the state through the *state transition matrix* \mathbf{A} can be described :

$$\hat{\mathbf{x}}^-(t) = \mathbf{A}\hat{\mathbf{x}}(t - \Delta t) = \begin{bmatrix} \mathbf{I}_3 & \Delta t\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \Delta t\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \hat{\mathbf{x}}(t - \Delta t), \quad (3)$$

where \mathbf{I}_3 and $\mathbf{0}_3$ are, respectively, 3×3 identity and zero matrices. Note that we use the minus sign on the upper right corner for the predicted values. Similarly, the *error covariance matrix* $\mathbf{P}(t)$ is predicted:

$$\mathbf{P}^-(t) = \mathbf{A}\mathbf{P}(t - \Delta t)\mathbf{A}^T + \mathbf{Q}. \quad (4)$$

Here, $\mathbf{P}(t) = E \left[(\hat{\mathbf{x}}^-(t) - \mathbf{x}(t)) (\hat{\mathbf{x}}^-(t) - \mathbf{x}(t))^T \right]$, which models estimation uncertainty. The *process noise covariance matrix* \mathbf{Q} characterizes the accuracy of the dynamic model. In our implementation, we simplify \mathbf{Q} as follows:

$$\mathbf{Q} = \begin{bmatrix} q_1\mathbf{I}_3 & q_2\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ q_3\mathbf{I}_3 & q_4\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & q_5\mathbf{I}_3 & q_6\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & q_7\mathbf{I}_3 & q_8\mathbf{I}_3 \end{bmatrix}. \quad (5)$$

When the values of q_i 's are small, the filter tends to suppress the detail of the captured motion. On the other hand, if they are large, it tends to preserve the captured motion. Therefore, q_i 's should be tuned properly with respect to the condition of the capture and the contents of the motion by increasing q_i 's for detailed motion while decreasing them for highly noisy input data.

We sample motion signals at a higher frame rate (~ 120 fps) than that actually

required for animation to avoid the overshooting which occasionally occurs in constant velocity models, especially when the velocity changes suddenly. Our measurement consists of the position of a sensor and its incremental orientation represented by a rotation vector, that is, $\mathbf{z} = (\mathbf{p}^T \mathbf{r}^T)^T$ which can be obtained from of the state vector directly. Therefore, our measurement can be predicted from the predicted state:

$$\hat{\mathbf{z}}(t) = \mathbf{H}\hat{\mathbf{x}}^-(t) = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 \end{bmatrix} \hat{\mathbf{x}}^-(t). \quad (6)$$

Now, we are ready to compute the Kalman gain $\mathbf{K}(t)$ as [27]:

$$\mathbf{K}(t) = \mathbf{P}^-(t)\mathbf{H}^T(\mathbf{H}\mathbf{P}^-(t)\mathbf{H}^T + \mathbf{R})^{-1}, \quad (7)$$

where \mathbf{R} is the measurement noise covariance matrix. That matrix is either given from the device manufacturer or acquired by off-line measurement. In practice, we measure the noise while holding the sensor stationary to compute its noise covariance matrix \mathbf{R} .

The residual between the actual sensor measurement $\mathbf{z}(t)$ and the predicted measurement $\hat{\mathbf{z}}(t)$ from Equation (6) is:

$$\Delta\mathbf{z}(t) = \mathbf{z}(t) - \hat{\mathbf{z}}(t). \quad (8)$$

Then, the predicted state and the error covariance matrix are corrected as follows:

$$\begin{aligned} \hat{\mathbf{x}}(t) &= \hat{\mathbf{x}}^-(t) + \mathbf{K}(t)\Delta\mathbf{z}(t), \text{ and} \\ \mathbf{P}(t) &= (\mathbf{I} - \mathbf{K}(t)\mathbf{H})\mathbf{P}^-(t). \end{aligned} \quad (9)$$

We finish filtering at each frame by updating the external target orientation using the rotation vector $\hat{\mathbf{r}}(t)$. Taking the exponential map of the rotation vector and post-multiplying it with the external target orientation $\hat{\mathbf{q}}_e(t - \Delta t)$ at the previous frame, we

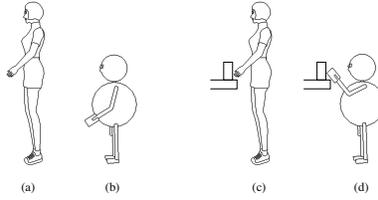


Figure 3: Two different situations

can find the final target orientation $\hat{\mathbf{q}}_e(t)$ at the current frame:

$$\hat{\mathbf{q}}_e(t) = \hat{\mathbf{q}}_e(t - \Delta t)e^{\hat{\mathbf{r}}(t)}. \quad (10)$$

The rotation vector $\hat{\mathbf{r}}(t)$ is reset to zero for filtering at the next frame.

4 Importance Analysis

When the performer and the target character do not have the same size and proportion, not all aspects of the original motion can be preserved. A system must determine which aspects of the motion are important to preserve, so that other less important aspects may be changed to preserve them.

For an articulated figure, differing segment lengths means that both the joint angles and end-effector positions cannot be simultaneously be recreated. There are three obvious choices of motion aspects to preserve:

1. The position of the root of the character.
2. The joint angles.
3. The positions of the end-effectors.

There exist situations under which any of these three might be most important. For example, observe the arm postures in Figure 3. Figure 3(a) shows a captured arm

posture from the performer. Retargetting this motion to a virtual character that does not touch any object, we prefer the posture in the Figure 3(b) that preserves the joint angles. However, the position of a hand needs to be preserved when it touches an object as shown in Figure 3(c) and (d).

Our system must choose which of the three choices above is most important in a dynamic, on-line way. To make this decision, we employ a number of heuristics:

1. The position of the root is most likely *not* important. This heuristic comes from the observation that the choice of making the root is arbitrary: we could have just as easily chosen any point as the root. In fact, preserving the root position may change some important parameters that characterize a posture itself. Because of this, the importance of the root position is downplayed in many approaches that consider importance. Like our solver, described in Section 5, Gleicher's retargetting system [12] uses a heuristic that attempts to satisfy the constraints (generally on the end-effectors) as much as possible by moving the root position.
2. If an end-effector is interacting with another object (such as the floor), then its position is likely to be important. Therefore, proximity to objects in the environment should increase the importance of an end-effector.
3. If an end-effector will be interacting with another object in the near future, then its position is important (as it is likely to be getting ready for the interaction). Therefore, we incorporate prediction of proximity of an end-effector to an object in the measure of its importance.
4. If an end-effector has just finished interacting with another object and is moving away from it, its position may not be as important as its proximity suggests.
5. If the end-effector is not in proximity to another object, it is likely that its position is unimportant.

