Transactions on Computational Science XXIII, pp. 78-97, Springer, 2014.

Multi-touch Interface and Motion Control Model for Interactive Character Animation

Masaki Oshita

Kyushu Institute of Technology
Iizuka, Fukuoka, Japan

Abstract. In this paper, we propose a new method for interactive motion control with a multi-touch interface. A user of our system can touch and drag character's body parts to control its motion. The character's full body motion is driven by our interactive motion control model based on the movement of a few body parts which are directly manipulated by the user via the multi-touch interface. We propose a method for determining 3-dimensional positions of controlled body parts from 2-dimensional touch inputs based on the character's local coordinates and drag speed. We introduce a point-based pose representation which consists of the positions or orientations of a small number of primary body parts. Based on the representation, we develop a motion control model that includes modules for tracking, balance, inter-body interaction, relaxing and self-collision avoidance. The character's pose is reconstructed from the point-based pose representation. We present our experimental results to show that our framework can realize various natural-looking motions.

Keywords: motion control, multi-touch interface, computer animation, character animation

1 Introduction

Tablet computers that support multi-touch inputs have recently become commonplace. Although many applications control character motion in the virtual environment, most use virtual buttons or stroke gestures for selecting actions. Multi-touch input is a simple substitute for gamepads or keyboards. The strengths of multi-touch are underutilized. Using multi-touch, users should be able to control character motion freely and intuitively rather than simply executing predefined actions.

In theory, using inverse kinematics (IK), a user can change a character's pose by dragging its body part. However, this kind of interface is not suitable for controlling a character's motion in interactive applications, for two primary reasons. First, because multi-touch inputs on the screen are 2-dimensional, 3-dimensional position and orientation of body parts cannot be easily controlled. Second, since multiple body parts must be moved in a coordinated way to realize natural-looking motion, a user must control multiple body parts, which is very difficult. For example, to execute a punch motion, in addition to the hand, the pelvis and trunk should also be moved.

Fig. 1. Examples of interactive motion control with multi-touch interface. The red circles represent touch inputs.

A statistics-based model for IK (style-based IK [1]) can be a solution for these problems. Using a large number of example poses, natural-looking pose and motion can be synthesized based on touch inputs. Oshita [2] applied this approach for a multi-touch interface for motion control. However, because different example data sets are required for each kind of actions, this approach requires a large number of examples and a mechanism to switch data sets automatically. Moreover, it is difficult to execute new types of actions whose example poses are not provided in advance.

In this paper, we propose a new method for interactive motion control with a multi-touch interface. A user of our system can touch and drag a character's body parts to control its motion. Unlike the data-based approach (the statistics-based IK) discussed above, we take a model-based approach. The character's full body motion is driven by our interactive motion control model based on the movement of a few body parts which are directly manipulated by the user via the multi-touch interface. Our motion control model can perform various motions without preparing any example poses (Figure 1).

Our method for determining 3-dimensional positions of controlled body parts from 2-dimensional touch inputs is based on the character's local coordinates and drag speed. We introduced a point-based pose representation which consists of the positions or orientations of a small number of primary body parts (pelvis, hand, foot, trunk and head). Based on this representation, we developed a motion control model that includes modules for tracking, balance, inter-body interaction, relaxing and self-collision avoidance. The character's pose is reconstructed from the point-based pose representation. We present our experimental results to show that our framework can realize various natural-looking motions.

Even though our method can accept a number of multi-touch inputs, in our experience it is difficult for a user to control multiple touches simultaneously. With our method, a user needs to control the movement of only one or a few primary body parts to perform an action. The motion control model then generates the full body motion accordingly.

This paper is an extended version of our previous work [3]. We describe more specific details of our motion control model (Section 6). We also provide a result from a user test which is a comparison with conventional mouse-based interface for posing a character (Section 7).

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 shows the system overview. Sections 4, 5 and 6 explain our methods for pose representation, interpretation of multi-touch inputs, and the motion control model, respectively. Section 7 presents the experimental results and discussion. Finally, Section 8 concludes this paper.

2 Related Work

2.1 Point-based Pose Representation

We used a point-based pose representation in this research. Representing a pose by using points is an efficient way to handle character motion. Jakobsen [4] represented a character as a set of connected particles for an efficient physics simulation. However, he did not consider reconstructing a full body pose. Popović and Witkin [5] reconstructed full body motion using an optimization process. These approaches are not applicable to our research.

Similar pose representations to our method which use the positions and orientations of primary body parts have been used in previous research [6][7]. The difference between these studies and our method is that they used their representation for encoding existing motions for motion editing or retargetting. Kulpa et al. [6] used a numerical IK for limbs. Neff et al. [7] included the center of mass position in the pose representation and computed the ankle joints accordingly during pose reconstruction. We didn't take such approaches to avoid redundancy and achieve robust and efficient method.

2.2 Motion Control Interface

There are several existing multi-touch interfaces for interactive motion control. Krause et al. [8] applied conventional IK to a character model based on multi-touch inputs for animation. Kip and Nguyen [9] proposed a system to control one arm and hand of a character using a multi-touch interface by changing several parameters to blend arm and hand postures. Oshita [2] applied a statistics-based IK (style-based IK [1]). Although this approach successfully generates natural-looking full-body pose and motion based on multi-touch inputs, as explained in Section 1, it requires a large number of examples and a mechanism to switch data sets automatically. Moreover, it is difficult to execute new types of actions whose example poses are not provided. On the other hand, our research used a model-based approach and our motion control model makes it possible to perform various motions without preparing any example poses.

There are many methods for controlling a character's motion by using a single point or trajectory. Generating locomotion along with a given trajectory is common [10]. However, the type of motion is limited to walking or running. Throne et al. [11] introduced gesture-based motion selection. Based on gestures drawn along a trajectory, their system inserts predefined motions such as a jump or flip. Oshita [12] proposed a stroke-based motion selection technique

that chooses an appropriate action according to the initial and terminal points of a single stroke drawn on the screen. With these systems [11][12], users can simply select actions but the postures and speed of the actions are fixed and cannot be controlled. Igarashi et al. [13] proposed a spatial keyframing animation technique. This approach enables changing a character's pose continuously based on the cursor position. To use this technique, key postures must be placed at appropriate positions depending on a specific action.