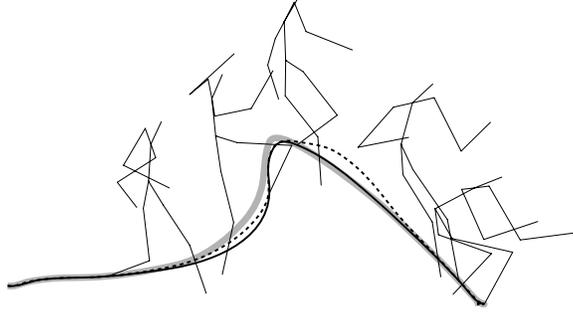


Figure 4: Trajectories of the left foot generated by varying importance measure

In order to measure the interactivity of an end-effector with its environment, we introduce the notion of *importance* of an end-effector, which can be determined by analyzing the posture of the character in relation to the environment. In particular, the distance from the end-effector to objects in the environment is a good measure of interaction possibility. That is, the end-effector is more likely to interact with the environment when it is closer to objects in the environment. Therefore, as the end-effector approaches an object, its importance value should be increased to enforce the geometric constraints created by the object. As the end-effector moves away from the object, the importance value should be continuously decreased to preserve the captured posture of the corresponding limb. Moreover, it is desirable to develop the distance measure to reflect the trajectory of an end-effector and its dynamic nature.

Given end-effector e_i of a character and object o_j in the virtual space which is corresponding to the real object, let $d_{ij}(t)$ be Euclidean distance between them at time t . The new distance function $d_{ij}^+(t)$ is defined as

$$d_{ij}^+(t) = \frac{d_{ij}(t) + d_{ij}(t + \kappa\Delta t)}{2} \quad (11)$$

for some positive κ and Δt . $d_{ij}^+(t)$ represents the average of the current distance and the predicted distance after $\kappa\Delta t$ time. For small Δt , $d_{ij}^+(t)$ can be approximated as

follows:

$$\begin{aligned} d_{ij}^+(t) &\approx \frac{d_{ij}(t) + (d_{ij}(t) + \kappa\Delta t \dot{d}_{ij}(t))}{2} \\ &= d_{ij}(t) + \frac{\kappa\Delta t}{2} \dot{d}_{ij}(t) = d_{ij}(t) + \lambda \dot{d}_{ij}(t), \end{aligned} \quad (12)$$

where $\dot{d}_{ij}(t)$ is the first derivative of $d_{ij}(t)$. $d_{ij}^+(t)$ reflects both the distance at t from e_i to o_j and its changing rate $\dot{d}_{ij}(t)$. By varying λ we can control the degree of prediction for $d_{ij}^+(t)$.

For an example, Figure 4 exhibits a jumping motion adapted with $\lambda = 0$ and $\lambda = 0.15$. The legs of the character are shorter than the performer's. For $\lambda = 0$, the left foot trajectory of the character (dashed line) agrees with that of the performer (thicker line) only near the floor. For $\lambda = 0.15$, the former follows the latter while approaching down to the floor (solid line). The foot is moving off the captured trajectory to preserve the captured joint angles, either near the peak ($\lambda = 0$) or approaching to the peak ($\lambda = 0.15$).

Let D_{ij} denote the maximum distance within which e_i is influenced by o_j . Then, the normalized distance \bar{d}_{ij} is defined as

$$\bar{d}_{ij} = \frac{d_{ij}^+}{D_{ij}}. \quad (13)$$

An animator assigns D_{ij} for the pair of end-effector e_i and object o_j in the environment in accordance with a given animation context. A wider range of D_{ij} shows a sensitive interaction of end-effector e_i with object o_j . On the other hand, a narrower range exhibits that e_i moves independently of o_j unless e_i is close to o_j .

The importance is zero when the normalized distance \bar{d}_{ij} is greater than or equal to one, that is, e_i is out of the influence of o_j . As the distance decreases to zero, the importance increases to one. Thus, the importance function p of the normalized distance \bar{d}_{ij} can be designed with the condition of $p(1) = 0$ and $p(0) = 1$. In addition,

we set its derivatives there to be zero, that is, $p'(0) = 0$ and $p'(1) = 0$, to reduce the rate of change of the function p at both extreme points. Thus, the importance of e_i with respect to o_j is represented by the cubic polynomial function p satisfying those conditions. That is,

$$p(\bar{d}_{ij}) = \begin{cases} 2\bar{d}_{ij}^3 - 3\bar{d}_{ij}^2 + 1, & \text{if } \bar{d}_{ij} < 1, \\ 0, & \text{otherwise.} \end{cases} \quad (14)$$

The importance value w_i of end-effector e_i over all external objects can be defined as the maximum of them:

$$w_i = \max_j(p(\bar{d}_{ij})). \quad (15)$$

It requires much time to compute the distance \bar{d}_{ij} from each end-effector e_i of a virtual character to every object o_j in the environment, especially for a complex surrounding environment. To achieve a real-time performance, we need to minimize the number of possible objects that interact with each end-effector in accordance with an animation context. An object that is hardly touched during the animation may be eliminated in importance value computation. Moreover, objects may also be described approximately with simpler geometry for easy distance computation.

5 Real-time Inverse Kinematics Solver

For computer puppetry, we must position the character such that the important aspects of a captured motion are preserved while providing real-time performance. For our application, this demands computing the character's posture 30 times per second. Therefore, we need an IK solver that not only can incorporate the importance measures of the previous section, but also has real-time performance even in the worst case.

As discussed in Section 2.3, previous IK solution methods do not meet the demands of computer puppetry. Analytic methods provide guaranteed performance but cannot incorporate importance measures required for retargetting. Numerical solvers can include the importance metrics, but they hardly guarantee real-time performance. To meet these two conflicting demands, we have developed a hybrid solver.

In this section, we present a fast IK algorithm which is specialized for human-like articulated characters. We divide the IK process into three sub-problems: root position estimation, body posture computation, and limb-posture computation. For each step, we give a method that is specialized to achieve high-performance. This leads us to employ inexpensive, closed-form solutions if applicable, and reserve numerical optimization for the case in which it is absolutely required.

5.1 Root Position Estimation

In order to position the end-effectors of a character, an IK solver may change the root position of the character or adjust its joint angles. As mentioned in Section 4, the root of the character has been arbitrarily chosen as the character’s root, which is rarely the most important aspect to preserve. Therefore, our solver first attempts to make the character satisfy the constraints as much as possible by moving the root position. This strategy was demonstrated for retargetting by Gleicher [12].

Beginning with the positional offset has an important advantage: unlike angular changes that cause non-linear equations to compute, positional offset computation is trivial and therefore efficient. Let \mathbf{p}_i^e represent the position of the i -th end-effector when the character is posed with the captured joint angles, and \mathbf{p}_i^g denote the goal position for that end-effector. The displacement vector $\mathbf{d}_i = \mathbf{p}_i^g - \mathbf{p}_i^e$ measures how much the solver must move an end-effector to reach its goal. If there were only one end-effector with a specified goal position, this constraint could be met by simply moving the character’s root position by the displacement vector, where the joint angles would

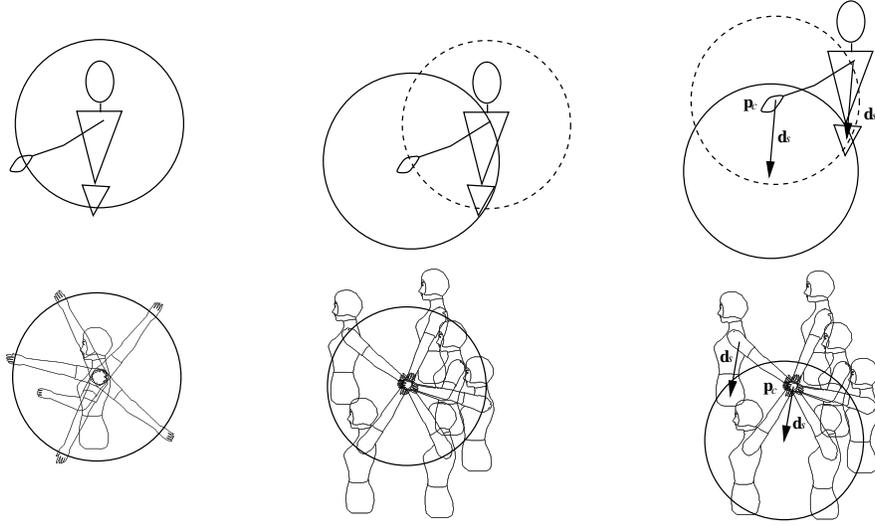


Figure 5: Range 3D disks: range of hand, shoulder, and root position

not need to be changed.

In the event that multiple end-effectors are to be positioned, we compute the weighted average of the displacements to find an initial offset \mathbf{d} as follows:

$$\mathbf{d} = \frac{\sum_i^n w_i \mathbf{d}_i}{\sum_i^n w_i}, \quad (16)$$

where w_i is the importance of the i -th end-effector. In the (unlikely) event that all end-effectors require the same displacement, this displacement will solve all of the constraints. More likely, the joint angles will need to be adjusted so that all of the end-effector goals can be met.

While the weighted averaging attempts to position the root to meet all of the goals simultaneously, it does not necessarily guarantee that all goals can be met. Once the root position is fixed, the character can meet its goals by straightening its joints. Therefore, the root position must be chosen such that all end-effector goals are “reachable,” that is, close enough that straightening limbs will be sufficient. We refine our root posi-

tion estimate such that it guarantees reachability if possible. We relocate the root such that it is within the reachability limits to the goals while being as close to the initial estimate as possible.

As shown in the left of Figure 5, the reachable space of the hand can be represented as the 3D disk centered at the shoulder, and its radius is the length of the arm. Here, a 3D disk consists of a sphere and the set of all points bounded by it. The middle of Figure 5 shows that the same 3D disk centered at the goal position represents the range of the shoulder joint position. Finally, with the orientations of the pelvis and the waist fixed as in the captured posture, we compute the range of the root position as illustrated on the right of Figure 5. Let \mathbf{d}_s denote the vector from the shoulder to the root position. The translation of the 3D disk at the goal position \mathbf{p}_c by the vector \mathbf{d}_s yields the 3D disk that gives the range of the root position. If the root is in this 3D disk, the character can reach the goal position by stretching the limb only.

When the importance value of an end-effector is low, the root position does not need to be modified to make this end-effector reachable at its goal. Therefore, the range corresponding to this end-effector may be larger than the actual reachable range. To avoid an unnecessary offset of the root position, we enlarge the size of the 3D disk, so that its size is inversely proportional to the importance value. The increased radius r_i corresponding to the i -th limb is given as follows:

$$r_i(l_i, w_i) = \frac{l_i}{w_i}, \quad (17)$$

where l_i is the length of the i -th limb and w_i is its importance value.

Since the virtual character has four end-effectors, we have four 3D disks. The common intersection of these 3D disks is the range of the root position that makes all of the end-effectors reachable to their goal positions. As an initial guess for the root position, we choose the closest point from the offset root position to this intersection to preserve the posture of the performer as much as possible. Thus, the root position estimation

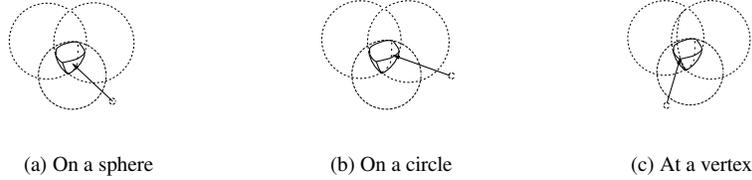


Figure 6: Closest points

is formulated as the problem of finding the closest point from a given position to the common intersection of four 3D disks.

The intersection of 3D disks consists of four surface elements as shown in Figure 6: spherical regions, circular edges, and vertices. A spherical region is a part of a sphere bounded by a sequence of spherical arcs. A circular edge is a part of a circle that is the intersection of two spheres. A vertex is determined by the common intersection of three spheres.

There are two cases depending on the offset root position with respect to the intersection. If this point is contained in the interior of the intersection, then the point itself is the closest point to the intersection. Suppose that it is not contained in the interior. Then the closest point must lie on the boundary of the intersection. Therefore, we may enumerate all possible surface elements due to the intersection of the four spheres corresponding to the bounding surfaces of the disks, respectively.

Three spheres determine at most two vertices. Since there are four ways of choosing a triple out of four spheres, we have a maximum of eight vertices. Every pair of vertices can possibly admit a spherical edge, and thus we have at most 24 edges. However, these are completely included in a maximum of six circles. Moreover, each spherical face is completely contained in one of four spheres. Instead of enumerating all surfaces elements, we equivalently check those spheres, circles and vertices.

We first compute the closest point to each sphere from the offset root position. Among these points, if any, we choose the point that is contained in the intersection and

the closest to the root position. If such a point does not exist, then we compute the set of points, each of them is the closest from the root position to each circle. Out of them, we choose the one that is closest to the root position and in the intersection. Suppose that there does not exist such a point. Then one of vertices may be the solution. We choose the one closest to the root position among those contained in the intersection. For more details in computing the initial root position, refer to the Appendix. If there does not exist a common intersection of the disks, we discard the spheres which do not intersect the one whose corresponding end-effector has the largest importance value and repeat this process for the remaining disks.

5.2 Body Posture Computation

If the initial root position estimate does not allow all limbs to be reachable to the goal positions, we need to adjust the body posture consisting of the root position, the orientation of the pelvis, and that of the upper body. Since those segments are tightly coupled, a numerical method is adopted to find their configurations. Numerical methods hardly guarantee a real-time response for computing the inverse kinematics of an entire human figure, while it is practical to solve only a small part of the IK problem numerically, and to employ analytic methods for the rest of the task. Such a hybrid solver was demonstrated in [17].

We formulate a restricted version of the IK problem for determining the posture of the body posture separately from the problem of computing the posture of the limbs. The body posture of a character can be written as $\mathbf{v} = (\mathbf{p}_0, \mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_n)$, where \mathbf{p}_0 and \mathbf{q}_0 are the position and the orientation of the root, respectively. $\mathbf{q}_j, 1 \leq j \leq n$, are the orientations of body segments such as the waist and the upper body. When the character has a rigid torso, \mathbf{v} is simply reduced to $(\mathbf{p}_0, \mathbf{q}_0, \mathbf{q}_1)$, since $n = 1$.

The objective function consists of two terms:

$$E = E_g + \alpha E_p, \quad (18)$$

where the first term E_g is for making the end-effectors reachable to their goals and the last term E_p is to preserve the captured posture. We will explain those two terms in detail.