Some systems allow a user to specify a number of trajectories and constraints for motion creation [14] and deformation [15]. However, with these systems, the user is expected to prioritize the inputs. Our methods uses multiple inputs and control modules and prioritizes them automatically.

2.3 Motion Control Model

There are various approaches for motion control and synthesis based on user input. A combination of physics simulation and physics-based controllers [16][17] is one approach. A controller determines joint torques based on a target pose, balancing, etc. and the physics simulation generates physically valid motions. However, designing a stable controller is difficult and different controllers must be designed for each kind of action.

Space-time optimization is another approach [5][18]. It synthesizes a continuous motion based on given constraints such as footsteps and timings so that the generated motion minimizes an objective function that evaluates its physical validity. However, this approach generate a motion sequence and requires computational time. Therefore, it is difficult to apply it for interactive motion control.

Previous researches has applied optimization (Quadratic Programming) for computing the pose of the next frame instead of a motion sequence [19][20] to realize interactive motion generation. These controllers are designed for autonomous control instead of user control and also require computational time.

In contrary to these approaches, we developed a kinematics based controller. Our controller considers physics in part but directly changes the positions and orientations of body parts rather than using physics simulation or optimization. Our controller is designed to move the character's full body naturally based on the user's inputs.

3 System Overview

The structure of our framework is shown in Figure 2. We use an intermediate point-based pose representation for motion control (controlled pose). Multi-touch inputs from the user are interpreted and represented as constraints in the same pose representation.

In addition to constraints for controlling the character's pose, when the character is moved over a certain distance, a moving motion (step) is executed. In

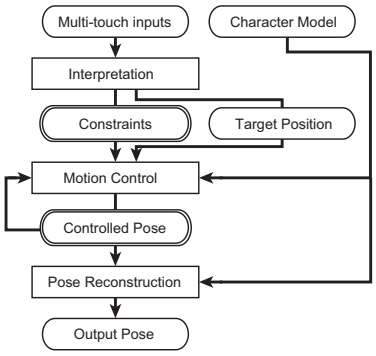

Fig. 2. Data flow in the proposed framework.

this case, the target position of the moving motion is sent to the motion control module.

The skeletal structure of the character is given to the system in advance. It includes the information on the shape and weight of the body parts. Shape information is necessary for self-collision avoidance, and weight information is necessary for balance control.

The user can also control the camera using multi-touch interface. A swipe can control the camera direction and pinch-in and -out can control zoom (the distance from the camera to the character).

4 Point-based Pose Representation

Our intermediate point-based pose representation $\mathbf{P}$ includes pelvis position $\mathbf{p}_{pelvis}$, hand positions $\mathbf{p}_{r\_hand}$, $\mathbf{p}_{l\_hand}$, foot positions $\mathbf{p}_{r\_foot}$, $\mathbf{p}_{l\_foot}$, trunk orientation $\mathbf{q}_{trunk}$ and head orientation $\mathbf{q}_{head}$; 5 positions and 2 orientations as shown in Figure 3. All positions and orientations are represented in the absolute (world) coordinates.

In general, there are several ways to represent a rotation such as quaternion, axis-angle, 3×3 matrix and Euler angles. Although any of these can be used with our method, in the following explanation, we treat them as 3×3 matrices such that the product of two rotations $\mathbf{q}_b\mathbf{q}_a$ equates to the combination of two rotations. $\mathbf{q}^{-1}$ represents the inverse of the rotation. $\mathbf{qv}$ denotes the rotation of a vector $\mathbf{v}$. In this paper, $|\mathbf{q}|$ represents the rotational angle and $w\mathbf{q}$ represents the scaling of the rotational angle.

4.1 Pose Reconstruction

The character's output pose $\mathbf{X}$ is represented by the position and orientation of the pelvis and rotations for all joints as $\mathbf{X} = \{\mathbf{p}_{pelvis}, \mathbf{q}_{pelvis}, \mathbf{q}_i(i = 1...n)\}$ where n is the number of joints. Our method reconstructs this output pose from our

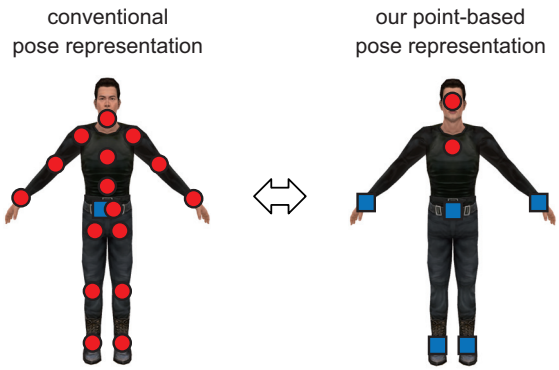

Fig. 3. Pose representation. A blue rectangle represents a position and a read circle represents a rotation (orientation).

intermediate pose representation $\mathbf{P} = \{\mathbf{p}_{pelvis}, \mathbf{p}_{r\_hand}, \mathbf{p}_{l\_hand}, \mathbf{p}_{r\_foot}, \mathbf{p}_{l\_foot}, \mathbf{q}_{trunk}, \mathbf{q}_{head}\}$. Figure 3 shows the difference between the conventional pose representation and our point-based pose representation. In our experiments, we used a character model which has 15 joints in the conventional pose representation as shown in Figure 3. The pose reconstruction follows 4 steps as explained below:

Pose Reconstruction for Pelvis Position, Pelvis Orientation and Back Joints The pelvis position of the intermediate representation is simply used for the pelvis position $\mathbf{p}_{pelvis}$.

Pelvis orientation and back joint rotations are computed from trunk orientation $\mathbf{q}_{trunk}$. The number of back joints depends on the skeleton model. In general, the back joint rotations can be computed from the total rotation of the back joints by distributing the total rotation to each back joint in specific ratios [21]. Our method determines pelvis orientation in addition to the back joint rotations. We divide the trunk orientation into horizontal rotation $\mathbf{q}_{trunk\_h}$(1DOF) and front-back and right-left rotations $\mathbf{q}_{trunk\_v}$(2DOF). The horizontal rotation is assigned to pelvis orientation. The other rotations are distributed into pelvis orientation $\mathbf{q}_{pelvis}$ and back joints $\mathbf{q}_{i(back)}$ with a specific weight $w_{pelvis\_back\_ratio}$. In our implementation, we use $w_{pelvis\_back\_ratio} = 0.5$.

$$\mathbf{q}_{trunk} = \mathbf{q}_{trunk\_v}\mathbf{q}_{trunk\_h} \tag{1}$$

$$\mathbf{q}_{pelvis} = ((w_{pelvis\_back\_ratio})\mathbf{q}_{trunk\_v})\mathbf{q}_{trunk\_h} \tag{2}$$

$$\mathbf{q}_{i(back)} = (1 - w_{pelvis\_back\_ratio})\mathbf{q}_{trunk\_v} \tag{3}$$

If the skeleton model contains more than one back joints, the total back rotation $\mathbf{q}_{i(back)}$ can be further distributed to each back joint with specific weights. The number of back joints of a human character model can be 19 at most [21]. Typically a character model with from one to three back joints is used. In our

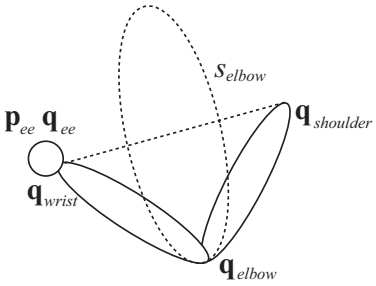

Fig. 4. Analytical inverse kinematics for a limb.

implementation, we used a character model with two back joints as depicted in Figure 3.

Pose Reconstruction for Neck Joints The neck joint rotations are computed from trunk and head orientations.

$$\mathbf{q}_{i(neck)} = \mathbf{q}_{head}(\mathbf{q}_{trunk})^{-1} \tag{4}$$

Pose Reconstruction for Limb Joints After the pelvis and trunk states are determined, limb joints rotations are computed from the end-effector (hand or foot) position. In the followings explanation, we take the case of an arm to make the explanation easier, although the same method is applied to legs too.

In general, as shown in Figure 4, an analytical IK [22] determines the rotation of limb joints (3 DOF shoulder joint $\mathbf{q}_{shoulder}$, 1 DOF elbow joint $\mathbf{q}_{elbow}$ and 3 DOF wrist joint $\mathbf{q}_{wrist}$) based on the relative position and orientation of the end-effector from the shoulder $\mathbf{p}_{ee}$, $\mathbf{q}_{ee}$ and swivel angle of the elbow s_{elbow}. Since we determined the wrist angle joint rotations differently as explained in the next subsection, if we focus on shoulder (3 DOF) and elbow (1 DOF) joints, their rotations can be determined based on hand position $\mathbf{p}_{ee}$ (3DOF) and swivel angle s_{elbow}(1 DOF).

How swivel angle is determined is important. If analytical IK is used for changing an existing pose, the swivel angle of the original pose can be retained. Because of the need to generate arbitrary poses in our case, such an approach is not applicable. It is known that the swivel angle depends on the end-effector position [23]. Therefore, we determined the swivel angle from the hand position based on examples that are prepared in advance.

We prepared sets of normalized relative hand positions and swivel angles as examples. Given a hand position, we blend nearby samples to determine swivel angle. We use Radial Basis Function (RBF) to compute the weight of each example depending on hand position.

$$s_{elbow} = \sum f_i(\mathbf{p}_{ee}) s_i \tag{5}$$

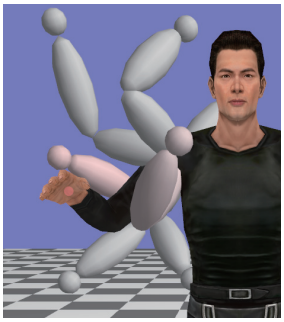

Fig. 5. Examples of swivel angles.

$$f_i(\mathbf{p}_{ee}) = exp(-(|\mathbf{p}_i - \mathbf{p}_{ee}|/r_i)^2) \ \ (if \ |\mathbf{p}_i - \mathbf{p}_{ee}| < r_i) \tag{6}$$

where $f_i(\mathbf{p}_{ee})$ is the RBF for each example $\{\mathbf{p}_i, s_i, r_i\}$. In our implementation, we use about 10 examples for each limb which are tuned manually. Alternatively, it is possible to use motion capture data to create more accurate samples.

Although we used example data here, because these examples are common for all types of poses and actions, it is not necessary to prepare separate sets of examples for each type of action, unlike a statistics-based posture synthesis [1][2].

Figure 5 shows poses that are generated from the examples. The colors represent the weights computed from the hand position. Note that each example has a swivel angle and not a pose.

Pose Reconstruction for Hand and Foot Joints Since we do not use hand and foot orientations, wrist and ankle rotations $\mathbf{p}_{wrist}$, $\mathbf{p}_{ankle}$ are determined automatically. Unless the character is performing a gesture or holding an object in the environment, neither is considered in our system, it is natural to keep the wrist and foot rotations in the rest pose. Therefore, we simply set the joint rotation to zero after limb joint rotation is determined.

When an end-effector (typically foot) is contacting the ground, the foot must be kept horizontal. Also, when the foot is near the ground, the ankle joint must be flexed or extended to prevent the foot from penetrating into the ground. Similar to trunk orientation control, ankle rotation is set, when the foot is near or on the ground.

5 Interpretation of Multi-touch Inputs

In this section, we explain how to interpret multi-touch inputs to determine constraints in the point-based pose representation form. A user can touch and drag the pelvis, hand, foot, trunk and head of the character. Since we use multi-touch inputs, multiple body parts can be touched and dragged at the same time.

In this paper, we did not take inputs on the middle of limbs (e.g. upper arm, forearm, elbow, etc.); limbs can be controlled only by moving the end-effectors (hand and foot). Although a person has 10 fingers, from our experience, we can control at most two or three touches at the same time. Therefore, the limited number of controllable body parts in our method is considered reasonable.

Our method treats each touch input as the spatial translation of the touched body part. The biggest challenge is how to determine 3-dimensional positions of controlled body parts. When a user touches and drags a body part on the 2-dimensional screen, the touched position on the screen represents a line in the 3-dimensional scene. There is no simple way to determine a unique point on the line.