E_g is the sum of E_i 's, each of which is a function of the distance from the i -th end-effector e_i to its goal position. Provided with the shoulder (or the coxa) position \mathbf{p}_i^s of the i -th limb and its goal position \mathbf{p}_i^g , E_i is given as follows:

$$E_i = \begin{cases} 0, & \text{if } \|\mathbf{p}_i^s - \mathbf{p}_i^g\| < l_i, \\ (\|\mathbf{p}_i^s - \mathbf{p}_i^g\| - l_i)^2, & \text{otherwise,} \end{cases} \quad (19)$$

where l_i is the length of the i -th limb when it is maximally stretched. E_i is zero when the end-effector e_i is able to reach its goal position. For this case, we prefer to lengthen or shorten the corresponding limb than to adjust the body posture. Recall that an end-effector of a low importance value has no need to preserve its captured position. Thus, to relax the constraint on this end-effector we enlarge the range of the shoulder. By substituting the length l_i of each limb with the new radius $r_i = \frac{l_i}{w_i}$ as mentioned in Section 6.1, we have

$$E_i = \begin{cases} 0 & , \text{ if } \|\mathbf{p}_i^s - \mathbf{p}_i^g\| < r_i, \\ (\|\mathbf{p}_i^s - \mathbf{p}_i^g\| - r_i)^2 & , \text{ otherwise.} \end{cases}$$

Note that with the importance value w_i of one, E_i plays a role of pulling the end-effector to reach the goal position exactly. On the other hand, as importance value w_i approaches zero, the i -th end-effector keeps the original posture by preserving the joint angles.

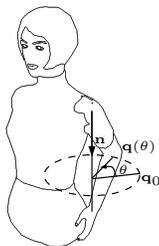


Figure 7: Residual degree of freedom of shoulder

Letting \mathbf{q}_j^* and \mathbf{p}_0^* be the captured orientation of the j -th segment and the estimated position of the root, respectively, E_p is a weighted sum of the squared geodesic distances between \mathbf{q}_j and \mathbf{q}_j^* for all $0 \leq j \leq n$, and the squared distance between \mathbf{p}_0 and \mathbf{p}_0^* :

$$E_p = \sum_{j=0}^n \beta_j \|\ln(\mathbf{q}_j^{-1} \mathbf{q}_j^*)\|^2 + \gamma \|\mathbf{p}_0 - \mathbf{p}_0^*\|^2. \quad (20)$$

Minimizing E_p preserves the captured motion as much as possible. We find the optimal solution that minimizes the objective function by employing the conjugate gradient method. Here, we use the captured joint angles and the root position computed in Section 5.1 as the initial guess for our optimization.

5.3 Limb Postures Computation

Given the position of a shoulder and that of the goal together with a hand orientation, we present how our IK solver computes the configuration of an arm. The configuration of a leg can similarly be computed from the hip position and foot position and orientation. As pointed out by Tolani et al. [25] and Lee et al. [17], the angle between the brachium of the arm and its forearm can be computed uniquely from the distance between the shoulder and the goal. We adjust the shoulder joint to locate the wrist at the goal position. Even with the wrist position fixed at the goal position, the shoulder joint

still has one residual degree of freedom that rotates the elbow about the axis passing through the shoulder and the wrist. Korein et al. [16] have parameterized that degree of freedom by the swivel angle θ . As illustrated in Figure 7, the elbow traces a circle called the *elbow circle* as θ varies. Once θ is chosen, the joint angle of the wrist is determined uniquely by preserving the orientation of the hand.

This swivel angle θ can be described with a unit quaternion formulation. The unit quaternion $\mathbf{q}(\theta)$ representing the rotation by θ about the axis \mathbf{n} is $e^{\frac{\theta\mathbf{n}}{2}}$ for $-\pi < \theta \leq \pi$, where θ is measured from an arbitrarily chosen reference point on the circle. Denoting this point by a unit quaternion \mathbf{q}_0 , we have

$$\mathbf{q}(\theta) = e^{\frac{\theta\mathbf{n}}{2}} \mathbf{q}_0. \quad (21)$$

Unlike the original version of Lee et al. [17] which determines θ by a numerical optimization, we solve for θ analytically so that the arm posture deviates as small as possible from the captured posture. Thus, we choose θ that minimizes the geodesic distance ϕ in $[0, 2\pi)$ from the captured shoulder joint orientation \mathbf{q}^* to $\mathbf{q}(\theta)$. Note that both \mathbf{q}^* and $-\mathbf{q}^*$ represent the same orientation, due to the antipodal equivalence of the unit quaternion space. Therefore,

$$\phi(\theta) = \min(\arccos(\mathbf{q}^* \cdot \mathbf{q}(\theta)), \arccos(-\mathbf{q}^* \cdot \mathbf{q}(\theta))) = (\arccos(|\mathbf{q}^* \cdot \mathbf{q}(\theta)|)). \quad (22)$$

$\phi(\theta)$ is minimized when $|\mathbf{q}^* \cdot \mathbf{q}(\theta)|$ is maximized. By definition, $e^{\frac{\theta\mathbf{n}}{2}} = (\cos \frac{\theta}{2}, \mathbf{n} \sin \frac{\theta}{2})$. Let $\mathbf{q}^* = (w^*, \mathbf{v}^*)$ and $\mathbf{q}_0 = (w_0, \mathbf{v}_0)$. Then, the absolute value of their inner product

is given as follows:

$$\begin{aligned}
|\mathbf{q}^* \cdot \mathbf{q}(\theta)| &= \left| (w^*, \mathbf{v}^*) \cdot \left\{ \left(\cos \frac{\theta}{2}, \mathbf{n} \sin \frac{\theta}{2} \right) (w_0, \mathbf{v}_0) \right\} \right| \\
&= \left| a \cos \frac{\theta}{2} + b \sin \frac{\theta}{2} \right| \\
&= \sqrt{a^2 + b^2} \left| \sin \left(\frac{\theta}{2} + \alpha \right) \right|, \tag{23}
\end{aligned}$$

where

$$\begin{aligned}
\alpha &= \tan^{-1} \frac{a}{b} \\
a &= w^* w_0 + \mathbf{v}^* \cdot \mathbf{v}_0 \\
b &= w_0 \mathbf{n} \cdot \mathbf{v}^* - w^* \mathbf{n} \cdot \mathbf{v}_0 + \mathbf{v}^* \cdot (\mathbf{n} \times \mathbf{v}_0).
\end{aligned}$$

The absolute value of the sine function has the maximum at $\frac{\pi}{2}$ and $-\frac{\pi}{2}$. Thus, $|\mathbf{q}^* \cdot \mathbf{q}(\theta)|$ is maximized either at $\mathbf{q}(-2\alpha + \pi)$ or at $\mathbf{q}(-2\alpha - \pi)$. Since the distance of two points in the unit quaternion space is inversely proportion to their dot product, $\mathbf{q}(-2\alpha + \pi)$ is the closest from \mathbf{q}^* if $\mathbf{q}^* \cdot \mathbf{q}(-2\alpha + \pi) > \mathbf{q}^* \cdot \mathbf{q}(-2\alpha - \pi)$; otherwise, $\mathbf{q}(-2\alpha - \pi)$ is the closest.