We solved this problem by introducing several assumptions. First, when a person performs motions, the pelvis, hand and foot are generally moved in either front-back or right-left direction relative to the person. For example, for punch and kick motions, the body parts are moved in the front-back direction. For a waving hand and arm extending motion, the body parts are moved in the right-left direction. Although some complex motions include movements in combinations of both front-back and left-right directions, in many cases the body parts are moved on one plane. We determined in which plane the touched body part should be moved depending on the character's orientation and the camera direction.

A second assumption is to move the selected body part (hand or foot) in a perpendicular direction to the screen. When a hand or foot is moved away from the trunk on the screen, there is not much freedom of movement in the perpendicular direction, because arm and leg lengths are limited. However, when the hand or foot is moved toward or near the trunk, it may be moved in the perpendicular direction. When the hand or foot is moved toward the camera, it is likely that they are dynamic motions such as a punch or kick which would require quick movement. Therefore, the resultant translation in the perpendicular direction is based on the speed of the body part being controlled by the user. Although this may not be the case all of the time, we consider that this is a reasonable assumption.

In addition to these assumptions, self-collision avoidance is considered. For example, when a hand is moved toward the trunk on a plane that crosses the trunk, the arm can penetrate into the trunk. In this case, the hand position is adjusted to a position where such penetration does not happen.

5.1 Translation of Pelvis, Hand and Foot

Based on the above approach, we determined the target position of pelvis, hand or foot as follows:

Translation on a Hyper Plane Based on the first assumption above, when a body part (pelvis, hand or foot) is touched and dragged on the screen, it is moved on either the XY, ZX or YZ hyper plane which is defined by the local coordinates of the character as shown in Figure 6.

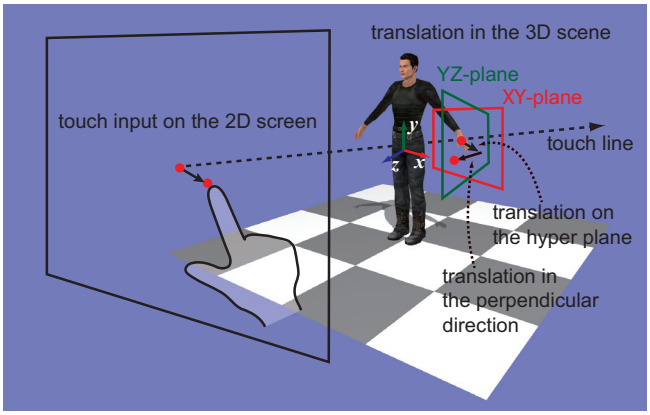

Fig. 6. Touch interpretation.

X, Y and Z axes are determined based on pelvis orientation. Y axis is always (0, 1, 0), while X and Z axes are determined from the horizontal pelvis orientation. Then, among these axes, the one whose inner product with the camera vector (camera direction) $\mathbf{d}_{camera}$ is the largest is chosen as the normal vector that defines the hyper plane. The hyper plane is defined by the normal vector $\mathbf{n}$ and the current position of the controlled body part $\mathbf{p}_{current}$ as follows

$$\mathbf{n}(\mathbf{p} - \mathbf{p}_{current}) = 0 \tag{7}$$

The touch line where the controlled body part exist is defined as follows

$$\mathbf{p} = t\mathbf{d}_{touch} + \mathbf{p}_{camera} \tag{8}$$

where $\mathbf{d}_{touch}$ is the normalized vector from the camera to the touch point and $\mathbf{p}_{camera}$ is the camera position. From equations (7) and (8), the target position of the controlled body part on the hyper plane is computed.

Translation in the Perpendicular Direction Based on the second assumption above, position is adjusted in the direction perpendicular to the screen. This adjustment is applied to end-effectors (hand and foot) but not to the pelvis. When the speed of a dragged end-effector is faster than the threshold, its position is adjusted in proportion to speed.

There are two directions for the perpendicular vector. The body part should be moved forward from the character. The moving direction vector $\mathbf{d}_{forward}$ is determined as follows.

$$\mathbf{d}_{forward} = \begin{array}{ll} \mathbf{d}_{touch} & (if\ \mathbf{d}_{z\_axis} \cdot \mathbf{d}_{touch} > cos45°) \\ -\mathbf{d}_{touch} & (if\ \mathbf{d}_{z\_axis} \cdot \mathbf{d}_{touch} < cos45°) \\ \mathbf{0} & (otherwise) \end{array} \tag{9}$$

where $\mathbf{d}_{z\_axis}$ is the Z axis of the character's pelvis. Although it is possible to apply this translation on any camera direction in theory, throughout our tests, we found that it does not work well when the body parts are moved on the YZ plane (when the camera is on the side of the character). Therefore, we limit this control when the angle between $\mathbf{d}_{z\_axis}$ and $\mathbf{d}_{touch}$ is less than 45°.

The position of the end-effector is adjusted in the direction of $\mathbf{d}_{forward}$ based on the velocity of the end-effector $\mathbf{v}$.

$$\mathbf{p}' = \mathbf{p} + \frac{|\mathbf{v}| - v_{th}}{v_s}\mathbf{d}_{forward} \ \ (if\ |\mathbf{v}| > v_{th}) \tag{10}$$

where v_{th}, v_s are the threshold and scaling parameters. We tuned those parameters empirically. In our implementation, we use $v_{th} = 1.0$ and $v_s = 0.2$.

Translation for Self-collision Avoidance Finally, the end-effector is moved in the same perpendicular direction to avoid self-collision.

The distances between the controlled end-effector and other body parts are computed. If the distance is below a threshold, the position is adjusted using the similar equation to equation (10).

$$\mathbf{p}' = \mathbf{p} + k\mathbf{d}_{forward} \tag{11}$$

In this case, the scaling parameter k is computed based on the distance and the threshold.

For collision detection and distance computation between body segments, any existing method can be used. The method for representing the shapes of body segments is also flexible. A simplified representation such as a bounding box or ellipsoid makes distance computation easier. In our implementation, we represented each body part as an oriented bounding box [24].

5.2 Rotation of Trunk and Head

Because we use the rotations of the trunk and head in our point-based pose representation, we interpreted the touch and drag of a point on a body part as their rotation.

The translation of the touched point of the trunk or head is computed the same way as the computation for pelvis translation $\mathbf{p}'_{touched}$. The touched point on the body depends on the point where the user touches first. Based on the translation of the touched point, the trunk rotation that satisfies it is computed $\Delta\mathbf{q}_{trunk}$ as follows.

$$(\mathbf{p}'_{touched} - \mathbf{p}_{back}) = \Delta\mathbf{q}_{trunk}(\mathbf{p}_{touched} - \mathbf{p}_{back}) \tag{12}$$

where $\mathbf{p}_{touched}$ is the initial position of the touched point without trunk rotation and $\mathbf{p}_{back}$ is the back joint position. Because the touched point is on the surface of the character's body, when the touched body is moved downward on the screen, the character bends forward, and when the touched body is moved upward on the screen, it bends backward.

5.3 Execution of Moving Motion

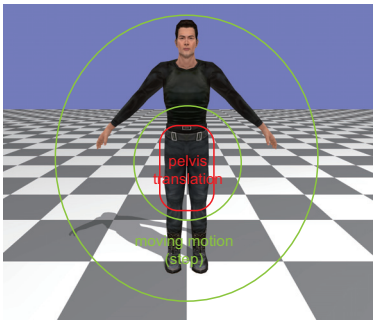

Fig. 7. Touch interpretation for the pelvis.

As explained in Section 3, when the pelvis is moved over a large distance, this is interpreted as moving in a step and the target position is computed.

Figure 7 shows the touch interpretation for the pelvis. The horizontal movement of the pelvis is limited within the support polygon computed from foot positions (see Section 6.2). Vertical movement of the pelvis is limited by leg length and minimum duck height. When the pelvis is moved within this range, the input is interpreted as pelvis translation. When the pelvis is moved outside this range, the input is interpreted as moving motion (step) and the target position is computed.

The target position $\mathbf{p}_{move}$ is computed by computing the crossing point of the viewing vector with the plane that is parallel to the ground and crosses the pelvis of the character. To keep the step distance within a reasonable range, the distance between $\mathbf{p}_{move}$ and the current pelvis position is limited within a specific range (0.5m $\sim$ 0.8m in our implementation). Note that up-down touch movement on the screen controls front-back step direction.

6 Motion Control Model

In this section, we describe our motion control model that determines the output pose $\mathbf{P}_{output}$ based on the constraints from multi-touch inputs $\mathbf{P}_{input}$ and the previous pose $\mathbf{P}_{prev}$. These are represented by the intermediate point-based pose representation described in Section 4.

Our motion control model includes several modules and each of them determines an output pose. The output pose is computed by blending the results of all modules.

$$\mathbf{P}_{output} = (\sum \frac{w_i}{\sum w_i} \mathbf{P}_i) \tag{13}$$

where $\mathbf{P}_i$, w_i are the output pose and weight of each control module and $i =$ *tracking, balance, inter_body_ interaction, relaxing and self_collision_avoidance*. Note that all modules may not determine all parameters. For example, when there is no user input, the tracking control does not produce its output pose and weight. High priority is given to tracking, balance and self collision avoidance while low priority is given to inter body interaction and relaxing.

The concepts behind our each control module are not new. We designed these models based on our intermediate point-based pose representation. Our models directly change positions and orientation rather than using physics simulation. We introduced simple approximations and parameters for these modules. We argue that our framework is very simple and easy to implement, but also powerful and flexible.

In the remainder of this section, we briefly describe our approach for implementing each controller.

6.1 Tracking Control

Tracking control is for satisfying input constraints from the multi-touch interface. The output of the tracking control module $\mathbf{P}_{tracking}$ uses input constraints as $\mathbf{P}_{tracking} = \mathbf{P}_{input}$.

When an end-effector (hand or foot) is controlled by the user and the target position is within the reachable range of the limb, only the end-effector position is changed. However, when the target position is outside of the reachable range, the pelvis and the trunk should be moved to reach the target position. In such case, the pelvis position $\mathbf{p}_{pelvis}$ and the trunk orientation $\mathbf{q}_{trunk}$ are changed accordingly as follows.

$$\Delta\mathbf{p}_{root} = \mathbf{p}_{root}^{input} - \mathbf{p}_{root} \tag{14}$$

$$(\Delta\mathbf{q}_{trunk}(\mathbf{p}_{trunk} - \mathbf{p}_{back}) - (\mathbf{p}_{trunk} - \mathbf{p}_{back})) + \Delta\mathbf{p}_{pelvis} = \Delta\mathbf{p}_{root} \tag{15}$$

where $\Delta\mathbf{p}_{root}$ is the required translation of the root of the limb (shoulder). $\mathbf{p}_{trunk}$ and $\mathbf{p}_{back}$ are the positions of the trunk and back joints, respectively. First, the trunk orientation $\Delta\mathbf{q}_{trunk}$ is computed to satisfy $\Delta\mathbf{p}_{root}$ as much as possible. Then, the pelvis translation $\Delta\mathbf{p}_{pelvis}$ is computed to satisfy the remaining required translation $\Delta\mathbf{p}_{root}$.

Also, while an end-effector (foot) is contacting the ground, tracking control keeps the current foot position. When no constraint is specified to a body part, the tracking control does not handle the body part, as result, the body part is controlled by other modules.

Since the priority of tracking control should be high, $w_{tracking}$ is set to 1.0 (the highest weight).

6.2 Balance Control

Balance control maintains the balance of the character. Our balance control module controls pelvis position and trunk orientation to keep the projection of the center of mass of the character within its support polygon.