Now, with both captured and computed limb postures available, we blend them together to obtain a realistic motion. For this purpose, we perform spherical linear interpolation between each captured joint orientation of a limb with its corresponding IK solution. Let q_{ik} and q_{ik}^* be the orientation of the k -th joint in the i -th limb obtained from the IK solver and that from the captured posture. Then the blended joint angle q'_{ik} can be described by spherical linear interpolation as follows:

$$\begin{aligned}
\mathbf{q}'_{ik} &= \text{slerp}(q_{ik}^*, q_{ik}, w_i) \\
&= e^{w_i \ln(\mathbf{q}_{ik} \mathbf{q}_{ik}^{*-1})} \mathbf{q}_{ik}^*, \tag{24}
\end{aligned}$$

where w_i is the importance value of the i -th limb. That is, the limb with a high importance value can preserve the end-effector position, and that with a low importance value can preserve the captured joint angles.

The non-penetration condition may be violated since the posture is blended regardless of the constraints. Thus the blended posture has to be adjusted explicitly to prevent unwanted penetration. Provided with the predefined external objects for each end-effector, this violation can be detected easily. Before penetrating an object, the end-effector touches the boundary of the object. Thus, the preferable position of the end-effector is the intersection point of the object boundary and the ray from the shoulder to the end-effector during penetration. This position moves continuously on the object in accordance with the end-effector movement. The penetration problem can be effectively eliminated by adjusting the limb posture using the IK solver for limbs.

6 Analysis of Temporal Constraints

In retargetting motions, we must preserve important temporal aspects of the motion along with spatial aspects. Gleicher [12] emphasizes the importance of avoiding the introduction of high-frequencies during adaptation. Both this work and the work of Lee and Shin [17] provide approaches for avoiding the addition of discontinuities during adaptation. Unfortunately, both schemes rely on examining durations of motions and therefore can only be applied in off-line applications. In this section, we show that the approach presented in this paper does not introduce unwanted discontinuities into the resulting motion.

To begin, we must assume that the initial motion is free of unwanted discontinuities. This assumption is not restrictive because the movement of the performer is continuous. Discontinuities may be introduced by noise in the capture process, but these are generally removed by the filtering process described in Section 3. The continuity of the initial motion applies to both the captured joint angles and end-effector positions.

Given continuous paths for the end-effectors, our IK solver will provide continuous trajectories for the parameters. Achieving this requires the solver to make consistent changes. That is, similar inputs to the solver must provide similar outputs. To guarantee this consistency, our IK solver tries to find the solution in an on-line manner so that it is close to the filtered input posture, while satisfying the kinematic constraints.

For our IK solver, the only kinematic constraints are the positions of end-effectors. These constraints are specified at every frame as temporal constraints. As an end-effector is approaching an object in the environment, its distance to the object is monotonically decreasing. Similarly, the distance is monotonically increasing as the end-effector is departing from the object. When the end-effector touches (or passes by) the object, the monotonicity changes but the distance function is still continuous at that instance.

For any continuous distance function, our importance function gives continuous importance values as described in Section 4. In other words, the importance values are consistently changed to reflect the temporal proximity of end-effectors to the environment. Therefore, the importance values have inter-frame coherence. Since our IK solver utilizes as input the reference motion data and the importance values, we can exclude unexpected motion artifacts such as unwanted jerkiness. That is, enforced to minimize the change from the reference motion, our IK solver tries to find an intended motion. Moreover, guided by the importance values for interaction with the environment, it also predicts the future temporal constraints and continuously pays attention to them for motion coherence.

7 Experimental Results

For puppetry performance we use a MotionStar Wireless motion capture device from Ascension Tech, Inc. with 14 sensors and two extended range transmitters. Each of sensors detects the magnetic field emitted by a transmitter to report its position and



(a) Pang-Pang

(b) Aliang

Figure 8: Virtual characters on air controlled by our prototype system

orientation up to 144 times per second.

Our prototype system has been deployed for production and used successfully to create a virtual character for a children’s television program as well as a virtual news reporter. Both have been shown on Korean national television, called KBS. The frog-like creature shown in Figure 8(a) (“Pang-Pang”) who regularly appears in a daily TV show for children to demonstrate his comic performance. Thanks to the capability of our system for synthesizing realistic motion in real time, Pang-Pang and a real actor can interact with each other. Figure 8(b) shows a virtual character (“Aliang”) who has performed the role of a news reporter for the election of Korea National Assembly. Even in a time-critical situation such as reporting interim election results, Aliang can accomplish his role successfully.

The skeleton used in our system has 43 degrees of freedom including 11 revolute joints of 3 degrees of freedom, 4 hinges on elbows and knees, and the position of the root and its orientation. The floor is modeled as a plane for all of the uses of our system to date.

To test our system’s performance, we created two puppets specifically designed to provide challenging retargetting problems. The character named *long tall Sally* has

Table 1: The number of iterations in numerical solver with and without root position estimation

motion	#frames	the number of iterations			
		without		with	
		Blubby	Sally	Blubby	Sally
Walk	39	47	0	0	0
Throw	157	244	0	0	0
Jump	88	111	0	0	0
Handstand	211	266	38	0	0
Dance	591	1253	0	1	0
Total (61 Clips)	9692	15634	429	8	0

long arms and legs, while a ball-shaped man called *Blubby* with extremely short legs. To perform experiments, 61 prerecorded motion clips were used as the input for motion retargetting.

Table 1 shows the number of iterations in numerical optimization with and without initial root position estimation. Statistics for five selected motion clips are given in the first five rows of the table. The total figures for 61 clips are shown in the last row. Since Sally has long arms and legs, she can reach the captured end-effector positions without moving its root position. Thus, the number of iterations for Sally is small even without initial root position estimation. However, with estimated initial root positions, the number of iterations decreases to zero for our test motion clips. The effect of initial root position estimation is more apparent for Blubby with short legs. In most cases, our estimation algorithm finds the root position that makes the end-effectors reachable to their goal positions without any help of the numerical solver given in Section 5.2.

Table 2 gives an overall performance of our on-line motion retargetting algorithm excluding rendering time. Timing information was obtained on a SGI Indigo2 workstation with an R10000 195 MHz processor and 128 Mbytes memory. The execution time for each example mainly depends on the number of iterations in numeric optimization. The tables show real-time performance for each examples.

In Figure 13, a captured walking motion is applied to a character with various methods. The upper images of Figure 13 reveal artifacts due to the geometric inconsistency

Table 2: Elapsed time

motion	#frames	Blubby		Sally	
		elapsed time (msec)	per frame (msec)	elapsed time (msec)	per frame (msec)
Walk	39	203	5.2	206	5.3
Throw	157	864	5.5	876	5.6
Jump	88	466	5.3	474	5.4
Handstand	211	1135	5.4	1139	5.4
Dance	591	3188	5.4	3201	5.4
Total (61 Clips)	9692	52543	5.4	52732	5.4

between the performer and the puppet. Since the positions of the feet are not incorporated into the motion retargetting, the supporting foot is sliding. In contrast, the middle motion preserves the positions well. However, the motions of the arms look unrealistic, since the joint angles of the arms are over-adjusted to preserve the positions of the hands. The bottom figure is generated by our motion retargetting. The supporting foot is fixed at the proper position without sliding, and the joint angles of the arms are preserved as the original ones.