First, based on the character's current state, the position of the center of mass and its projection to the ground $\mathbf{p}_{com}$ is computed. The support polygon S is also computed based on the positions and contact conditions of the feet. Then it is determined if $\mathbf{p}_{com}$ is within S. If not, the closest point to $\mathbf{p}_{com}$ within S is computed as $\mathbf{p}'_{com}$. To maintain the character's balance, the center of mass is moved from $\mathbf{p}_{com}$ to $\mathbf{p}'_{com}$.

$$\Delta\mathbf{p}_{com} = \mathbf{p}'_{com} - \mathbf{p}_{com} \tag{16}$$

The translation of the upper body (position of the pelvis) $\Delta\mathbf{p}_{pelvis}$ and the rotation of the upper body (orientation of the trunk) $\Delta\mathbf{q}_{trunk}$ are computed to move the center of mass within the support polygon $\Delta\mathbf{p}_{com}$, when it is found to be outside the support polygon.

$$M\Delta\mathbf{p}_{pelvis} = w_{pelvis}\Delta\mathbf{p}_{com} \tag{17}$$

$$M(\Delta\mathbf{q}_{trunk}(\mathbf{p}_{trunk} - \mathbf{p}_{back}) - (\mathbf{p}_{trunk} - \mathbf{p}_{back})) = w_{trunk}\Delta\mathbf{p}_{com} \tag{18}$$

where M is the total mass of upper body. The required translation of the center of mass $\Delta\mathbf{p}_{com}$ is distributed to the translation of the pelvis position $\mathbf{p}_{pelvis}$ and the orientation of the trunk $\mathbf{q}_{trunk}$ with a specific ratio $w_{pelvis} : w_{trunk} = w_{pelvis\_trunk\_ratio} : 1 - w_{pelvis\_trunk\_ratio}$. In our implementation, we use $w_{pelvis\_trunk\_ratio} = 0.5$.

Since the priority of balance control is as high as tracking control, $w_{balance}$ is set to 1.0 too.

6.3 Inter-Body Interaction Control

This control is for simulating the physical interface between connected body parts. When a person moves his or her body part (e.g. an arm), any connected body part (e.g. trunk) also moves a little even if he or she tries to keep it still because there is physical influence between the connected body parts.

This module controls trunk orientation based on the velocities of end-effectors and the positions of end-effectors based on the velocity of the trunk. Because it is difficult to simulate this kind of effect accurately even with physics simulation, because it also requires realistic muscle stiffness models, we chose to scale the velocities of connected body parts.

The change of trunk orientation $\Delta\mathbf{q}_{trunk}$ is computed as follows.

$$\mathbf{v}_i = (\mathbf{p}_i - \mathbf{p}_i^{prev})/\Delta t \tag{19}$$

$$\Delta\mathbf{q}_{trunk}(\mathbf{p}_{trunk} - \mathbf{p}_{back}) - (\mathbf{p}_{trunk} - \mathbf{p}_{back}) = w_{ee\_to\_trunk}\Delta t\mathbf{v}_i \tag{20}$$

where $\mathbf{p}_i, \mathbf{p}_i^{prev}$ are the current and previous end-effector positions, $w_{ee\_to\_trunk}$ is the influence from the end-effector velocity to the trunk orientation. In our implementation, we use $w_{ee\_to\_trunk} = 2.0$.

The end-effector position is computed as follows.

$$\mathbf{v}_i = ((\Delta\mathbf{q}_{trunk}(\mathbf{p}_i - \mathbf{p}_{trunk}) - (\mathbf{p}_i - \mathbf{p}_{trunk})) + (\mathbf{p}_{pelvis} - \mathbf{p}_{pelvis}^{prev}))/\Delta t \quad (21)$$

$$\mathbf{p}_i' = \mathbf{p}_i + w_{trunk\_to\_ee}\Delta t\mathbf{v}_i \quad (22)$$

We use $w_{trunk\_to\_ee} = 0.2$. This is not applied to the end-effector (feet) that is contacting on the ground.

Since the priority of this control is low, a low value is set to $w_{interaction}$ (0.1 in our implementation).

6.4 Relaxing Control

Relaxing control is for moving the body parts to the rest pose when there is no user input. We introduced this module because it looks unnatural if the character stops in an unnatural pose. This module keeps the trunk vertical, lowers arms and legs, and maintains the head elevation within certain limits.

The trunk orientation is computed by

$$\mathbf{q}_{trunk} = \mathbf{q}_{trunk_v}\mathbf{q}_{trunk_h} \quad (23)$$

$$\mathbf{r} = {\mathbf{q}_{trunk_v}}^{-1} \quad (24)$$

$$\mathbf{q}_{trunk}' = \begin{cases} \mathbf{q}_{trunk_h} & (if\ |\mathbf{r}| \leq v_{trunk}\Delta t) \\ \frac{v_{trunk}\Delta t}{|\mathbf{r}|}\mathbf{r}\mathbf{q}_{trunk} & (if\ otherwise) \end{cases} \quad (25)$$

where $\mathbf{q}_{trunk_h}$ and $\mathbf{q}_{trunk_v}$ are decomposition of the trunk rotation into the horizontal orientation and the other component. The rotation $\mathbf{r}$ makes the trunk orientation closer to $\mathbf{q}_{trunk_h}$ in a predefined angular speed v_{trunk}.

Each end-effector positions is computed by

$$\mathbf{p}_i' = \mathbf{p}_i + \Delta t\mathbf{g}\ \ (if\ \mathbf{p}_{i,y} > h_{ee}) \quad (26)$$

where $\mathbf{g} = (0, v_{ee}, 0)$ is a predefined spatial speed. This equation is applied when the position of the end-effector is higher than the predefined height h_{ee}.

When there is no input to the head. The head is moved based on the trunk. However, this does not look natural. When a person moves, the head elevation is kept horizontal. To realize such head movements, the head orientation $\mathbf{q}_{head}$ is computed by the same equation with equation (25).

Since the priority of this control is low, a low value is set to $w_{relaxing}$ (0.1 in our implementation).

There are more sophisticated methods for gazing (head and eye orientation) control. However, such control is not considered in our motion control model, because as such methods work only on a virtual environment with other characters and objects which can be gazing targets and those are not considered in our current system.

6.5 Self-Collision Avoidance Control

Self-collision avoidance requires care because we cannot know if self-collision will occur until the output pose is computed. Therefore, an interim output pose is computed without self-collision avoidance. If there is any self-collision, this control is then applied repeatedly until on self-collision problem exists.

This module solves self-collision between limbs and trunk by moving limbs. Assuming the limbs didn't penetrate into the trunk on the previous frame, the self-collision can be fixed by moving the limb (end-effector position) toward the state of the previous frame.

$$\mathbf{p}'_i = t\mathbf{p}_i + (1-t)\mathbf{p}_i^{prev} \tag{27}$$

where $t(0 < t < 1)$. t is reduced from 1 to 0, until the collision is resolved.