With conventional approaches based on joint angle preservation, there would also exist foot-sliding artifacts when the character has longer limbs, as given in the top of Figure 14. The middle image exhibits unintended bending of legs due to position preservation and an ill-selected initial root position. By assigning low importance values to the hands and offsetting the root position, we have a better result in which the legs are not bent as shown in the bottom figure. Figure 15 gives images of more examples. In Figure 16, we present the motions including crawling in which both hands and feet contact the floor and picking up a box which exhibits interaction between the hands and a movable object.

8 Conclusions & Future Work

We have presented a new approach for on-line motion retargetting that transforms motions of a performer to a virtual character of a different size. Introducing the notion of the importance of an end-effector, we have been able to generate realistic motion for a character in real time while preserving the characteristics of captured motions as much as possible. KBS (Korean Broadcasting System), the largest public television broadcasting company in Korea, has been adopting our on-line motion retargetting algorithm to control the virtual character, Pang Pang in a daily TV show for children. This show has become one of the favorites among children partly due to Pang Pang's successful performance. KBS also successfully showed the performance of a virtual reporter, Aliang for the real election using this algorithm.

Our inverse kinematics solver is specialized for human-like characters to insure real-time performance, although it can be easily adapted to other types of creatures with limbs. The notion of importance gives reasonable look-ahead capability useful for avoiding jerkiness in motion. However, unlike full-scale space-time optimization [12], our approach has no look-back capability due to the on-line nature of computer puppetry and allows only a limited repertoire of constraints.

Our approach addresses only the interaction between the end-effectors of a character and objects in the environment. However, we may also be interaction among the segments of a character. For an example, due to the geometric difference of the character from a performer, its end-effectors may penetrate its body, and also its hands are hard to touch each other without interpenetration for clapping. Thus, to resolve these artifacts, the IK solver should be enhanced to efficiently handle self-interactions.

We focus on handling only the geometric discrepancy between a performer and a puppet. To generate more realistic motions, retargetting should also incorporate the characteristics of the puppet. Anthropomorphized animals such as cartoon-like birds and monkeys have their unique characteristics of motions. Those motions can hardly

be captured directly from a human performer, and thus give an additional barrier to overcome.

Acknowledgements

This work was supported in part by the NRL (National Research Laboratory) program of KISTEP (Korea Institute of Science & Technology Evaluation and Planning).

References

- [1] Ali Azarbayejani and Alex P. Pentland. Recursive estimation of motion structure, and focal length. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(6):562 – 575, 1995.
- [2] Ronald Azuma and Gary Bishop. Improving static and dynamic registration in an optical see-through hmd. In *Proceedings of SIGGRAPH 94*, pages 197–204, 1994.
- [3] N. Badler, M. J. Hollick, and J. P. Granieri. Real-time control of a virtual human using minimal sensors. *PRESENCE*, 2(1):82–86, 1993.
- [4] Rama Bindiganavale and Normal I. Badler. Motion abstraction and mapping with spatial constraints. In *Proceedings of International Workshop, CAPTECH'98*, pages 70–82, 1998.
- [5] Boddy Bodenheimer, Charles Rose, Seth Rosenthal, and John Pella. The process of motion capture: Dealing with the data. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '97*, 1997.

- [6] Ted J. Broida and Rama Chellappa. Estimation of object motion parameters from noisy images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):90–99, 1986.
- [7] Kwang-Jin Choi and Hyeong-Seok Ko. On-line motion retargetting. *Journal of Visualization and Computer Animation*, 11:223–243, 2000.
- [8] Ascension Technology Corporation. *Motion Star Installation and Operation Guide*. Ascension Technology Corporation, 1996.
- [9] Martin Friedmann, Thad Starner, and Alex Pentland. Synchronization in virtual realities. *PRESENCE*, 1(1):139–144, 1991.
- [10] Michael Girard and Anthony A. Maciejewski. Computational modeling for the computer animation of legged figures. In *Proceedings of SIGGRAPH 85*, pages 263–270, 1985.
- [11] Michael Gleicher. Motion editing with spacetime constraints. In *Proceedings of 1997 Symposium on Interactive 3D Graphics*, pages 139–148, 1997.
- [12] Michael Gleicher. Retargeting motion to new characters. In *Proceedings of SIGGRAPH 98*, pages 33–42, 1998.
- [13] Vijaykumar Gullapalli, Jack J. Gelfand, Stephen H. Lane, and Wade W. Wilson. Synergy-based learning of hybrid position/force control for redundant manipulators. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, 1996.
- [14] Myoung-Jun Kim, Sung Yong Shin, and Myung-Soo Kim. A general construction scheme for unit quaternion curves with simple high order derivatives. In *Proceedings of SIGGRAPH 95*, pages 369–376, 1995.
- [15] Y. Koga, K. Kondo, J. Kuffer, and J. Latombe. Planning motions with intentions. In *Proceedings of SIGGRAPH 94*, pages 395–408, 1994.

- [16] J. U. Korein and N. I. Badler. Techniques for generating the goal-directed motion of articulated structures. *IEEE Computer Graphics & Application*, 2:71–81, 1982.
- [17] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of SIGGRAPH 99*, pages 39–48, 1999.
- [18] T. Molet, R. Boulic, and D. Thalmann. A real-time anatomical converter for human motion capture. In *Proceedings of 7th Eurographics Workshop on Animation and Simulation*, 1996.
- [19] T. Molet, R. Boulic, and D. Thalmann. Human motion capture driven by orientation measurements. *PRESENCE*, 8(2):187–203, 1999.
- [20] B. Paden. *Kinematics and Control Robot Manipulators*. PhD thesis, University of California, Berkeley, 1986.
- [21] Zoran Popovic and Andrew Witkin. Physically based motion transformation. In *Proceedings of SIGGRAPH 99*, pages 11–20, 1999.
- [22] Protozoa. Technology information. http://www.protozoa.com/Page_2/info_index.html, 1999.
- [23] Charles F. Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of SIGGRAPH 96*, pages 147–154, 1996.
- [24] D.J. Sturman. Computer puppetry. *IEEE Computer Graphics & Applications*, 18(1):38–45, 1998.
- [25] D. Tolani and N. I. Badler. Real-time inverse kinematics of the human arm. *PRES-ENCE*, 5(4):393–401, 1996.

- [26] D. Tolani, A. Goswami, and N. Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models*, 62(5), 2000.
- [27] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical Report TR95-041, University of North Carolina at Chapel Hill, Department of Computer Science, 1995.
- [28] Greg Welch and Gary Bishop. Scaat: Incremental tracking with incomplete information. In *Proceedings of SIGGRAPH 97*, pages 333–344, 1997.
- [29] D. J. Whitney. Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man-Machine System*, pages 47–53, 1969.
- [30] J. Zhao and N. I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics*, 13(4):313–336, 1994.