However, in case that the trunk is also moving, moving the limbs to the previous state may not solve the collision. In such case, the limb (end-effector position) is moved outward as follows.

$$\mathbf{p}'_i = \mathbf{p}_i + \epsilon(\mathbf{p}_i - \mathbf{p}_i^{root}) \times \mathbf{d}_z \tag{28}$$

where $\mathbf{p}_i^{root}$ is the position of the root of the limb (shoulder) and $\mathbf{d}_z$ is the z-axis of its local coordinates. ϵ is a small displacement for each step. This step is repeated until the self-collision is avoided.

Note there is the difference between this module and the method described in Section 5.1. The former deals with self-collision during motion control, while the latter is for interpreting multi-touch inputs. Both are important for not only avoiding self-collision but also for generating natural motion from the touch and drag inputs.

Since the priority of this control is high, $w_{self\_collision\_avoidance}$ is set to 1.0.

6.6 Step Motion Control

When a step is executed and its target position is sent from the interface module, the motion control generates a step motion. In our implementation, we used a similar model to that to that used by [25]. During action control, the pelvis and feet are controlled based on a procedural action model. The full body motion is automatically generated by our motion control framework.

The step control determines trajectories of the pelvis and feet. First, the goal position of both feet $\mathbf{p}_{r\_foot}^{goal}$, $\mathbf{p}_{l\_foot}^{goal}$ are computed from the given goal position of the pelvis $\mathbf{p}_{pelvis}^{goal}$ so that the relative positions is kept before and after step as follows.

$$\mathbf{p}_{r\_foot}^{goal} = \mathbf{p}_{pelvis}^{goal} + (\mathbf{p}_{r\_foot}^{initial} - \mathbf{p}_{pelvis}^{initial}) \tag{29}$$

Our step control generate one cycle of step, that is, a leg is moved first and the other leg follows. The leg that is near to the target position is moved first. For example, when the character steps to the right, the right leg is moved first. The first moving leg is determined based on the target position.

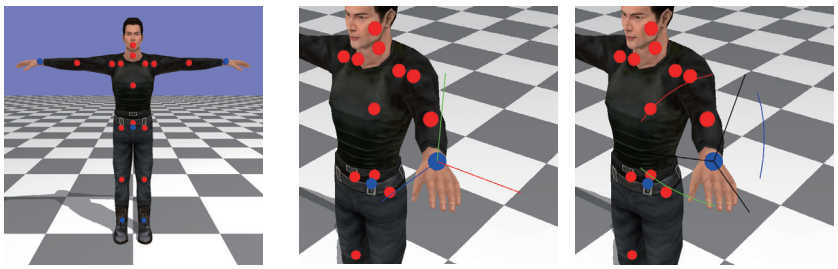

Fig. 8. Mouse-based interface for comparison. The position and orientation of a joint or end-effector can be controlled by dragging a handle.

The step action is executed in two phases; preparation and execution phases. During preparation phase, the pelvis is moved over the first supporting leg so that the character can move the moving leg. During execution phase, one cycle of step is executed. The trajectories of the pelvis and feet during those phases can be approximated by linear trajectories [25]. The durations of those phases are determined based on the length of the trajectories.

7 Results and Discussion

We have implemented our interface and tested it. Some of the resulting motions are shown in Figure 1 and the accompanying video. Various simple actions such as posing, reaching, stepping, gestures (nodding, pointing, waving), fighting actions (punch, kick) and combinations can be created by using our interface. As explained in Section 1, even though our method can take multi-touch inputs, it is difficult for a user to control many parts simultaneously. Based on our experiments, we found that using one to three touches at the same time is enough when performing typical actions. Each action typically has one primary limb (e.g. an arm for punch). One touch can be used to control its end-effector. Also, one touch can be used to control the body (pelvis and/or trunk). The rest of the body is driven by our motion control model.

We conducted a user test to show the efficiency of our multi-touch interface. Because there is no alternative common interface that can control full-body motion of a charter interactively, it is difficult to compare our interface for motion control tasks. Therefore, in this user study we evaluated our interface for posing control tasks. We compared our interface with a conventional mouse-based pose editing interface where a joint rotation or position can be manipulated using handles displayed on the screen as shown in Figure 8. The blue and red spheres on the character represent controllable end-effectors and joints, respectively. This kind of interface is commonly used in commercial animation systems such as maya, max and softimage. Although only one body can be controlled at a time, its position and rotation can be controlled precisely.

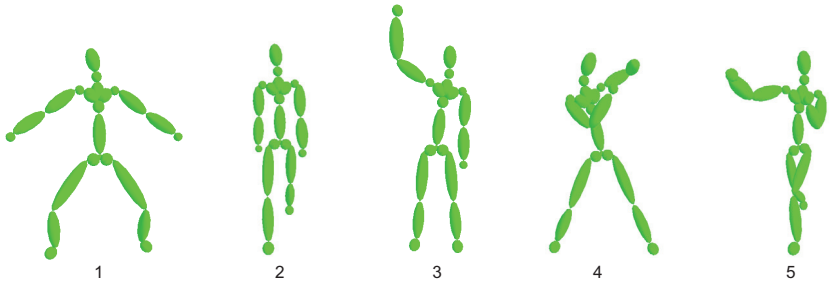

Fig. 9. Target poses used in the user test.

Table 1. Results from the user test. Average times for making each target pose.

Pose no.	Mouse interface	Multi-touch interface
1	27.4 s	8.6 s
2	33.2 s	9.9 s
3	34.7 s	11.9 s
4	76.2 s	25.6 s
5	132.5 s	40.1 s

Seven subjects who have basic knowledge of computer animation participated in the user test. During the experiment, a target pose was showed to the subject and he or she was asked to make the same pose by using one of the interfaces. The system determines that the task is completed when the maximum distance between the positions of all joints in the target and controlled poses becomes below the threshold (10 cm in our experiment). The five target poses used in our experiment are shown in Figure 9. The required times for making each target pose from the initial pose were measured. Each subject used both interfaces.

The average times for the posing tasks are shown in Table 1. For all target poses, our multi-touch interface was more efficient than the mouse-based interface. From our observation, the subjects first tried to make the lower body pose by using one or two touches and by changing the view direction if necessary. They then made the upper body pose by using multi-touch. This is because the relaxing control module lowers the arms if the user stops touching them. Since body orientation cannot be controlled by our multi-touch interface, it seemed that the subjects found it difficult to make some poses such as pose no. 4. The subjects also found it difficult to control end-effector positions when two hands are close to each other such as pose no. 4, because multiple touches cannot point the same position. The pose no. 5 took time the most, because the all feet and hands must be controlled both in front-back and right-left directions.

Using our interface, the user can make his or her avatar perform their own actions. This will be useful in many applications such as communication in virtual environments using avatars or fighting games. Conversely, performing all

motions using our interface may not be realistic. Making a combination of our interface and conventional interfaces available and allowing users to choose may be a more practical option.

There are limitations in our interface. Control is focused on the pose of a standing character and performing moving motions other than stepping was not considered in this paper. There are existing methods for such movement control [10][11] and our system can be integrated with these.

Because of the constraints of using inputs on a 2-dimensional screen, it is not possible to control which direction the character is facing, because the touch inputs are interpreted as translation or tilt of the controlled body parts and rotation is not considered. It may be possible to extend our system to include different methods of interpretation or to make it possible to switch between different interfaces. However, such extension makes the interface more complicated. Seeking a balance between freedom of control and usability is an important area for future research. Although our motion control model takes physics into account, it is not driven by physics simulation and may not always be physically correct. For example, pose and motion is limited so that the character never falls. We believe hat this is a reasonable constraint, but a user may want the character to perform a falling motion.

8 Conclusion

In this paper, we proposed a framework for controlling a character using a multi-touch interface. Although our framework is very simple, various kinds of motions can be realized using our interface without using any examples. Our interface is easy to implement and can be used by many tablet computers.

Acknowledgment

This work was supported in part by a Grant-in-Aid for Scientific Research (No. 24500238) from the Japan Society for the Promotion of Science (JSPS).

References

1. Grochow, K., Martin, S.L., Hertzmann, A., Popović, Z.: Style-based inverse kinematics. ACM Transactions on Graphics **23**(3) (2004) 522–531
2. Oshita, M.: Multi-touch interface for character motion control using example-based posture synthesis. In: International Conference on Computer Graphics, Visualization and Computer Vision (WSCG) 2012. (2012) 213–222
3. Oshita, M.: Multi-touch interface for character motion control using model-based approach. In: International Conference on Cyberworlds 2013. (2013) 330–337
4. Jakobsen, T.: Advanced character physics. In: Proceedings of Game Developer's Conference 2001. (2001)
5. Popović, Z., Witkin, A.: Physically based motion transformation. In: SIGGRAPH 1999. (1999) 11–20

6. Kulpa, R., Multon, F., Arnaldi, B.: Morphology-independent representation of motions for interactive human-like animation. Computer Graphics Forum (Eurographics 2005) **24**(3) (2005) 343–352
7. Neff, M., Kim, Y.: Interactive editing of motion style using drives and correlations. In: Eurographics/ACM SIGGRAPH Symposium on Computer Animation 2009. (2009) 103–112
8. Krause, M., Herrlich, M., Schwarten, L., Teichert, J., Walther-Franks, B.: Multitouch motion capturing. In: ACM International Conference on Interactive Tabletops and Surfaces 2008. (2008) 2
9. Kipp, M., Nguyen, Q.: Multitouch puppetry: Creating coordinated 3d motion for an articulated arm. In: ACM International Conference on Interactive Tabletops and Surfaces 2010. (2010) 147–156
10. Park, S.I., Shin, H.J., Shin, S.Y.: On-line locomotion generation based on motion blending. In: ACM SIGGRAPH Symposium on Computer Animation 2002. (2002) 105–111
11. Thorne, M., Burke, D., van de Panne, M.: Motion doodles: An interface for sketching character motion. ACM Transactions of Graphics (SIGGRAPH 2004) **23**(3) (2004) 424–431
12. Oshita, M.: Motion control with strokes. Computer Animation and Virtual Worlds **16**(3-4) (2005) 237–244
13. Igarashi, T., Moscovich, T., Hughes, J.F.: Spatial keyframing for performance-driven animation. In: ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2005. (2005) 253–258
14. Dontcheva, M., Yngve, G., Popović, Z.: Layerd acting for character animation. In: SIGGRAPH 2003. (2003) 409–416
15. Callennec, B.L., Boulic, R.: Interactive motion deformation with prioritized constraints. Graphical Models **68**(2) (2006) 175–193
16. Hodgins, J.K., Wooten, W.L., Brogan, D.C., O'Brien, J.F.: Animating human athletes. In: SIGGRAPH '95. (1995) 71–78
17. Faloutsos, P., van de Panne, M., Terzopoulos, D.: Composable controllers for physics-based character animation. In: SIGGRAPH 2001. (2001) 251–260
18. Liu, C.K., Popović, Z.: Synthesis of complex dynamic character motion from simple animations. ACM Transactions on Graphics (SIGGRAPH 2002) **21**(3) (2002) 408–416
19. Jain, S., Ye, Y., Liu, C.K.: Optimization-based interactive motion synthesis. ACM Transactions on Graphics **28**(1) (2009) Article No. 10
20. Macchietto, A., Zordan, V., Shelton, C.R.: Momentum control for balance. ACM Transactions of Graphics (SIGGRAPH 2009) **28**(3) (2009) Article No. 80
21. Monheit, G., Badler, N.I.: A kinematic model of the human spine and torso. IEEE Computer Graphics and Applications **11**(2) (1991) 29–38
22. Tolani, D., Goswami, A., Badler, N.I.: Real-time inverse kinematics techniques for anthropomorphic limbs. Graphical Models and Image Processing **62**(5) (2000) 353–388
23. Yonemoto, S., Arita, D., ichiro Taniguchi, R.: Real-time human motion analysis and ik-based human figure control. In: Workshop on Human Motion 2000. (2000) 149–154
24. Gottschalk, S., Lin, M., Manocha, D.: Obbtree: A hierarchical structure for rapid interference detection. In: SIGGRAPH '96. (1996) 171–180
25. Wu, C.C., Medina, J., Zordan, V.B.: Simple steps for simply stepping. In: International Symposium on Visual Computing (ISVC) 2008. (2008) 97–106