A Finding the Closest Point on the Intersection of Spheres

As given in Section 6, there are three types of surface elements: spheres, circles, and vertices. We describe how we find the closest point on each type of element to a given point \mathbf{p} . It is trivial to find the closest point on a sphere to the given point. Therefore, we proceed directly to the other cases.

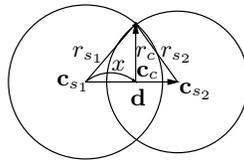


Figure 9: Intersection of two spheres

Now, consider the closest point on a circle to \mathbf{p} . We start with how to construct the circle C , which is the common intersection of the two spheres S_1 and S_2 . The radius

r_c of C can be computed with Pythagorean theorem. Let \mathbf{c}_{s_i} and r_{s_i} for $i = 1, 2, 3$ be the center of the sphere S_i and its radius, respectively. The radius r_c of C satisfies the following equations:

$$r_c^2 + x^2 = r_{s_1}^2, \text{ and} \quad (25)$$

$$r_c^2 + (\|\mathbf{d}\| - x)^2 = r_{s_2}^2, \quad (26)$$

where x is the distance between the center \mathbf{c}_c of C and that of S_1 , and \mathbf{d} is the vector from s_1 to s_2 . Solving those equations, we get

$$r_c^2 = r_{s_1}^2 - \frac{(r_{s_1}^2 - r_{s_2}^2 + \|\mathbf{d}\|^2)^2}{4\|\mathbf{d}\|^2}. \quad (27)$$

Here S_1 and S_2 intersect unless r_c^2 is negative. From Equations (25) and (26),

$$x = \frac{r_{s_1}^2 - r_{s_2}^2 + \|\mathbf{d}\|^2}{2\|\mathbf{d}\|}. \quad (28)$$

Thus,

$$\mathbf{c}_c = \frac{r_{s_1}^2 - r_{s_2}^2 + \|\mathbf{d}\|^2}{2\|\mathbf{d}\|} \cdot \frac{\mathbf{d}}{\|\mathbf{d}\|} + \mathbf{c}_{s_1}. \quad (29)$$

Let \mathbf{n} be the normal vector of the plane where the circle lies. Then,

$$\mathbf{n} = \frac{\mathbf{d}}{\|\mathbf{d}\|}. \quad (30)$$

We are ready to find the closest point on the circle C to the given point \mathbf{p} . Let \mathbf{h} be the projection of the vector $\mathbf{c}_c - \mathbf{p}$ onto the normal vector \mathbf{n} of the plane, that is, $\mathbf{h} = [\mathbf{n} \cdot (\mathbf{c}_c - \mathbf{p})]\mathbf{n}$. Then, the closest point \mathbf{p}_c on C to \mathbf{p} is

$$\mathbf{p}_c = \mathbf{c}_c + \frac{\hat{\mathbf{p}} - \mathbf{c}_c}{\|\hat{\mathbf{p}} - \mathbf{c}_c\|} r_c. \quad (31)$$

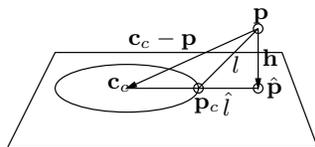


Figure 10: Closest point from a point to a circle

where $\hat{\mathbf{p}} = \mathbf{p} + \mathbf{h}$, that is, $\hat{\mathbf{p}}$ is the projection of \mathbf{p} onto the plane containing C . As shown in Figure 10, the distance l from \mathbf{p} to \mathbf{p}_c is $\sqrt{\|\mathbf{h}\|^2 + \hat{l}^2}$, where \hat{l} is the distance from $\hat{\mathbf{p}}$ to \mathbf{p}_c , that is, $\hat{l} = \|\hat{\mathbf{p}} - \mathbf{c}_c\| - r_c$.

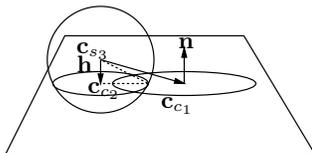


Figure 11: Intersection of a sphere and a plane

Finally, we show how to find the closest among vertices, if any, to the given point \mathbf{p} . Given those vertices, it is trivial to find the closest. Thus, we focus on explaining how to compute the vertices lying at the corners of the common intersection of three spheres, S_1 , S_2 and S_3 . We first calculate the intersection circle C_1 of two spheres S_1 and S_2 . Cutting the sphere S_3 with the plane containing C_1 , we have the circle C_2 . Provided with the center point \mathbf{c}_{c_1} of C_1 and the normal vector \mathbf{n} of the plain containing the circle C_1 , the center point \mathbf{c}_{c_2} of C_2 is the projection of the center point \mathbf{c}_{s_3} of the sphere S_3 onto the plane. Thus,

$$\mathbf{c}_{c_2} = \mathbf{c}_{s_3} + \mathbf{h}, \quad (32)$$

where \mathbf{h} is the vector from \mathbf{c}_{s_3} to \mathbf{c}_{c_2} on the plane, that is, $\mathbf{h} = [\mathbf{n} \cdot (\mathbf{c}_{c_1} - \mathbf{c}_{s_3})] \mathbf{n}$. The

radius r_{c_2} of C_2 is given as follows:

$$r_{c_2}^2 = r_{s_3}^2 - \|\mathbf{h}\|^2, \quad (33)$$

where r_{s_3} is the radius of the sphere S_3 . The sphere S_3 does not touch the plane if $r_{c_2}^2$ has a negative value. Two vertices determined by three spheres are the intersection

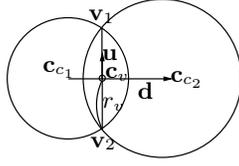


Figure 12: Intersection of two circles

of the circles C_1 and C_2 . To compute the intersection of C_1 and C_2 , we evaluate the mid-point \mathbf{c}_v of the vertices \mathbf{v}_1 and \mathbf{v}_2 (see Figure 12.) Similarly to the sphere-sphere intersection, the mid-point \mathbf{c}_v and the distance r_v from each of vertices to \mathbf{c} are given as follows:

$$r_v^2 = r_{c_1}^2 - \frac{(r_{c_1}^2 - r_{c_2}^2 + \|\mathbf{d}\|^2)^2}{4\|\mathbf{d}\|^2}, \text{ and} \quad (34)$$

$$\mathbf{c}_v = \frac{r_{c_1}^2 - r_{c_2}^2 + \|\mathbf{d}\|^2}{2\|\mathbf{d}\|} \cdot \frac{\mathbf{d}}{\|\mathbf{d}\|} + \mathbf{c}_{c_1}, \quad (35)$$

where the \mathbf{d} is the vector from the \mathbf{c}_{c_1} to \mathbf{c}_{c_2} . The normalized direction vector \mathbf{u} from \mathbf{c}_v to \mathbf{v}_1 is obtained from the cross product of \mathbf{n} and \mathbf{d} , that is, $\mathbf{u} = \frac{\mathbf{n} \times \mathbf{d}}{\|\mathbf{n} \times \mathbf{d}\|}$. Hence, we have the vertices $\mathbf{v}_1 = \mathbf{c}_v + r_v \mathbf{u}$ and $\mathbf{v}_2 = \mathbf{c}_v - r_v \mathbf{u}$.



(a) the captured joint angles only

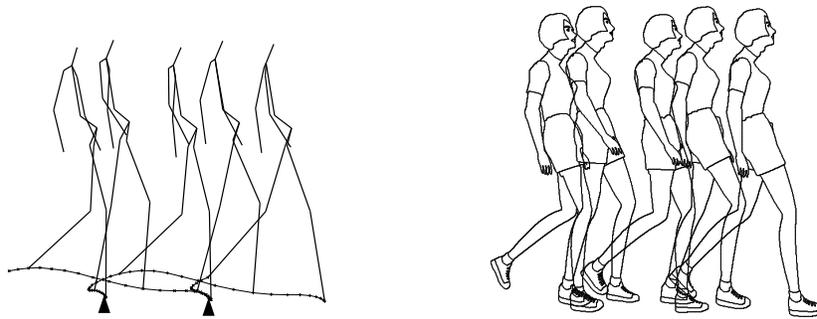


(b) a conventional IK solution with kinematic constraints on end-effectors

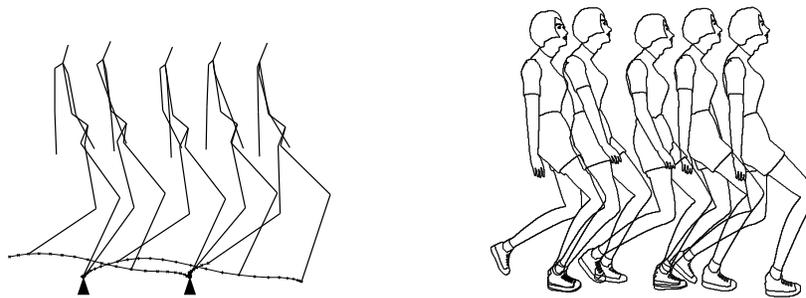


(c) Proposed algorithm combining the captured joint angles and the IK solution

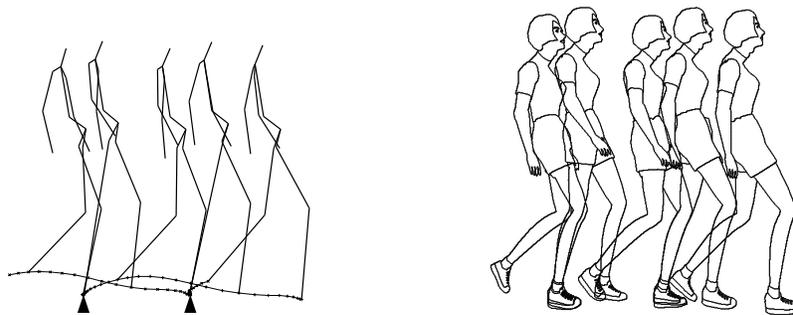
Figure 13: Walking motion of *Blubby*



(a) the captured joint angles only

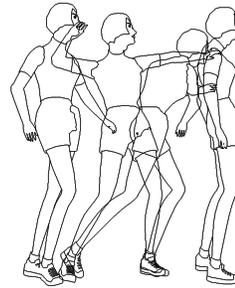
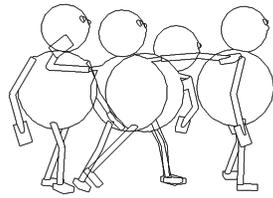


(b) a conventional IK solution with kinematic constraints on end-effectors

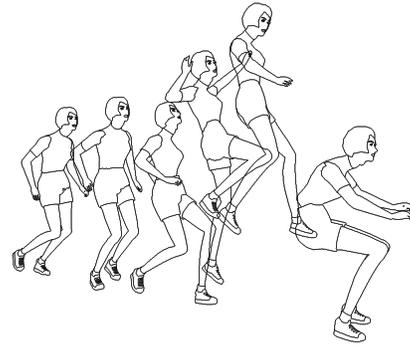
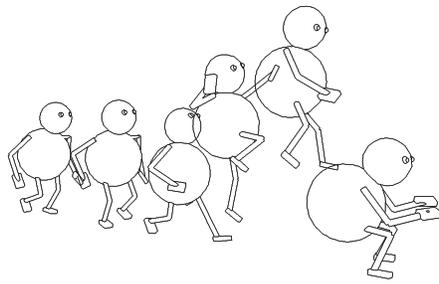


(c) Proposed algorithm combining the captured joint angles and the IK solution

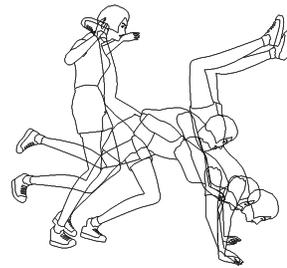
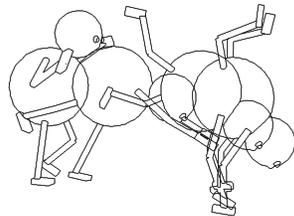
Figure 14: Walking motion of *Sally*



(a) Throwing

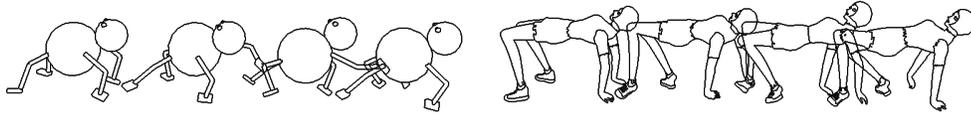


(b) Jumping

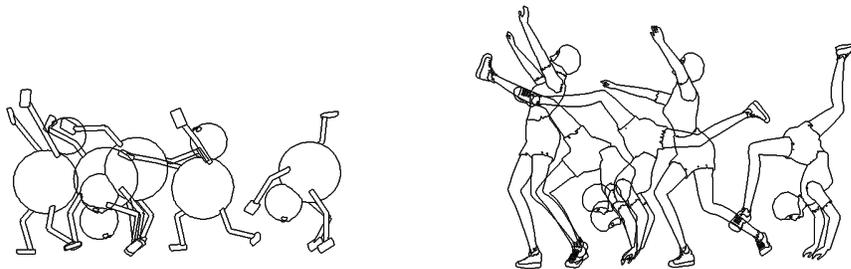


(c) Handstand

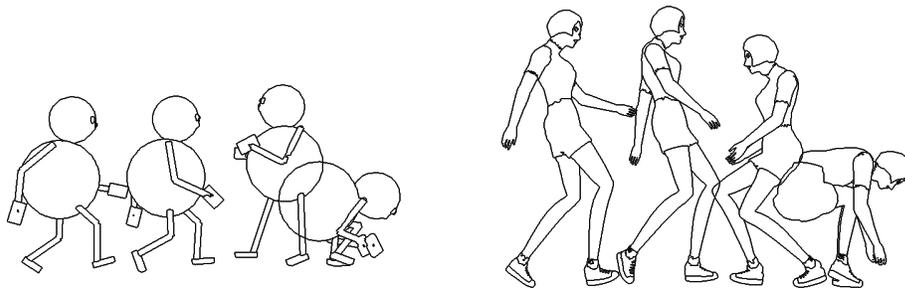
Figure 15: Example motions of *Blubby* and *Sally*



(a) Crawling



(b) Backflipping



(c) Picking a box up

Figure 16: Example motions with interaction of hands